ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"Simulación e Implementación en FPGA de un Esquema de Codificación del Canal sujeto al Estándar de Wimax"

Informe de Proyecto de Graduación

Previo a la Obtención del Título

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentado por:

José Andrés Marzo Icaza

GUAYAQUIL-ECUADOR

2009

DEDICATORIA

Este proyecto está dedicado a mi madre,

Ejemplo a seguir en nuestro hogar

y a la que debo todo mi ser.

A mi abuelo Pepe, por ser mi inspiración siempre.

AGRADECIMIENTO

A Dios, por todas sus bendiciones.

A mi familia, por sus consejos, ayuda y comprensión.

A mis profesores, que impartieron sus conocimientos en toda mi vida universitaria.

DECLARATORIA EXPRESA

“La responsabilidad por los hechos, ideas y doctrinas expuestas en este proyecto de graduación me corresponden exclusivamente; y, el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL”

_____

JOSE ANDRÉS MARZO ICAZA

TRIBUNAL

_____	_____
Ing. Jorge Aragundi	**Ing. Rebeca Estrada**
Sub-Decano de la FIEC	**Director de Proyecto de Graduación**
_____	_____
Dr. Boris Ramos	**Ing. Juan Carlos Aviles**
Vocal	**Vocal**

RESUMEN

El presente trabajo describe el diseño, simulación e implementación en una FPGA de un Codificador de Canal para Wimax, enfocándose en el estándar IEEE 802.16-2004 el cual representa la implementación fija y forma parte de la investigación en el campo de redes de acceso fijo inalámbrico de banda ancha sin línea de vista. El trabajo presenta las principales características usadas en Wimax para la transmisión y recepción además del funcionamiento de cada uno de los bloques usados para la corrección de errores.

Este proyecto presenta una implementación en FPGA usando diseño basado en modelo, usando el software System Generator junto a Matlab y Simulink, para poder obtener los datos que nos permitan comprobar el funcionamiento del diseño propuesto de acuerdo a las especificaciones del estándar de Wimax y además analizar la capacidad de corrección de errores del sistema.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABREVIATURAS

OFDM	Orthogonal Frequency Division Multiplexing
BPSK	Binary Phase Shift Keying
QPSK	Quadrature Phase Shift Keying
QAM	Quaternary Amplitude Modulation
FPGA	Field Programmable Gate Array
FEC	Forward Error Correction
RS-CC	Reed Solomon-Convolutional Coding
SNR	Signal to Noise Ratio
FFT	Fast Fourier Transform
IFFT	Inverse Fast Fourier Transform
WLAN	Wireless Local Area Network
WMAN	Wireless Metropolitan Area Network
MCM	Multi Carrier Modulation
TDM	Time Division Multiplexing
FDM	Frequency Division Multiplexing

TDD	Time Division Duplexing
FDD	Frequency Division Duplexing
MIMO	Multiple Input Multiple Output
PRBS	Pseudorandom Binary Sequence
LFSR	Linear Feedback Shift Register

INTRODUCCIÓN

La cobertura de aplicaciones usando FPGA es muy amplia, siendo una de ellas las comunicaciones inalámbricas. Por lo que gracias a las funcionalidades de las FPGA y a los entornos de desarrollo de aplicaciones usando lógica programable, se pueden realizar varios diseños en esta área.

En este proyecto se hará una implementación para el estándar IEEE 802.16-2004, más conocido como Wimax, y específicamente el diseño se centrará en la etapa de Codificación del Canal. Además de la implementación se harán varias simulaciones que demuestren las ventajas de usar este sistema, analizándose los recursos en la FPGA destinados y que deban ser considerados para proyectos futuros.

El uso del codificador del canal permitirá al sistema tener un mejor desenvolvimiento frente al ruido, el cual será simulado por un canal AWGN, por lo que tendrá una probabilidad de error de bit menor para valores bajos de relación de señal a ruido comparados a su contraparte sin codificación. Así mismo se describirán los distintos niveles de corrección de errores especificados en el estándar y se demostrará mediante simulaciones su funcionalidad y ventajas y desventajas.

En el capítulo 1 se hace una introducción a Wimax, poniendo énfasis en el diseño de la capa física, analizando cada etapa del estándar. Además se analiza la metodología usada para realizar la implementación en la FPGA.

En el capítulo 2 se realiza un análisis teórico de cada etapa usada en la codificación del canal, analizando las ventajas de cada una y los parámetros necesarios para realizar el diseño. El contenido de este capítulo nos ayudara a entender cómo es posible detectar y corregir los errores causados por el canal, describiendo que es lo que debe hacer el codificador y el decodificador para poder recibir la señal que ha pasado por un canal de comunicaciones.

En el capítulo 3 se describe el diseño e implementación del codificador y el decodificador del canal. Dado que existen varios niveles de corrección de errores, se describe cada etapa de una manera general para que pueda ser fácilmente adaptada al diseño. Los recursos necesarios en la FPGA son mostrados para cada etapa, dependiendo el tipo de codificación usado. Para terminar se realiza un diseño de pruebas que deben ser realizadas para confirmar que el diseño cumple las características del estándar de Wimax.

En el capítulo 4 se analizan los resultados obtenidos en cada una de las simulaciones. Los parámetros a tener en cuenta son las curvas BER vs SNR y las constelaciones a la salida del canal para valores extremos de SNR.

Finalmente se detallan las conclusiones y recomendaciones de este diseño, evaluando las funcionalidades de la Codificación del Canal para Wimax

CAPÍTULO I

1. FUNDAMENTOS TEÓRICOS

1.1. Antecedentes y Justificación

Desde que se comenzó a usar sistemas digitales, los sistemas de comunicaciones han ido creciendo para poder brindar cada vez más servicios. La industria de las telecomunicaciones se enfoca cada vez más en servicios como video conferencias o contenido multimedia, y no se limita a redes cableadas, por lo que las redes inalámbricas están en crecimiento continuamente.

Desde que comenzaron a usarse redes WLAN (Wireless local Network), estos han evolucionado para poder soportar cada vez más aplicaciones y tasas de datos superiores. Las primeras redes WLAN usaban Spread Spectrum, y evolucionaron hasta adoptar OFDM. Como consecuencia de

esta evolución nació Wimax (Worldwide Interoperability for Microwave Access), con el estándar IEEE-802.16 el cual define una WMAN (Wireless Metropolitan Area Network). Wimax soporta comunicaciones con o sin línea de vista y resulta una perfecta alternativa para sistemas de Cable, DSL, y T1/E1, por lo que es una solución para sistemas de última milla en lugares donde las tecnologías antes mencionadas tienen costos de implementación y mantenimiento que no justifiquen su inversión.

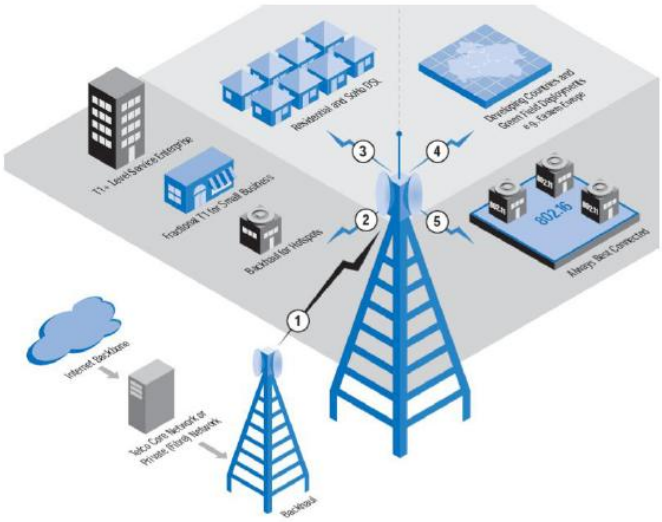

Figura 1.1 Escenarios de Aplicación de Wimax

La figura 1.1 muestra posibles escenarios de aplicación de Wimax.

- Red de Respaldo: Puede funcionar como respaldo para ser usado por varias redes comerciales.
- Residencial: Como una alternativa para DSL o Cable Modem.

- Áreas Rurales: Donde desplegar una red alambrada es muy costosa.
- Movilidad: Permite estar el usuario siempre conectado, y así pasar de una red, por ejemplo IEEE 802.11, a la red de Wimax sin perder en ningún momento la comunicación.

El uso de herramientas programables reconfigurables, como la FPGA, son un buen auxilio para el diseño de sistemas para Wimax. Las ventajas que ofrecen estas son las siguientes:

- **Velocidad de procesamiento:** Wimax, al ser una tecnología de banda ancha, posee requerimientos superiores en lo que se refiere al manejo de las tasas de datos y la velocidad de procesamiento del sistema. La FPGA cumple estos requerimientos.
- **Flexibilidad:** Wimax es una tecnología que está evolucionando, como es el caso de pasar de ambientes fijos a móviles, por lo que es necesario que la FPGA sea flexible para poder reprogramar el diseño hasta obtener el producto final.
- **Mercado:** Al ser Wimax una tecnología emergente, el tiempo que los productos lleguen al mercado es un factor clave. Las herramientas de desarrollo para las FPGA, y la existencia de Cores ya diseñados y optimizados, logran que el tiempo de desarrollo sea menor.

1.2. Introducción al estándar Wimax IEEE-802.16

Wimax ha evolucionado desde el punto de vista comercial y tecnológico, empezó con el estándar original IEEE 802.16 pensado para ofrecer altas tasas de datos para comunicaciones punto a punto con línea de vista. Pero al mercado donde se quería llegar era al usuario final, proveer comunicaciones de alto ancho de banda y donde estaban teniendo problemas muchos proveedores de servicio de Internet, con lo que llegaron los estándares IEEE 802.16a y IEEE 802.16d, más conocido como IEEE 802.16-2004, teniendo mucho éxito en zonas rurales donde no existe una buena tecnología alambrada. Pero Wimax no se quedo ahí, por lo que faltaba dar soporte a la movilidad, la cual es establecida en el estándar IEEE 802.16e (IEEE 802.16-2005).

Las características básicas de cada estándar son presentadas en la tabla 1.1. De aquí se puede notar que existen 3 implementaciones para la capa física que son: Capa física por portadora simple, basada en OFDM y basada en OFDMA. Así mismo hay varios escenarios para la capa MAC, duplexación, bandas de frecuencia, operación, etc; por lo que estos estándares fueron desarrollados para funcionar en varias aplicaciones y conjunto de escenarios.

Tabla 1.1 Características de los estándares de Wimax

	802.16	**802.16-2004**	**802.16-2005**
Completado	Diciembre 2001	Junio 2004	Diciembre 2005
Banda de Frecuencia	10-66 GHz	2,11 GHz	2,11 GHz (fijo) 2-6 GHz (móvil)
Aplicación	Fijo LOS	Fijo NLOS	Fijo y móvil NLOS
Arquitectura MAC	Punto a multipunto mesh	Punto a multipunto mesh	Punto a multipunto mesh
Transmisión	Portadora simple	Portadora simple, 256 OFDM, 2048 OFDM	Portadora simple, 256 OFDM o escalable OFDM con 128, 512 1024, o 2048 subportadoras
Modulación	QPSK, 16-QAM, 64-QAM	QPSK, 16-QAM, 64-QAM	QPSK, 16-QAM, 64-QAM
Tasa de datos	32-134.4 Mbps	1-75 Mbps	1-75 Mbps
Multiplexación	TDM/TDMA	TDM/TDMA/OFDMA	TDM/TDMA/OFDMA
Duplexación	TDD y FDD	TDD y FDD	TDD y FDD
Ancho de banda del canal	20,25,28 MHz	1.75, 3.5, 7, 14, 1.25, 5, 10, 15, 8.75 MHz	1.75, 3.5, 7, 14, 1.25, 5, 10, 15, 8.75 MHz
Interfaz Aire	WirelessMAN-SC	WirelessMAN-SC WirelessMAN-OFDM WirelessMAN-OFDMA WirelessHUMAN	WirelessMAN-SC WirelessMAN-OFDM WirelessMAN-OFDMA WirelessHUMAN
Implementación WIMAX	Ninguna	256-OFDM o Wimax Fijo	OFDMA escalable como Wimax móvil

La multiplexación por división de frecuencia ortogonal, más conocida como OFDM, es parte primordial de los estándares de Wimax, por lo que permite altas tasas de datos y combate el desvanecimiento por multicamino en las comunicaciones inalámbricas. OFDM es efectiva para comunicaciones sin línea de vista (NLOS), debido a su larga duración de símbolo. Esto lo logra al

combinar la modulación multiportadora MCM y modulación por frecuencia ortogonal.

MCM se encarga de pasar la señal de banda ancha en paralelo, a múltiples portadoras de menor ancho de banda, y cada portadora OFDM es ortogonal de las otras portadoras transmitidas.

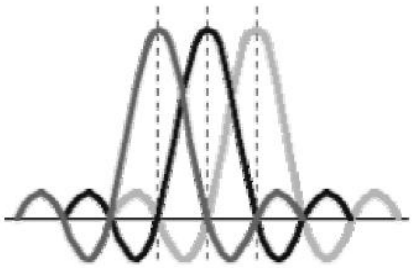

Figura 1.2 Ortogonalidad de OFDM

Además de OFDM, IEEE-802.16 usa modulación adaptativa con el fin de asegurar la calidad de servicio. La modulación adaptativa asegura un alto performance con la mayor tasa de datos sostenible.

Etapas Generales del Estándar

El enfoque de este proyecto se da en el estándar 802.16-2004, por lo que se describirá sus principales características tanto para la capa física como para la capa MAC, atendiendo más la primera que es donde esta descrita la codificación de canal que se va a implementar.

Capa Física

Para realizar este trabajo se considera WirelessMAN-OFDM que utiliza una transformada rápida de Fourier de 256 puntos y opera en la banda de frecuencia de 2 a 11 GHz. El estándar fijo de Wimax provee servicio para un área de 5 Km permitiendo una máxima tasa de datos de 70 Mbps con un ancho de banda del canal de 20MHz, ofreciendo a los usuarios conectividad de banda ancha sin la necesidad de tener línea de vista con la estación base. Las características y beneficios de la capa física son descritos en la tabla 1.2.

Tabla 1.2 Características de la Capa Física de Wimax

Característica	**Beneficio**
256 puntos FFT	Ecualización simple en ambientes LOS y NLOS
Modulación adaptativa y codificación de corrección de errores variable	Asegura un enlace robusto y maximiza la tasa de datos usada por cada suscriptor
Duplexación TDD y FDD	Variable según las regulaciones.
Tamaños de canal flexibles	Flexibilidad para operar en varias bandas de frecuencia.
Soporta DFS (Selección dinámica de frecuencia)	Minimiza interferencia entre canales adyacentes.
Soporta AAS	Las antenas inteligentes ayudan a suprimir interferencia e incrementar la ganancia del sistema.
Uso de TDM y FDM	Permite interoperabilidad entre sistemas celulares (TDM) y sistemas inalámbricos (FDM)
MIMO	Se implementa en el downlink para aumentar diversidad y capacidad.

El tamaño de los puntos de la FFT determina el número de subportadoras. De estas 256 subportadoras, 192 son usadas para datos del usuario, 56 son

datos nulos para la banda de guarda y 8 son usadas como subportadoras pilotos, tal como se puede apreciar en la figura 1.3 que describe OFDM en el dominio de la frecuencia. La capa física acepta una longitud variable del prefijo cíclico de 8, 16, 32 o 64 dependiendo de la dispersión del canal esperado. El prefijo cíclico (CP) lo podemos apreciar en la descripción del símbolo OFDM en el dominio del tiempo en la figura 1.4.

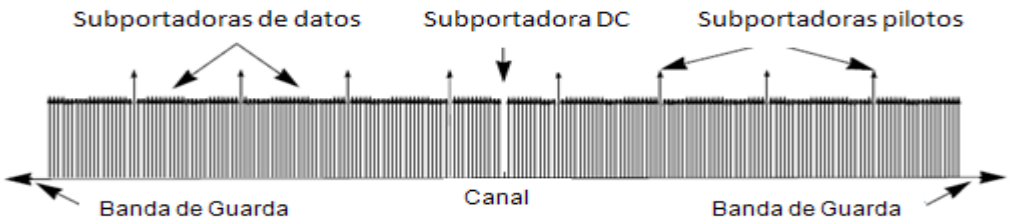

Figura 1.3 Símbolo OFDM en el dominio de la frecuencia

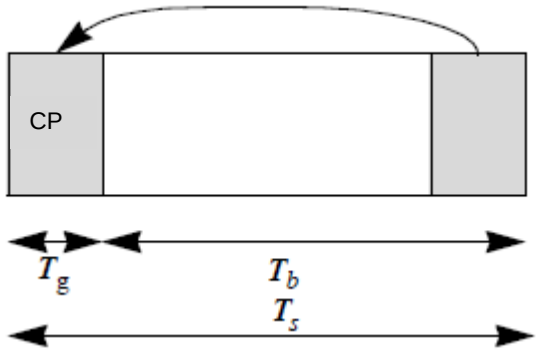

Figura 1.4 Símbolo OFDM en el dominio del tiempo

La corrección de errores (Forward Error Correction, FEC), es realizada en 2 fases. Primero pasando por un codificador exterior Reed-Solomon y luego por un codificador interior Convolucional. El codificador Reed-Solomon se

encarga de corregir errores de ráfaga a nivel de byte, lo cual es bien útil en la presencia de propagación en multicamino. El codificador convolucional corrige errores independientes de bits. Funcionalidades de puncturing se aplican en el codificador convolucional para variar la capacidad de corrección de errores.

La figura 1.5 muestra las modulaciones especificadas tanto para el enlace downlink como el uplink, las cuales son BPSK, QPSK, 16-QAM y 64-QAM. Las opciones de FEC son emparejadas con cada esquema de modulación. El estándar especifica siete combinaciones de modulación y tasa de codificación, las cuales se seleccionan para dar comunicación a cada usuario, lo que es conocida como modulación adaptativa. La tasa de codificación se refiere a la relación de los datos sin codificar por los datos codificados, por lo se tendrá que hacer una decisión si lo que se quiere es tener una tasa de datos alta o una mayor robustez del sistema. La tabla 1.3 muestra las combinaciones descritas.

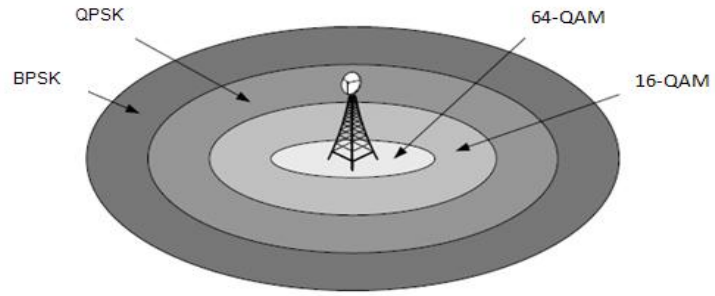

Figura 1.5 Modulación usada para cada radio de celda

Tabla 1.3 Perfiles de modulación y codificación para el estándar IEEE-802.16-2004

ID	Modulación	Tamaño del bloque no codificado (bytes)	Tamaño del bloque codificado (bytes)	Tasa de codificación total
0	**BPSK**	12	24	1/2
1	**QPSK**	24	48	1/2
2	**QPSK**	36	48	3/4
3	**16-QAM**	48	96	1/2
4	**16-QAM**	72	96	3/4
5	**64-QAM**	96	144	2/3
6	**64-QAM**	108	144	3/4

En la figura 1.5 se observa el radio de celda requerida para cada tipo de modulación, donde las de radio más pequeño alcanzan tasas de datos superiores pero al mismo tiempo son más susceptibles al ruido. En periodos donde existen altos desvanecimientos y el canal presenta una baja relación de señal a ruido (SNR), entonces el tamaño de la constelación de la señal es reducido para mejorar la fidelidad del sistema. Así mismo, en periodos de bajo desvanecimiento o alto SNR, el tamaño de la constelación de la señal es aumentado para así ofrecer tasas de datos mucho más altas. Por ejemplo BPSK y QPSK las cuales tienen una mejor performance en presencia de ruido, se usan con un radio de celda mayor y se necesita una relación de señal a ruido (SNR) menor, pero la cantidad de datos que se pueden

transmitir con esas modulaciones son bajas comparado con 16-QAM o 64-QAM.

Capa MAC

A pesar de que la implementación realizada en este trabajo es a nivel de capa física, se menciona las características de esta capa para conocer las características de acceso al medio presente en el estándar de Wimax. Algunas de estas funciones son asociadas con proveer servicio a los suscriptores, transmitiendo los datos en tramas y controlando el acceso al medio inalámbrico compartido.

La capa MAC es diseñada para acomodar múltiples especificaciones de la capa física y servicios, direccionando las necesidades para cada ambiente. Además el diseño está orientado a trabajar con topología de red punto a multipunto, con la estación base controlando múltiples sectores simultáneamente. Así mismo los protocolos MAC definen como y cuando la estación base (BS) o la estación suscriptora (SS) pueden iniciar la transmisión. En el downlink se usa TDM para multiplexar los datos, pero en el uplink, se utiliza TDMA ya que muchas estaciones suscriptoras van a querer acceder al medio.

Sus principales características y ventajas son descritas en la tabla 1.4.

Tabla 1.4 Características de la Capa MAC de Wimax

Características	Beneficios
Uso de TDM/TDMA	Eficiente uso del ancho de banda.
Escalable de uno a cientos de suscriptores	Permite mejorar costos.
Orientado a conexión	QoS por conexión. Enrutamiento y transmisión de paquetes más rápidos.
QoS	Baja latencia. Optimo tráfico de video. Priorización de datos.
Petición de Retransmisión Automática (ARQ)	Mejora el desempeño en el enlace ocultando errores inducidos por protocolos de capa superiores.
Soporte de Modulación adaptativa	Permite la más alta tasa de datos permitida por las condiciones del canal.
Seguridad y encriptación	Protege privacidad del usuario.
Control automático de potencia	Permite despliegues celulares minimizando su propia interferencia.

Transmisión y Recepción

El proceso de transmisión para la capa física de Wimax es presentada en la figura 1.6:

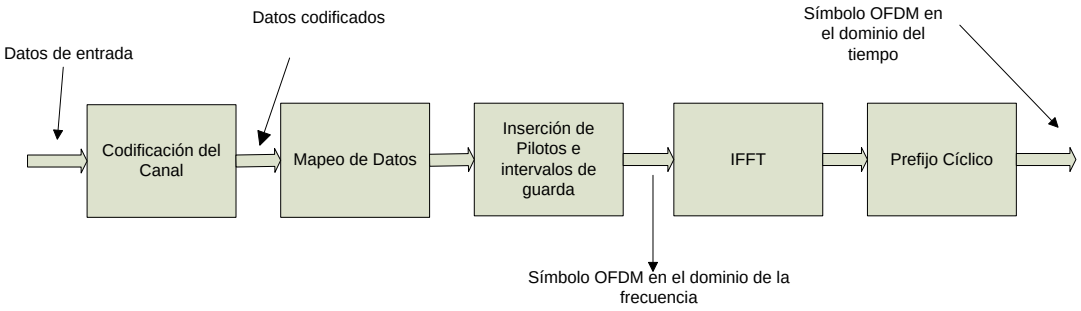

Figura 1.6 Diagrama de bloques para la transmisión

Como se puede observar los datos primero son codificados agregándoles redundancia para luego pasar por la etapa de modulación de los datos antes de pasar por la etapa de generación del símbolo OFDM que incluye la inserción de los pilotos antes de pasar por una transformada rápida de Fourier inversa y se le añade el prefijo cíclico teniendo el símbolo OFDM completo.

El codificador del canal (figura 1.7) se compone de varias etapas, que incluye el randomizer que se encarga de no permitir que queden subportadoras sin modular, lo que se hace logrando que la distribución de ceros y unos sea equiprobable. Luego como los datos MAC se asumen agrupados en bloques, cuyo tamaño debe coincidir con el tamaño del bloque RS menos uno, entonces se agrega un byte cero al final del bloque para permitir que el codificador convolucional termine siempre en el estado cero. A continuación se tiene el codificador exterior implementado usando un codificador de Reed-Solomon, donde los bytes de paridad deben ser enviados al inicio de cada bloque. Luego se agrega en el codificador interior un codificador convolucional que funciona a nivel de bit. Tanto el codificador exterior como interior en conjunto forman un codificador concatenado RS-CC que viene a ser el FEC del sistema. Para terminar, los datos codificados pasan por el interleaver que sirve como soporte al FEC para evitar errores en ráfagas. Las funcionalidades de cada bloque son descritas en la sección 2.1.

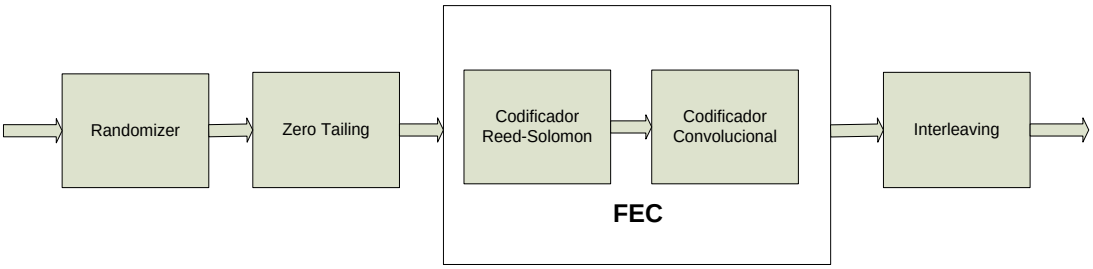

Figura 1.7 Diagrama de bloques de la Codificación del Canal

Luego del codificador del canal, viene el llamado modulador, que se encarga de formar el símbolo OFDM. Para eso se mapean los datos a la salida del codificador de canal como esta descrito en la figura 1.8 donde **c** es el valor al que se debe multiplicar la señal mapeada como método de normalización. Los símbolos generados deben mapearse en las subportadoras de datos. Luego se insertan los pilotos y los intervalos de guarda (256 subportadoras como se describió en la Tabla 1.2 para así crear el símbolo OFDM en el dominio de la frecuencia.

Para crear el símbolo OFDM en el dominio del tiempo, se aplica una IFFT de 256 puntos al símbolo OFDM en el dominio de la frecuencia. Finalmente se añade el prefijo cíclico al inicio del símbolo con la duración especificada.

En lo que se refiere a la recepción de los datos, el estándar no se refiere como debe ser su estructura, solo menciona que su implementación es en forma inversa al transmisor, por lo que existen más libertades para su diseño. Un esquema para la recepción se presenta en la figura 1.9.

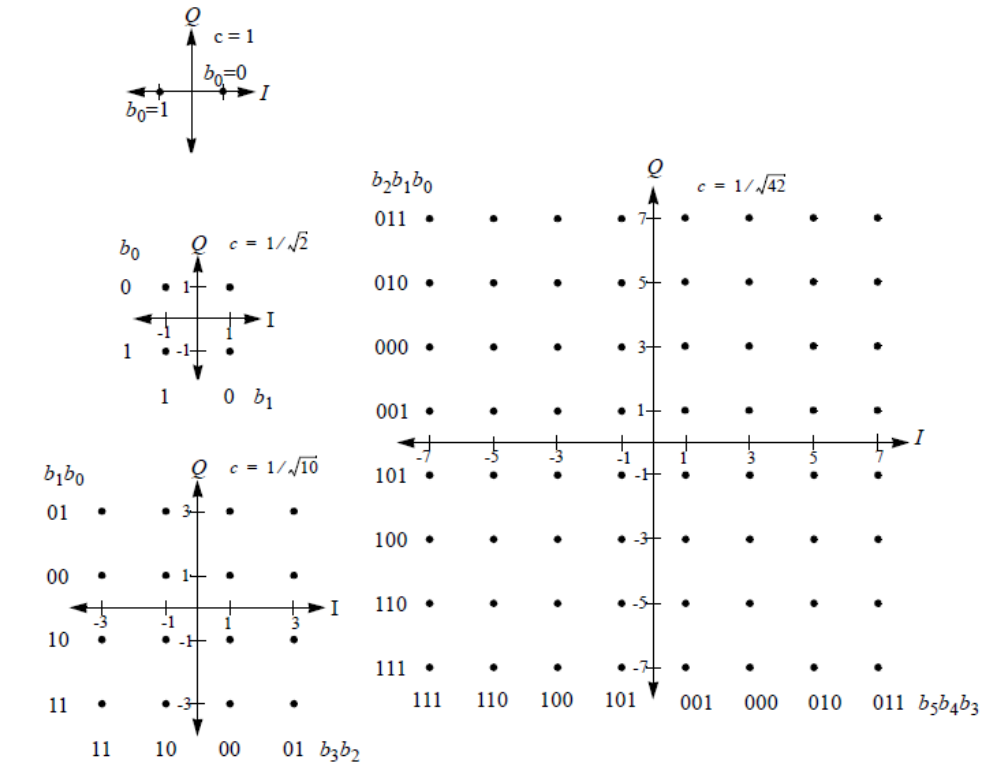

Figura 1.8 Constelaciones usadas en el estándar IEEE 802.16-2004

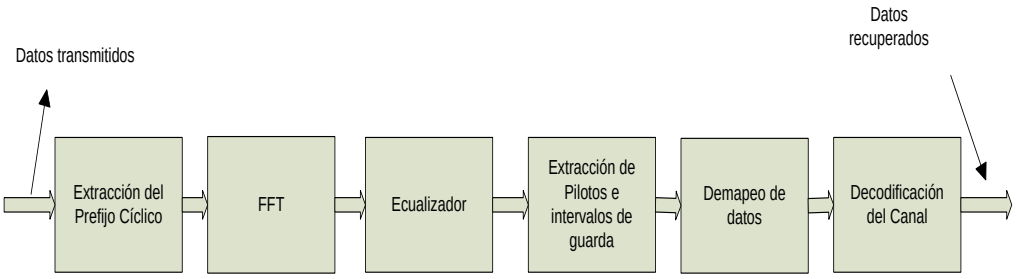

Figura 1.9 Diagrama de bloques para la recepción

La primera parte, encargada de la demodulación de los datos, se encarga de pasar las ráfagas OFDM en el dominio del tiempo a la secuencia de bits que la componen. Se asume que hay sincronismo con el receptor.

A cada símbolo OFDM se le extrae el prefijo cíclico, con lo que se tiene la señal con el correspondiente tiempo útil del símbolo. A continuación se pasa por una FFT de 256 puntos obteniendo el símbolo OFDM en el dominio de la frecuencia. El ecualizador se usa para compensar los efectos del canal ya que cada portadora presenta una atenuación distinta. Luego se extraen los pilotos y señales de guarda para poder así pasar a un detector de símbolo que se encargara del demapeo o demodulación de los símbolos para tenerlos a nivel de bits.

La etapa de decodificación del canal funciona en sentido inverso al codificador donde para la implementación realizada se usa el diagrama de bloques de la figura 1.10.

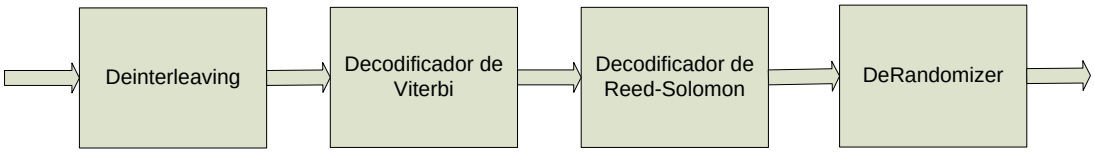

Figura 1.10 Diagrama de bloques del Decodificador de Canal

El decodificador de Viterbi se usa para recuperar los datos que pasaron por el codificador convolucional. Luego de realizar esta codificación los datos pasan por el decodificador de Reed-Solomon que se encarga de decodificar

los datos que no pudieron ser recuperados usando el decodificador de Viterbi. Tanto el deinterleaving como el DeRandomizer son bloques idénticos a los del codificador con ligeras modificaciones. Las funcionalidades de cada bloque son descritas en el capítulo 2.

1.3. Entorno de Diseño y Metodología

Para llegar a la implementación del sistema se sigue el diseño basado en modelo a nivel de sistema, donde se siguen 2 tendencias: utilizar lenguajes de alto nivel y utilizar entornos visuales para el flujo de datos. Los lenguajes de alto nivel se los usan especialmente para el modelado de algoritmos, pero no son adecuados para implementar sistemas con flujos de datos continuos. En cambio, utilizando un entorno visual para flujo de datos, se puede modelar de forma natural los flujos de datos que son los que se utilizan en sistemas DSP.

El entorno escogido para realizar la implementación de este proyecto, es el software System Generator de Xilinx en conjunto con Matlab y Simulink. Además, para realizar la experimentación en hardware, se utiliza la tarjeta de prototipado de Digilent Spartan 3E-Starter, la cual tiene la FPGA donde se va a implementar el diseño y donde se harán todas las pruebas hasta probar su funcionamiento.

El System Generator utiliza Matlab y Simulink para representar un nivel alto de abstracción del sistema a modelar. Además permite la generación del código VHDL de forma eficiente, permitiendo diseños adecuados para modelos DSP. La implementación se la puede realizar por medio de las librerías propias del programa, así como integrando código VHDL o utilizando el software AccelDSP de Xilinx. Así mismo se cuenta con IP Cores (componentes con propiedad intelectual) que proporcionan una funcionalidad desde operaciones aritméticas hasta complejos algoritmos de DSP.

Simulink proporciona además herramientas para la verificación y puesta a punto del sistema, utilizando estímulos que son convertidos de punto flotante a punto fijo para trabajar en hardware y de visualizadores donde se sigue el proceso inverso, convertir los datos de punto fijo a punto flotante para que puedan ser graficados.

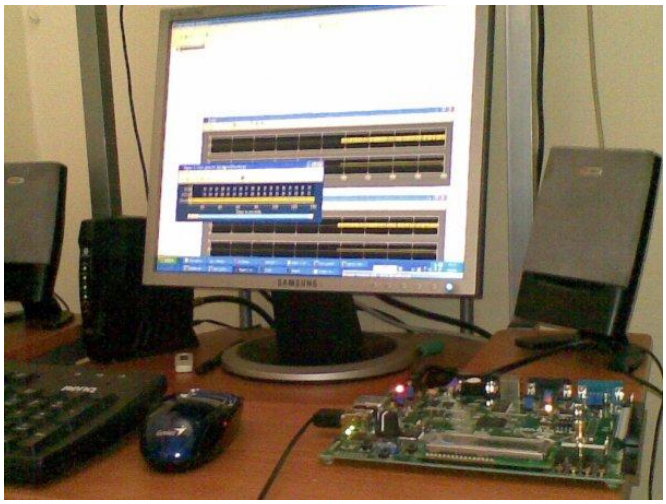

Figura 1.11 Tarjeta Spartan 3E- Starter

La tarjeta Spartan3E-Starter (figura 1.11), contiene una FPGA Spartan3E XC3S500E-4FG320C, un oscilador de cristal usado como reloj interno de 50 MHz, y memoria FLASH y RAM. Además tiene puertos de entrada y salida para ingresar señales que pueden ser procesados por convertidores ADC/DAC.

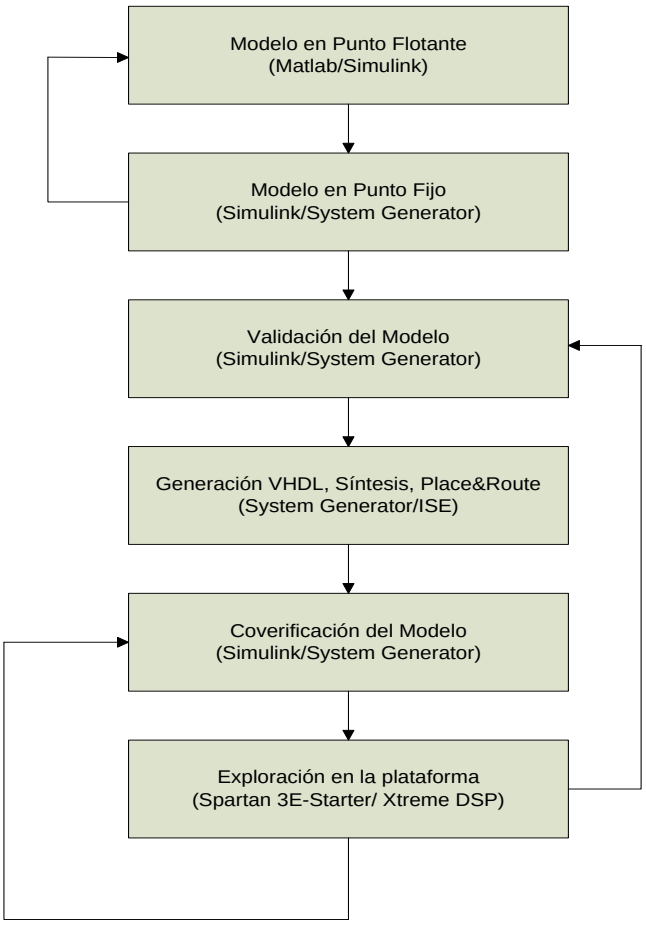

Figura 1.12 Metodología usada en el diseño basado en modelo

La metodología usada es explicada con la figura 1.12 (12). El primer paso es tener un modelo en punto flotante usando Matlab o Simulink. En este caso es preferible usar Simulink ya que nos permite revisar como van cambiando los datos continuamente, dejando Matlab para secciones concretas del diseño. Este paso puede que no sea necesario y se salte enseguida a la generación del modelo en punto fijo, sin embargo es una buena práctica por lo que de esta manera podemos revisar si el algoritmo que queremos implementar coincide con la teoría.

El siguiente paso es generar el modelo en punto fijo, para esto se usan las librerías propias de System Generator en la plataforma de Simulink. Los bloques que se van a usar son diseñados especialmente para la generación de hardware. Así mismo se pueden agregar bloques HDL propios o IP cores.

Una vez que se tiene el sistema validado y simulado y se comprueba su funcionamiento se procede a la generación del Hardware, es decir a la generación del código VHDL. Ya con el código VHDL generado se puede verificar el modelo a este nivel usando simuladores específicos como ModelSim o volver a subir el modelo VHDL y realizar una cosimulación (figura 1.13). Si el resultado no es el esperado se puede modificar el código VHDL directamente o modificar el modelo en Simulink y System Generator (9).

El modelo corriendo sobre la plataforma usando la coverificación o cosimulación, es el último paso donde ya se está probando la funcionalidad en hardware. La cosimulación permite importar el código VHDL a la FPGA y crear un bloque que es agregado al diseño del modelo en System Generator y se realiza una simulación conjunta donde el bloque insertado debe funcionar de manera similar al modelo verificado con anterioridad.

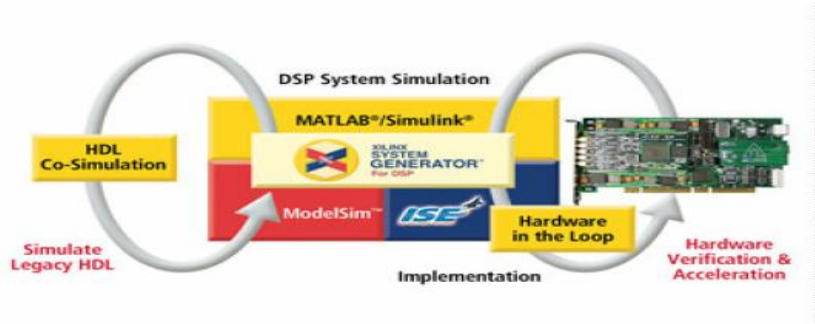

Figura 1.13 Verificación del diseño en hardware

CAPÍTULO II

2. CODIFICACIÓN Y DECODIFICACIÓN DEL CANAL

2.1. Codificación

El proceso de codificación del canal se encarga de tener una transmisión fiable en la comunicación de los datos, protegiendo a estos de degradaciones causadas por el canal. Para lograrlo se añade información redundante a los datos transmitidos de manera que el decodificador pueda usarlos para corregir los errores causados por medio en que son transmitidos.

Existen varios métodos para implementar un codificador del canal, donde sus componentes principales pueden ser un ARQ (Automatic Repeat

Request), el cual envía una solicitud de reenvió, o un FEC (Forward Error Correction), el cual detecta y corrige la información recibida.

El estándar IEEE 802.16-2004 especifica cuatro etapas para el codificador del canal: El Randomizer, el codificador de Reed-Solomon, el codificador convolucional y un interleaver.

2.1.1. Randomizer

El proceso de aleatorización, se encarga de evitar que en los datos de entrada existan largas secuencias de unos y ceros. Esto se logra por un generador de datos pseudoaleatorios (PRBS) el cual es implementado utilizando un registro de desplazamiento con retroalimentación lineal (LFSR).

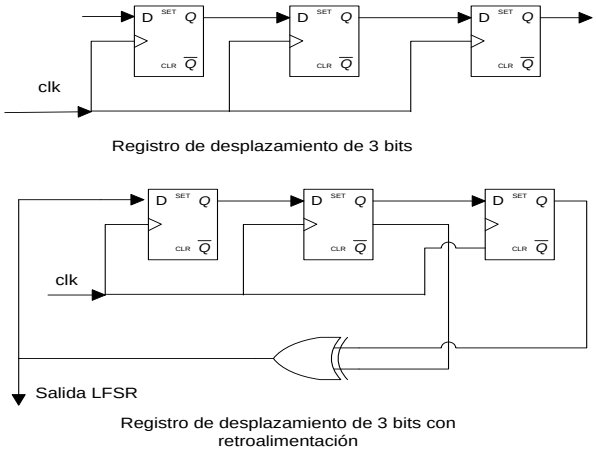

Figura 2.1 Implementación LFSR

El LFSR es un registro de desplazamiento que de manera sincrónica hace avanzar la señal a través de los registros comenzando con el bit más significativo. Algunas salidas son combinadas usando una puerta XOR en forma de retroalimentación. El contenido inicial de cada registro dependerá del valor de semilla especificado. En la figura 2.1 se muestra una implementación de LFSR:

El número de registros en el LFSR y la forma en que es usada la retroalimentación viene dado por la siguiente tabla:

Tabla 2.1 Polinomios de retroalimentación para LFSR (9)

Bits(Número de registros)	**Polinomio de retroalimentación**
4	$x^4 + x^3 + 1$
5	$x^5 + x^3 + 1$
6	$x^6 + x^5 + 1$
7	$x^7 + x^6 + 1$
8	$x^8 + x^6 + x^5 + x^4 + 1$
9	$x^9 + x^5 + 1$
10	$x^{10} + x^7 + 1$
11	$x^{11} + x^9 + 1$
12	$x^{12} + x^{11} + x^{10} + x^4 + 1$
13	$x^{13} + x^{12} + x^{11} + x^8 + 1$
14	$x^{14} + x^{13} + x^{12} + x^2 + 1$
15	$x^{15} + x^{14} + 1$
16	$x^{16} + x^{14} + x^{13} + x^{11} + 1$
17	$x^{17} + x^{14} + 1$
18	$x^{18} + x^{11} + 1$
19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$

Para completar el proceso de aleatorización se utiliza una puerta XOR cuyas entradas son la salida del LFSR y los datos entrantes en forma de bits.

Randomizer usado en el estándar 802.16-2004

La aleatorización de los datos es realizada en cada ráfaga de datos tanto en el downlink como en el uplink, esto quiere decir que es realizado en cada bloque de datos en forma independiente (subcanales en el dominio de la frecuencia y símbolos OFDM en el dominio del tiempo). Si la cantidad de datos a ser transmitidos no concuerda con la longitud del bloque, entonces se lo rellena con 0xFF (solo 1's) al final del bloque de transmisión. Dado que se va a realizar codificación concatenada RS-CC, se tiene que agregar un byte 0x00 al final del bloque para permitir que el codificador convolucional se inicialice en el estado cero. Finalmente el registro de desplazamiento del randomizer debe ser inicializado al transmitir cada bloque.

El PRBS es generado usando el polinomio generador $1 + x^{14} + x^{15}$ como es mostrado en la figura 2.2. Cada byte transmitido debe entrar secuencialmente al randomizer, comenzando con el bit más significativo. Los preámbulos no son pasados por el randomizer. El valor de semilla debe ser usado dependiendo si es en el downlink o el uplink. Estos datos generados son combinados usando la operación XOR con la cadena de bits que entran serialmente.

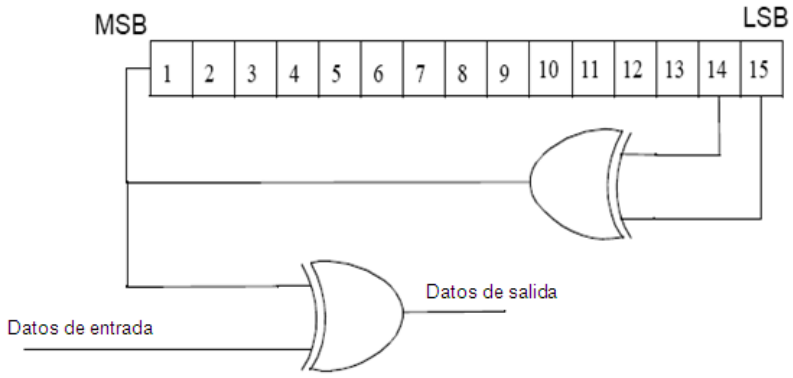

Figura 2.2 Randomizer usado en IEEE 802.16-2004

En el downlink, el randomizer debe ser re-inicializado al inicio de cada trama con la secuencia 100101010000000. El randomizer no debe ser inicializado al inicio de la primera ráfaga. Al inicio de las siguientes ráfagas de datos, el randomizer debe ser inicializado con el vector mostrado en la figura 2.3. El número de trama usado en la inicialización se refiere a la trama en la que la ráfaga de datos es transmitido.

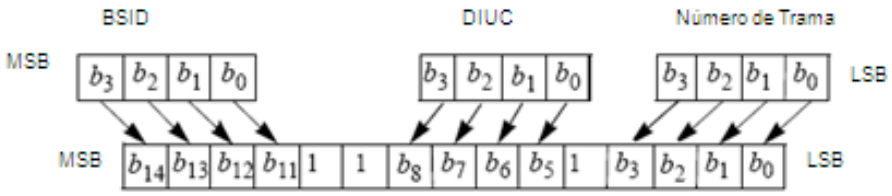

Figura 2.3 Vector de inicialización DL Randomizer OFDM

En el uplink, el randomizer es inicializado por el vector mostrado en la figura 2.4.

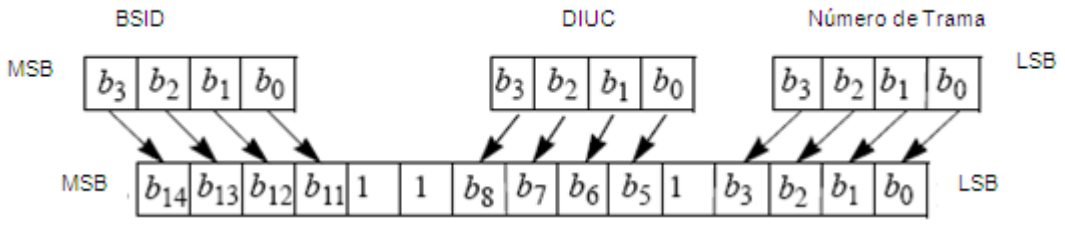

Figura 2.4 Vector de inicialización UL Randomizer OFDM

2.1.2. Codificador de Reed-Solomon

El código Reed-Solomon es un código corrector de errores, subconjunto de los códigos BCH (Bose and Ray-Chaudhuri), en donde el codificador le agrega redundancia a un bloque de símbolos dado. Son por lo tanto unos códigos cíclicos con parámetros (n,k,t), donde k es el numero se símbolos de entrada, n el tamaño del código total y t es el número de errores que pueden ser corregidos. Cada símbolo tiene una longitud de s bits, por lo que existen n-k símbolos de paridad de s bits cada uno y se puede expresar la relación de tamaño del bloque de entrada y salida del codificador con el número de errores que puede corregir el decodificador con la siguiente ecuación y expresada en la figura 2.5:

$$2t = n - k \quad \textbf{(2.1)}$$

$$(n, k, t) = (2^s - 1, 2^s - 1 - 2t, t) \quad \textbf{(2.2)}$$

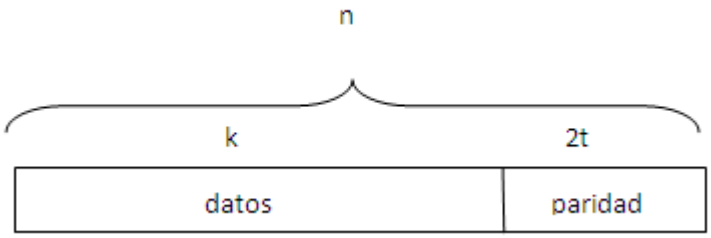

Figura 2.5 Bloque Codificado Reed-Solomon

La figura 2.5 muestra una palabra del codificador Reed- Solomon el cual tiene la característica de ser sistemático, es decir los datos no se alteran y los símbolos redundantes llamados también símbolos de paridad son colocados al final.

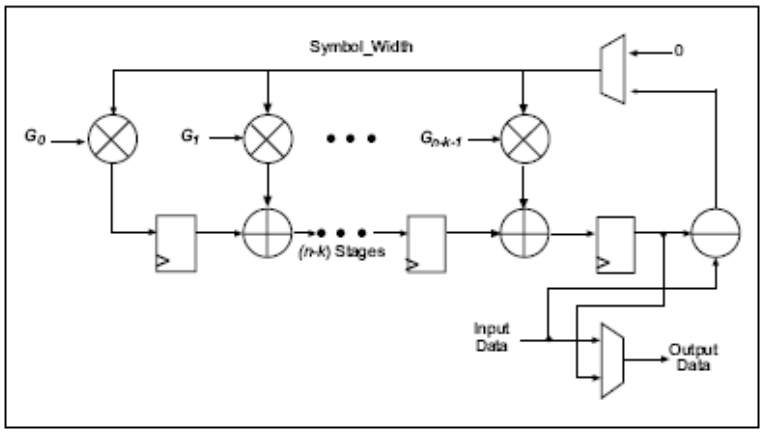

Figura 2.6 Implementación de un codificador Reed-Solomon

Para realizar esta codificación es necesario procesar los datos usando los fundamentos de los Campos Finitos de Galois mediante el circuito digital expresado en la figura 2.6.

Como primer paso es necesario definir el Campo de Galois para la codificación, el cual está definido en función del ancho del símbolo es decir como $GF(2^s)$. Para definirlo se utiliza el polinomio primitivo o también llamado polinomio generador del campo. Los polinomios dependen del ancho del símbolo y se expresan según la tabla 2.2:

Tabla 2.2 Polinomios primitivos en función del ancho del símbolo (4)

m		**m**	
3	$1 + x + x^3$	14	$1 + x + x^6 + x^{10} + x^{14}$
4	$1 + x + x^4$	15	$1 + x + x^{15}$
5	$1 + x^2 + x^5$	16	$1 + x + x^3 + x^{12} + x^{16}$
6	$1 + x + x^6$	17	$1 + x^3 + x^{17}$
7	$1 + x^3 + x^7$	18	$1 + x^7 + x^{18}$
8	$1 + x^2 + x^3 + x^4 + x^8$	19	$1 + x + x^2 + x^5 + x^{19}$
9	$1 + x^4 + x^9$	20	$1 + x^3 + x^{20}$
10	$1 + x^3 + x^{10}$	21	$1 + x^2 + x^{21}$
11	$1 + x^2 + x^{11}$	22	$1 + x + x^{22}$
12	$1 + x + x^4 + x^6 + x^{12}$	23	$1 + x^5 + x^{23}$
13	$1 + x + x^3 + x^4 + x^{13}$	24	$1 + x + x^2 + x^7 + x^{24}$

Las bases teóricas del funcionamiento del codificador esta dado por el polinomio generador del código en su forma general, el cual nos sirve para calcular los símbolos de paridad.

$$g(x) = \prod_{i=0}^{n-k-1}(x - \alpha^{hx(Generator\_Start+i)}) \quad \textbf{(2.3)}$$

Donde h es un factor de escala típicamente en 1 y Generator_Start es un valor que puede alterar el valor de la potencia de α definida por el polinomio primitivo**.** Al expandir el polinomio se obtiene la ecuación:

$$g(x) = g_0 + g_1x + g_2x^2 + \ ... + g_{2t-1}x^{2t-1} + \ x^{2t} \quad \textbf{(2.4)}$$

El grado de este polinomio generador es igual es igual al número de símbolos de paridad. Dado que es de grado 2t, tiene que haber 2t sucesivas potencias de α que son las raíces del polinomio primitivo.

Los códigos de Reed-Solomon son efectivos para ráfagas de ruido. Estas ráfagas pueden tener una longitud variable y afectar a más de un bit, y el decodificador podrá recuperar todo el símbolo alterado, lo que le da una ventaja en comparación con los códigos convolucionales, donde errores de ráfaga pueden afectar el desenvolvimiento del decodificador de Viterbi, además de que la teoría de los códigos de bloque esta mucho más desarrollada.

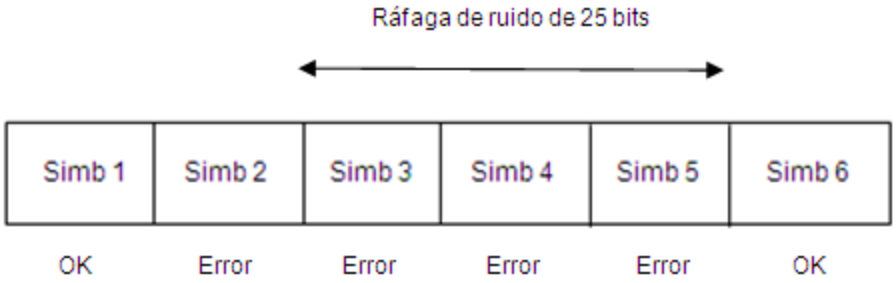

Figura 2.7 Ráfagas de ruido

Shortening y Puncturing

Para hacer el codificador de Reed-Solomon más flexible, es decir, permitir tamaño de bloques variables y capacidades de corrección de error distintas cada bloque pasa por un proceso de shortening (acortamiento) y puncturing (perforación). Al aplicar esta técnica nos evitamos el tener que usar el codificador más de una vez en el diseño, siendo el codificador de Reed-Solomon llamado código madre o código nativo.

Para lograr esto, definimos nuevas variables que son (n',k',t') que son los nuevos valores de tamaño del bloque a la salida y entrada del codificador respectivamente y t' la nueva capacidad de corrección del error.

El proceso es el siguiente:

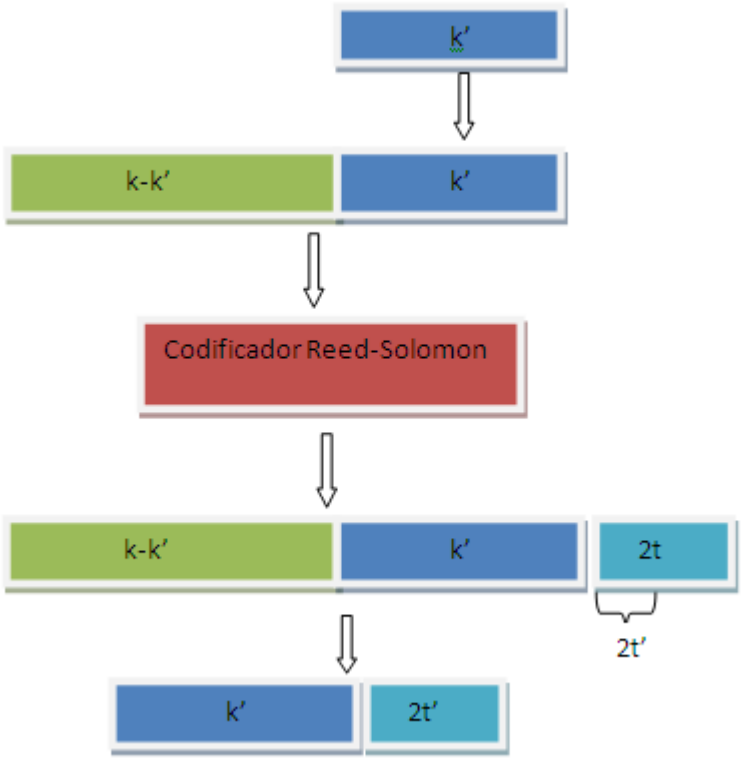

Figura 2.8 Operación del Shortening y Puncturing

1. Se agregan k-k' símbolos de valor cero antecediendo a los k' símbolos que tenemos de entrada. Este es el proceso de shortening (acortar) y es el que permite longitudes de bloques variables.
2. Este nuevo bloque que sigue manteniendo la longitud original k, es pasada por el codificador de Reed-Solomon.
3. Tenemos ya el bloque codificado de longitud n. Se extraen los k-k' símbolos que están al inicio.

4. Por último solo se mantienen los 2t' primeros símbolos de paridad generados, eliminando el resto. Este es el proceso de puncturing el cual permite variar el número de errores que pueden ser corregidos por el decodificador.

Este proceso se muestra en la figura 2.8.

De esta manera también estamos cambiando el valor de la tasa de codificación, lo que nos ayuda a ahorrar el ancho de banda que tenemos disponible en el canal.

Codificación de Reed-Solomon en base al estándar 802.16-2004

La codificación de Reed-Solomon debe ser derivada de un proceso de codificación sistematica siguiendo los siguientes valores (255,239,8) y un campo de Galois GF(2^8)

- **n**=255 (Número total de bytes después de la codificación)
- **k**=239 (Número de bytes antes de la codificación)
- **t**=8 (Numero de bytes que pueden ser corregidos)

Los siguientes polinomios son usados para la codificación:

Polinomio generador del código:

$$g(x) = (x - \alpha^0)(x - \alpha^1)(x - \alpha^2) \dots (x - \alpha^{2t-1});\ \alpha = 02_{HEX} \qquad \textbf{(2.5)}$$

Polinomio generador del campo

$$p(x) = x^8 + x^4 + x^3 + x^2 + 1 \quad \textbf{(2.6)}$$

Entonces multiplicando los elementos de g(x) se obtiene:

$$g(x) = x^{16} + \alpha^{120}x^{15} + \alpha^{104}x^{14} + \alpha^{107}x^{13} + \alpha^{109}x^{12} + \alpha^{102}x^{11} + \alpha^{161}x^{10} + \alpha^{76}x^9 + \alpha^3x^8 + \alpha^{91}x^7 + \alpha^{191}x^6 + \alpha^{147}x^5 + +\alpha^{194}x^2 + \alpha^{255}x + \alpha^{120} \quad \textbf{(2.7)}$$

Este codificador soporta todas las tasas de codificación y los requerimientos de corrección de error estipulados en el estándar de Wimax. Para lograrlo, este código es recortado (shortening) y perforado (puncturing) haciendo que los parámetros antes mencionados sean variables. Cuando se aplica shortening a los k' bytes de entrada, se añade 239-k' bytes ceros como prefijo. Después de la codificación se descartan estos 239-k' bytes. Cuando se aplica puncturing a la palabra codificada para permitir que t' bytes puedan ser corregidos, solamente los primeros 2t' bytes de paridad de los 16 en total serán empleados.

En la tabla 2.3 se muestran las variaciones hechas en el estándar al código nativo (255,239,8).

Tabla 2.3 Codificador Reed-Solomon usado en IEEE 802.16-2004

id	***Bloque de entrada k'***	***Bloque de salida n'***	***Símbolos que pueden ser corregidos t'***
1	32	24	4
2	40	36	2
3	64	48	8
4	80	72	4
5	108	96	6
6	120	108	6

2.1.3. Codificador Convolucional

Los códigos convolucionales son comúnmente expresados por los siguientes parámetros:

n= Número de bits de salida

k= Número de bits de entrada

m=Numero de registros de memoria

Con los valores mencionados tenemos el valor k/n que es llamada la tasa de codificación, cuyo valor esta en el rango de 1/8 a 7/8. Al codificador se lo puede también expresar por los parámetros (n,k,K) donde K es el *constraint length* (longitud de restricción) que representa el número de bits en la memoria del codificador que afectan la generación de los n bits de salida y está definido por:

$$K=k(m-1) \qquad \textbf{(2.8)}$$

Además, dada la naturaleza del codificador convolucional de ser secuencial, es decir, además del bit de entrada, depende se los valores presentes en los registros, por lo que el diseño tendrá un cierto número de estados que están expresados por la siguiente ecuación:

$$Número\ de\ estados = 2^K \qquad \textbf{(2.9)}$$

La estructura del codificador convolucional es fácil de expresar gráficamente por sus parámetros como se observa en la figura:

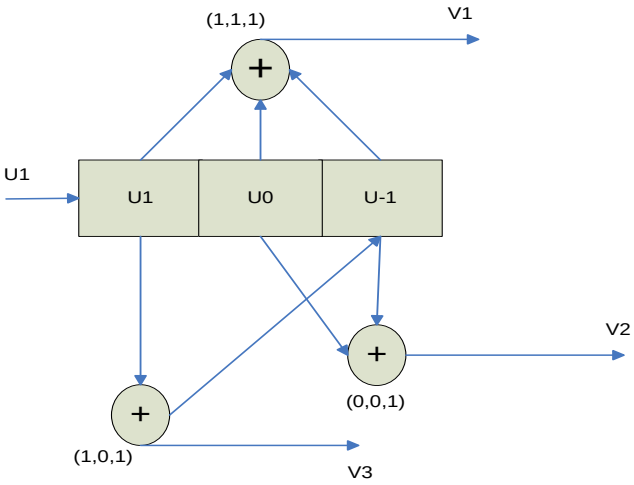

Figura 2.9 Codificador Convolucional

Este es un codificador de tasa 1/3. Cada bit de entrada es codificado con 3 bits a la salida. Podemos expresar el codificador como (3,1,2). Los bits de

salida son generados con los valores de los registros de memoria usando tres sumadores de modulo-2 cuyas entradas dependerán del polinomio generador, cuyo número dependerá del número de bits de salida n. Los polinomios dan al codificador una capacidad única de corrección, es decir variara dependiendo el polinomio escogido en el diseño.

La secuencia saliente v, puede ser calculado convolucionando la secuencia de entrada con su respuesta al impuso g, dado por la ecuación:

$$v = u * g \qquad \textbf{(2.10)}$$

O en una forma más genérica:

$$v_l^j = \sum_{i=0}^{m} u_{l-i} g_i^j \qquad \textbf{(2.11)}$$

Donde v_l^j es el bit de salida l del codificador j, u_{l-i} es el bit de entrada, y g_i^j es el i_{th} término en el polinomio j.

La cantidad de errores que pueden ser corregidos usando codificación convolucional viene dado por la distancia libre del código y la capacidad de corrección de errores por la variable t, por lo que se podrá resolver una cantidad de t errores en una secuencia de nK bits.

$$t = \frac{d_{free}-1}{2} \qquad \textbf{(2.12)}$$

Para representar el diseño del codificador tenemos 3 métodos gráficos:

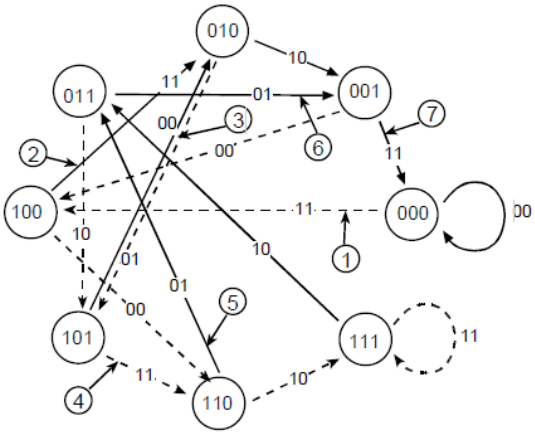

Figura 2.10 Diagrama de estados

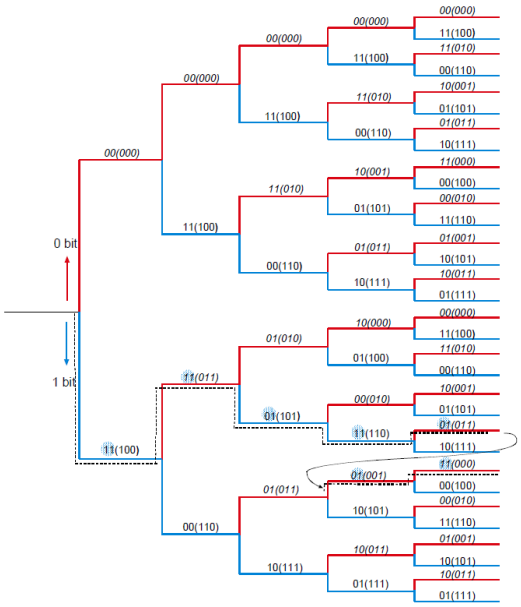

Figura 2.11 Diagrama de Árbol

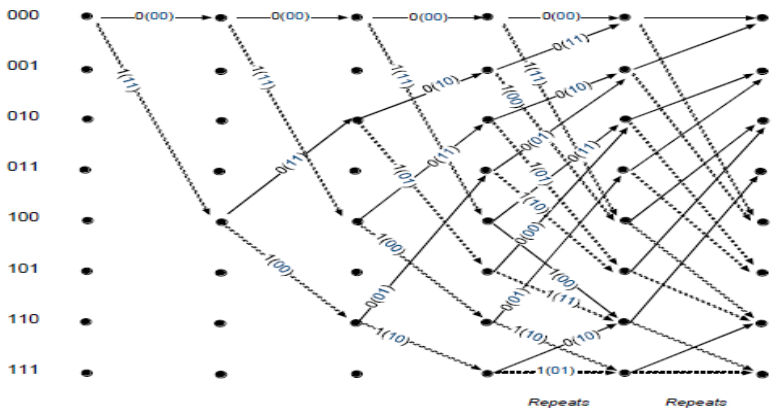

Figura 2.12 Diagrama de Trellis

De los tres diagramas el más difícil de entender es el de Trellis, pero al mismo tiempo es preferible porque representa linealmente en el tiempo la secuencia de eventos presentes en el codificador. El eje-x representa el tiempo en que transcurren los eventos y el eje-*y* muestra todos los posibles estados. Hay 2 posibilidades en cada estado y está definido por el bit de entrada, 0 o 1. Los bits de salida están representados entre paréntesis.

Puncturing

El puncturing es el proceso de borrar sistemáticamente bits de una secuencia de datos, con el fin de reducir la cantidad de datos que son transmitidos y de esta manera pasar de un sistema de baja tasa de datos a una alta tasa de codificación. Se lo usa en conjunto con el codificador convolucional como muestra el grafico:

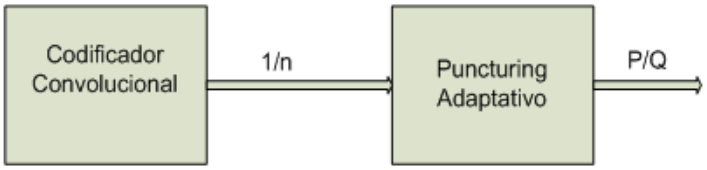

Figura 2.13 Codificador Convolucional con Puncturing

Los bits a la salida del codificador convolucional son eliminados de acuerdo a un vector de perforación. Los elementos de este vector pueden ser unos o ceros, donde un '1' indica transmitir el bit y el '0' eliminarlo.

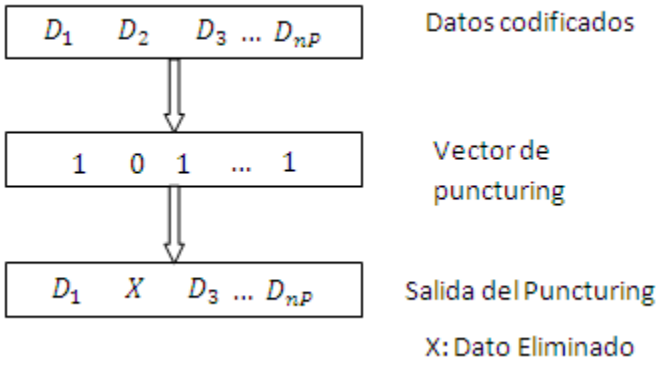

Figura 2.14 Operación del Puncturing

Una tasa P/Q de codificación para un codificador convolucional con puncturing puede ser obtenido de un código convolucional nativo 1/n eliminando los nP-Q bits para cada nP bits codificados. Entonces si por ejemplo quisiéramos tener una tasa de codificación de ¾ para un código nativo ½ tendríamos que agrupar 6 bits que entraran al sistema de puncturing y se deberán eliminar 2 bits de esa secuencia.

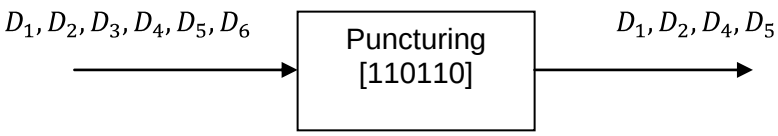

Figura 2.15 Puncturing usando un vector de perforación

Existen varios vectores de puncturing para soportar varios niveles de corrección de errores, los cuales dependerán de las características del estándar.

Codificador Convolucional de acuerdo al estándar 802.16-2004

Para el codificador interior del codificador concatenado RS-CC se utiliza el codificador convolucional analizado con anterioridad que debe tener una tasa de código nativo de ½ y una longitud de restricción igual a 7. Para generar las salidas se usan los siguientes polinomios generadores:

$$G_1 = 171_{oct} \quad para\ X \qquad \textbf{(2.13)}$$

$$G_1 = 133_{oct} \quad para\ Y \qquad \textbf{(2.14)}$$

Los patrones de puncturing y el orden de serialización que deben ser usados para realizar los diferentes tasas de codificación son especificadas en la tabla

2.4. En la tabla ‘1’ significa transmitir bit y ‘0’ remover el bit. X y Y hacen referencia a la figura 2.16.

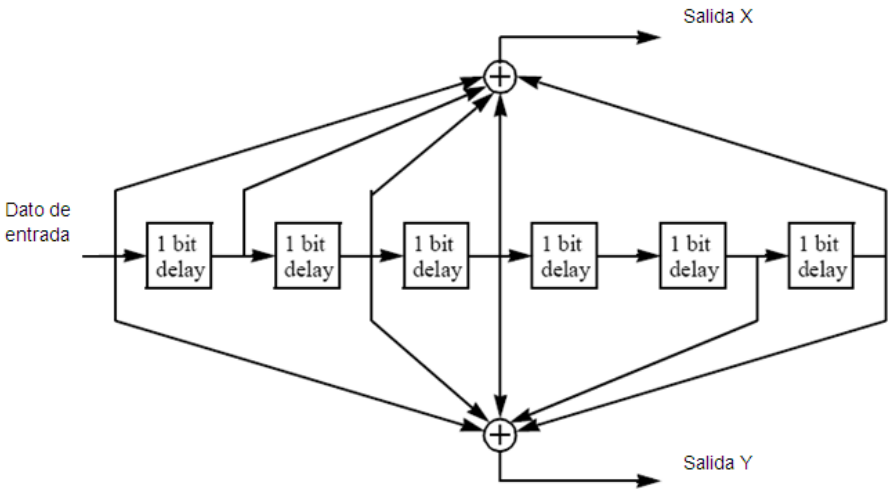

Figura 2.16 Codificador Convolucional usado en el estándar IEEE 802.16-2004

Tabla 2.4 Vectores de Puncturing usados con el codificador convolucional

	Tasas de Codificación			
Tasa	$^{1}/_{2}$	$^{2}/_{3}$	$^{3}/_{4}$	$^{5}/_{6}$
d_{free}	10	6	5	4
X	1	10	101	10101
Y	1	11	110	11010
XY	X_1Y_1	$X_1Y_1Y_2$	$X_1Y_1Y_2X_3$	$X_1Y_1Y_2X_3Y_4X_5$

2.1.4. Interleaving

El interleaver (entrelazador) se encarga de alterar el orden de una secuencia de símbolos de entrada, donde un símbolo es un conjunto de bits que puede ser usado tanto en forma de bit como de bus de datos.

El propósito de usar el interleaver es el de aleatorizar la posición en que se localizan los errores en la transmisión de una señal y así aumentar la eficiencia del FEC dispersando los errores de ráfaga introducidos en el canal de comunicaciones.

Existen varias implementaciones del interleaver entre las que se destacan el interleaver convolucional de Forney y el interleaver por bloque, del cual nos vamos a enfocar.

El interleaver de bloque funciona guardando un bloque da datos de entrada dado en una matriz mxn. La escritura se realiza fila por fila, y una vez guardado todos los datos del bloque se realiza la lectura pero esta vez columna por columna.

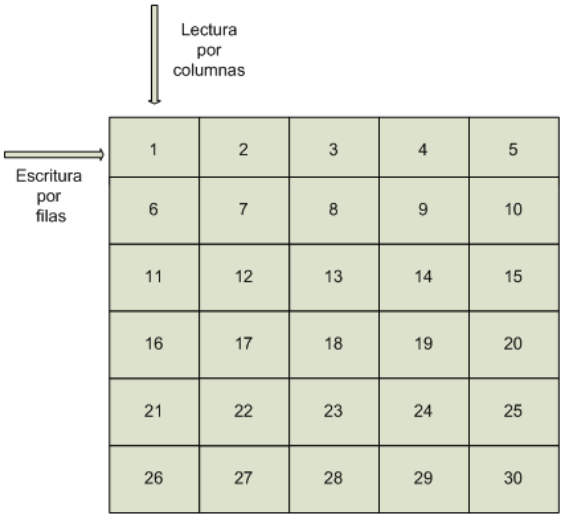

Figura 2.17 Interleaver en su forma matricial

Como ejemplo se presenta un bloque de entrada de 30 símbolos donde el proceso de interleaving se realiza por una matriz 6x5 como se muestra en la figura 2.17. La escritura se realizara con la posición de los datos en el bloque {1, 2, 3,…30} y la lectura se realizara con los datos guardados en las posiciones {1, 6, 11, 16, 21, 26, 2, 7, 12, 17, 22, 27, 3, 8, 13,18, 23, 28, 4, 9, 14, 19, 24, 29, 5, 10, 15, 20, 25, 30}.

Interleaver usado en el estándar 802.16-2004

El número de bits de los datos codificados que deben pasar por el interleaver de bloque debe ser igual al número de bits alojados por cada subcanal del símbolo OFDM, y se lo representa como N_{cbps}. El reordenamiento de los datos se lo realiza por medio de 2 permutaciones. La primera permutación asegura que los bits codificados sean mapeados en subportadoras no adyacentes. La segunda permutación asegura que los bits codificados sean mapeados alternativamente en el menor y más significativo bit de la constelación, así evitando largas secuencias de bits poco fiables.

N_{cpc} es el número de bits codificados por subportadora que son los que corresponden a BPSK (1 bit), QPSK (2 bits), 16QAM (4 bits) y 64QAM (6 bits). De este valor obtenemos el valor de s definido como $s = ceil(N_{cpc}/2)$. Teniendo un bloque de N_{cbps} bits de transmisión, k es el índice de los bits codificados antes de la primera permutación, m_k los bits

codificados después de la primera y antes de la segunda permutación y j_k el índice de después de la segunda permutación, justo antes de la modulación de los datos. Con estos datos se definen las 2 ecuaciones:

$$m_k = \left(\frac{N_{cbps}}{12}\right).k_{mod12} + floor\left(\frac{k}{12}\right) \qquad k = 0,1,\dots,N_{cbps}-1 \quad \textbf{(2.15)}$$

$$j_k = s.floor\left(\frac{m_k}{s}\right) + \left(m_k + N_{cbps} - floor\left(\frac{12.m_k}{N_{cbps}}\right)\right)_{mod(s)} \qquad k = 0,1,\dots,N_{cbps}-1 \quad \textbf{(2.16)}$$

El primer bit a la salida del interleaver debe mapear el bit más significativo de la constelación.

2.2. Decodificador

La decodificación del canal se encarga de recuperar la información transmitida dado un bloque de símbolos que contiene información redundante y que han sido alterados por las imperfecciones del canal. Para lograr esto, se tiene que tener conocimiento de las características con que se realizo la codificación y dependiendo del código usado, podrá detectar y corregir una cierta cantidad de errores.

El estándar IEEE 802.16-2004 no específica cómo debe ser implementado el decodificador, por lo que este debe ser implementado de manera inversa. Sus etapas son las siguientes: Deinterleaver, Decodificador de Viterbi (usado para decodificar los códigos convolucionales), el Decodificador de Reed-Solomon y el Derandomizer.

2.2.1. Deinterleaving

El de-interleaver se encarga de reordenar los bits de cada bloque da datos y así recuperar el orden que tenían antes de pasar por el interleaver. Su funcionamiento es exacto al realizado en la etapa de codificación, pero las permutaciones que lo definen son diferentes de tal manera que representen la operación inversa de ordenamiento.

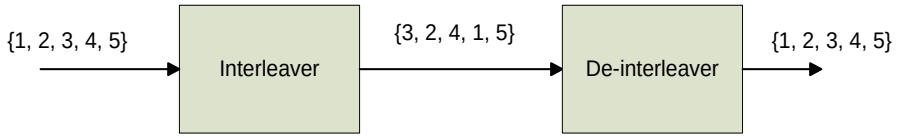

Figura 2.18 Operación del deinterleaver

En el caso que se realice demodulación por *soft decision*, la entrada y salida del deinterleaver será un símbolo de n bits, donde n es el número usado en la cuantización hecha a continuación de la demodulación. De esta manera no se recuperan los mismos bits que estaban antes del codificador, sino la posición en que se encontraban, lo cual es la función del proceso del interleaving, dejando la función de recuperar los datos al FEC.

Deinterleaver usado en el estándar 802.16-2004

A diferencia de las otras etapas del decodificador donde no está especificado en el estándar el método a usar de decodificación, para el deinterleaving es

necesario conocer las ecuaciones que representen la permutación de los datos a fin de recuperar la posición inicial de los mismos.

Definiendo el tamaño del bloque recibido como N_{cbps} bits, j el índice de los bits recibidos antes de la primera permutación, m_j el índice de los bits después de la primera y antes de la segunda permutación y k_j el índice de los bits después de la segunda permutación y antes de entregar el bloque de bits recibidos a la etapa del FEC.

$$m_j = s.floor\left(\frac{j}{s}\right) + \left(j + floor\left(\frac{12.j}{N_{cbps}}\right)\right)_{mod(s)} \qquad j = 0,1,\dots,N_{cbps}-1 \qquad \textbf{(2.17)}$$

$$k_j = 12.m_j - \left(N_{cbps}-1\right).floor\left(\frac{12.m_j}{N_{cbps}}\right) \qquad j = 0,1,\dots,N_{cbps}-1 \qquad \textbf{(2.18)}$$

La primera permutación en el deinterleaver es el inverso de la segunda ecuación del interleaver y así sucesivamente.

2.2.2. Decodificador de Viterbi

Hay varias formas de realizar la decodificación de códigos convolucionales, los cuales están agrupados en 2 grupos:

1. Decodificación Secuencial
    - Algoritmo de Fano.
2. Decodificación de Máxima Probabilidad
    - Decodificación de Viterbi.

El algoritmo de Viterbi es el más usado y más conocido por lo que el proyecto se enfoca en su uso. Su ventaja radica en que podemos reducir las opciones o caminos sistemáticamente en cada instante de tiempo. Para hacer esto es necesario asumir lo siguiente:

1. Los errores ocurren infrecuentemente. Por la tanto la probabilidad de error es pequeña.
2. La probabilidad de dos errores en fila es mucho menor que un simple error. Lo que significa que los errores están distribuidos aleatoriamente.

El decodificador de Viterbi examina la secuencia entera recibida, calcula una métrica por cada camino y hace una decisión basada en esta métrica. Todos los caminos son seguidos hasta que dos caminos converjan en un nodo. Entonces el camino con mejor métrica se mantiene y el de la métrica menor es descartado. Los caminos seleccionados son llamados sobrevivientes.

La métrica más usada es la distancia de Hamming, discutida anteriormente, que es usada cuando se realiza una demodulación por *Hard Decision*. Sin embargo también se usa la distancia euclidiana cuando se realiza una demodulación por *Soft Decision.* Por lo tanto la métrica usada dependerá del bloque decisor usado en el receptor del sistema de comunicaciones.

El canal gaussiano dispone de un alfabeto discreto de entrada y un alfabeto continuo de salida que estaría de $-\infty$ a $+\infty$. De esta manera, con una

secuencia binaria a la entrada del canal, la salida tendría infinitos valores. En la figura 2.19 se muestra la probabilidad de la señal recibida cuando se envía un 0 f(y/0) y la probabilidad si se envió un 1 f(y/1) si se tiene un canal AWGN.

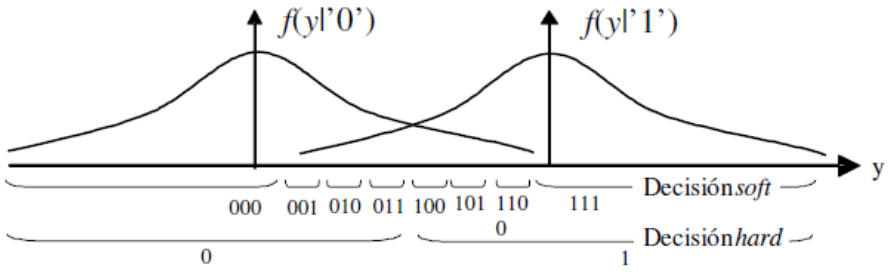

Figura 2.19 Métodos de detección soft y hard

Usando Hard Decision la salida del demodulador es cuantizada en 2 niveles, cero o uno, y es la entrada del codificador. En el caso que la salida del demodulador sea cuantizada a mas de 2 niveles estamos usando Soft Decision, de esta forma estamos proporcionando al decodificador mayor información, como en el caso del ejemplo de la figura donde tener una salida de “111” implicaría haber recibido un 1 seguro y si tenemos “100” tenemos que es el valor que recibió el canal es poco probable de ser 1, pero aun así más probable de no ser un 0. Los resultados de cuantizar en 8 niveles implican una reducción en 2 dB de la relación de señal a ruido comparado cuando se usa Hard Decision. El precio que se paga es un decodificador más complejo ya que tendrá que trabajar con una cantidad adicional de datos dependiendo del número de soft bits, es decir un aumento en memoria y de retardos.

Para explicar el funcionamiento del algoritmo usado por el decodificador de Viterbi se expresa en el siguiente ejemplo, donde suponemos la secuencia recibida resultante del codificador convolucional 01 11 01 11 01 01 11 donde el bit subrayado es un error resultado de pasar por un canal AWGN y un demodulador de Hard Decision.

a. Se siguen todos los caminos posibles, comenzando del estado cero, y se va colocando la métrica, que indica el número de concordancias que tiene la señal recibida en cada tramo. A diferencia de la decodificación secuencial, el algoritmo de Viterbi revisa todos los caminos posibles.
b. Una vez que los caminos convergen, se elimina el que posea menor métrica, con el fin de optimizar el algoritmo. Los caminos resultantes son llamados los sobrevivientes.
c. Se siguen los pasos a y b hasta terminar la secuencia recibida.
d. Una vez terminada la secuencia de recibida, encontramos en la figura 2.20 que existen 15 caminos posibles, del cual se toma el que posee la mayor métrica, en este caso la métrica igual a 13.
e. Para obtener la señal decodificada, se sigue el camino escogido. Es decir pasa por los estados 000, 100, 010, 101, 110, 011, 001, y 000; y recordando que las líneas continuas representan un 0 y las discontinuas un 1 entonces la secuencia decodificada es 1011000 la cual fue la secuencia de entrada usada en el ejemplo del codificador.

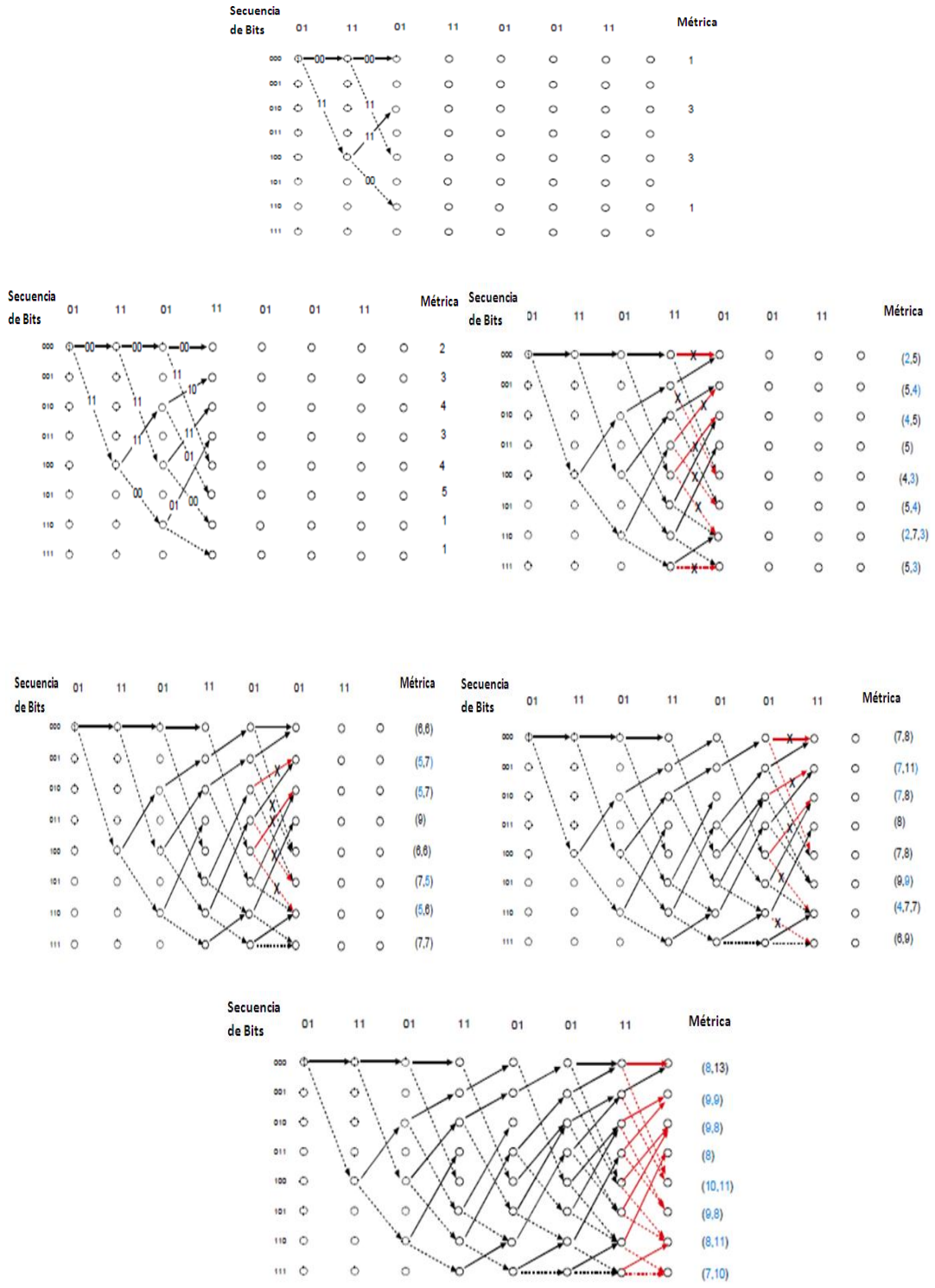

Figura 2.20 Algoritmo de Viterbi

Depuncturing

Dado que los datos han pasado por un proceso de puncturing antes de la transmisión, entonces una etapa de depuncturing debe ser realizada antes del decodificador de Viterbi. Para esto se necesita usar el mismo vector de perforación usado en el codificador, y en el caso donde este el '0' se deberá insertar un símbolo nulo, que va a depender del método de demodulación usado. Si es Hard Decision, se insertara un '0' como símbolo nulo, y si la demodulación se realiza por Soft Decision, el símbolo a insertar debe ser un valor que este entre los niveles que exista más probabilidad de haber recibido un '0'. Además para que el decodificador de Viterbi tenga conocimiento de la existencia del símbolo nulo, este será indicado por una señal llamada *erasure.*

Este proceso se lo indica en el ejemplo de las figuras 2.21 y 2.22 usando una tasa de codificación 3/4.

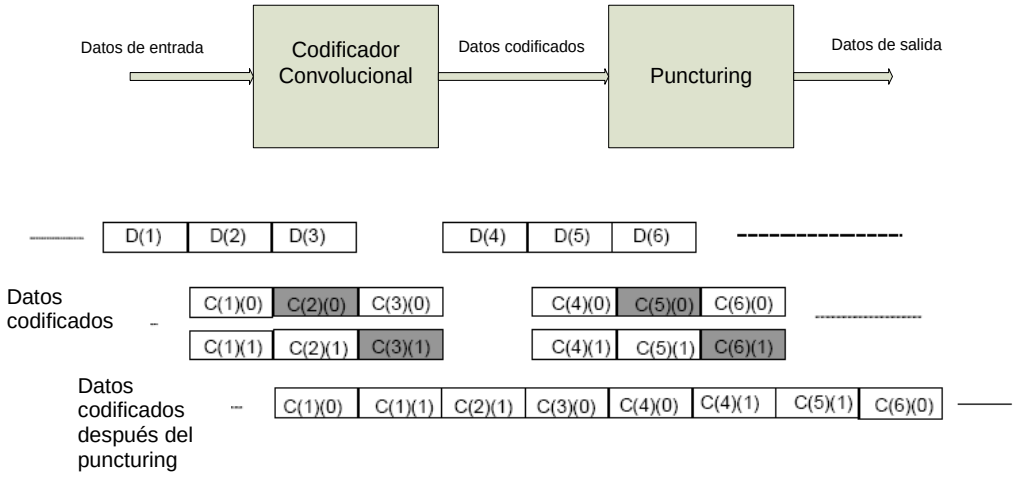

Figura 2.21 Puncturing con una tasa de codificación ¾

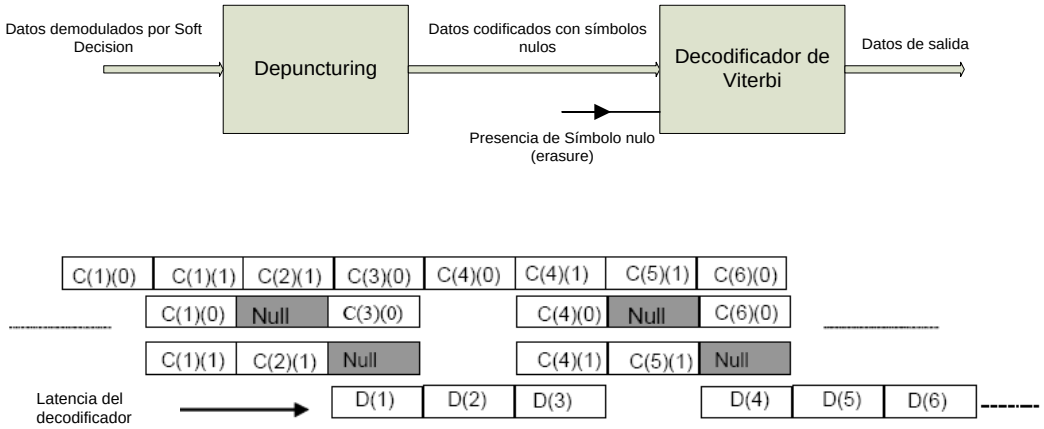

Figura 2.22 Depuncturing con una tasa de codificación ¾

2.2.3. Decodificador de Reed-Solomon

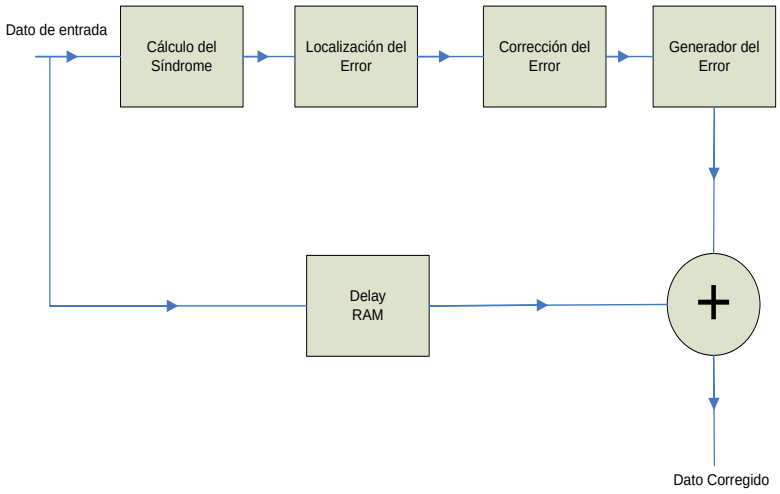

Figura 2.23 Pasos para decodificar códigos Reed-Solomon

Para el diseño del decodificador de Reed-Solomon, se sigue un proceso más complejo que el explicado en el codificador, pues tendrá que analizar si el símbolo recibido contiene un error, localizarlo y así poder cambiarlo por el valor que fue enviado al inicio, todo esto recordando que el decodificador puede corregir un máximo de t errores, valor que debe conocer para así trabajar en sincronía con la etapa codificadora. En la figura 2.23 se muestra el diagrama de bloques del decodificador de Reed-Solomon.

En el decodificador vamos a tener una variable adicional llamado e(x) que es el ruido que entra al sistema y que va a depender de las características del canal. Se le puede representar por la siguiente ecuación:

$$e(x) = \sum_{i=0}^{n-1} u_i x^i \qquad \textbf{(2.19)}$$

Donde n es el tamaño del bloque de salida del codificador. Es decir, tomando como ejemplo un codificador RS(7,3) van a existir 6 símbolos de error, donde en condiciones ideales u_i tiene que ser igual a cero, y el número de símbolos diferentes de cero tienen no pueden pasar la capacidad de corrección del código Reed-Solomon si es que se quiere corregir todos los errores encontrados.

Entonces teniendo estas variables, tenemos que la secuencia recibida por el decodificador es de la forma dada por la ecuación:

$$r(x) = U(x) + e(x) \qquad \textbf{(2.20)}$$

Los pasos seguidos por el decodificador dado en el diagrama de bloques son los siguientes:

Cálculo del Síndrome

El síndrome S es el resultado del chequeo de los símbolos de paridad sobre r(x) para determinar si el símbolo recibido es una palabra de código valida y está compuesto por n-k símbolos $S_i\{i = 0,1, \dots n-k\}$. Cualquier valor diferente de cero en S indica la presencia de error. Recordando que $U(x) = m(x)g(x)$, al ser U(x) múltiplo de g(x), se tiene entonces que las raíces de g(x) son también las raíces de U(x).

Dado que $r(x) = U(x) + e(x)$, entonces evaluando R(x) con las raíces obtenidas si da un valor diferente de cero, entonces hay presencia de error en el símbolo. Entonces el Síndrome S viene dado por:

$$S_i = r(x)|_{x=\alpha^i} = r(\propto^i) \qquad i = 1, \dots, n-k \qquad \textbf{(2.21)}$$

Localización del error

Para esta parte del decodificador se toma ventaja de la estructura matricial del código, ya que es necesario resolver de forma simultánea ecuaciones con t incógnitas.

Suponiendo que hay v errores en la secuencia recibida con locación $x^{j_1}, x^{j_2}, \dots, x^{j_v}$, entonces e(x) puede ser escrita por la siguiente ecuación:

$$e(x) = e_{j_1}x^{j_1} + e_{j_2}x^{j_2} + \cdots + e_{j_v}x^{j_v} \qquad \textbf{(2.22)}$$

Donde el índice v se refiere al número de error y j a la locación del error. Para corregir la palabra con error, cada valor de error e_{j_l} y su locación x^{j_l}, donde l=1,2,…,v deben ser encontrados.

Definimos la locación del error como $\beta_l = \alpha^{j_l}$ **(2.23)**, y a continuación obtenemos los n-k síndromes sustituyendo α^i en el polinomio recibido para i=1,2,…2t.

$$S_1 = r(\alpha) = e_{j_1}\beta_1 + e_{j_2}\beta_2 + \cdots + e_{j_v}\beta_v$$

$$S_2 = r(\alpha^2) = e_{j_1}{\beta_1}^2 + e_{j_2}{\beta_2}^2 + \cdots + e_{j_v}{\beta_v}^2 \ , \quad \ldots.$$

$$S_{2t} = r(\alpha^2 t) = e_{j_1}{\beta_1}^{2t} + e_{j_2}{\beta_2}^2 t + \cdots + e_{j_v}{\beta_v}^{2t} \qquad \textbf{(2.24)}$$

Entonces al resolver este sistema de 2t ecuaciones no lineales y 2t incógnitas podremos saber la ubicación de cada error de la secuencia recibida. Para esto el decodificador de Reed-Solomon utiliza un algoritmo propio que permita resolver este sistema.

Corrección y Generación del error

Una vez que se tiene la localización del error, hay que cambiarlo por el valor correcto en el símbolo. Para esto se necesita armar el polinomio localizador de error en base a la locación de los errores encontradas con anterioridad

utilizando el algoritmo de Berlekamp- Massey o el Algoritmo de Euclides (17). Utilizando el algoritmo de búsqueda de Chien (18) encontramos las raíces del polinomio anterior y con el algoritmo de Forney (19) se encuentran los valores del símbolo erróneo.

Finalmente se realizara la adición de módulo-2 (operación XOR), entre el dato recibido por el decodificador y que se mantiene guardado en una memoria RAM, y el error generado con lo que se obtiene el código del símbolo recuperado.

DeShortening y Depuncturing

Para esto se sigue el proceso inverso de la Figura 2.8 para el shortening. Es decir agrego una cantidad de k-k' ceros al inicio del bloque para realizar el deshortening. Para el depuncturing se necesita insertar una cantidad de t-t' ceros al final del bloque. Con esto se tiene el tamaño del bloque original n que será la entrada del decodificador Reed-Solomon. Para diferenciar los símbolos de paridad nulos que han sido insertados en el proceso de de-puncturing, se ingresa una señal al decodificador que indique la nulidad de esos símbolos.

2.2.4. DeRandomizer

Para la parte de decodificación se tiene que usar la misma etapa usada en la codificación, con el mismo polinomio de retroalimentación y vector de inicialización usado como semilla.

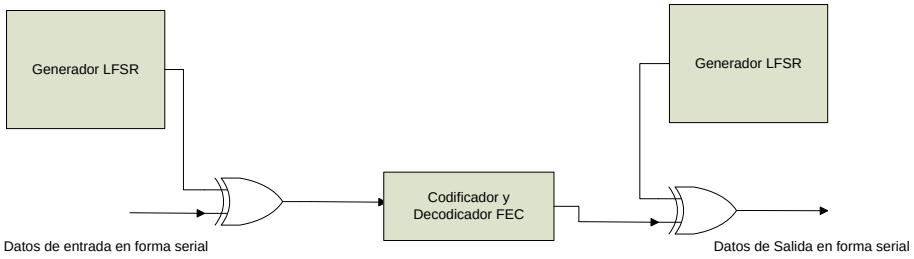

Figura 2.24 Uso del DeRandomizer

El de-randomizer recibe la señal recuperada por el decodificador de Reed-Solomon en forma serial para poder trabajar a nivel de bit de igual manera que la parte de la codificación.

CAPÍTULO III

3. DISEÑO E IMPLEMENTACIÓN DEL ESQUEMA DE CODIFICACIÓN/DECODIFICACIÓN

El diseño descrito en este capítulo se centra en la implementación del codificador y decodificador del canal para el estándar IEEE 802.16-2004 WirelessMAN-OFDM, y tal como se describieron sus componentes en el capítulo 2. El entorno de trabajo es Simulink y Matlab y la implementación se hará únicamente usando los bloques del System Generator.

3.1. Diagrama de Bloques del Codificador

El diseño del codificador del canal contiene 4 importantes bloques:

- Randomizer.
- Codificador de Reed-Solomon.

- Codificador Convolucional.
- Interleaver.

Estos bloques se tienen que ajustar a los 6 diferentes perfiles de codificación y modulación especificados en la Tabla 1.3. El randomizer tiene una implementación única para los 6 perfiles por lo que se explica su diseño de manera independiente.

3.1.1. Descripción de los bloques del Codificador.

Randomizer

Para la implementación del randomizer se utilizo el diseño especificado en la figura 3.1.

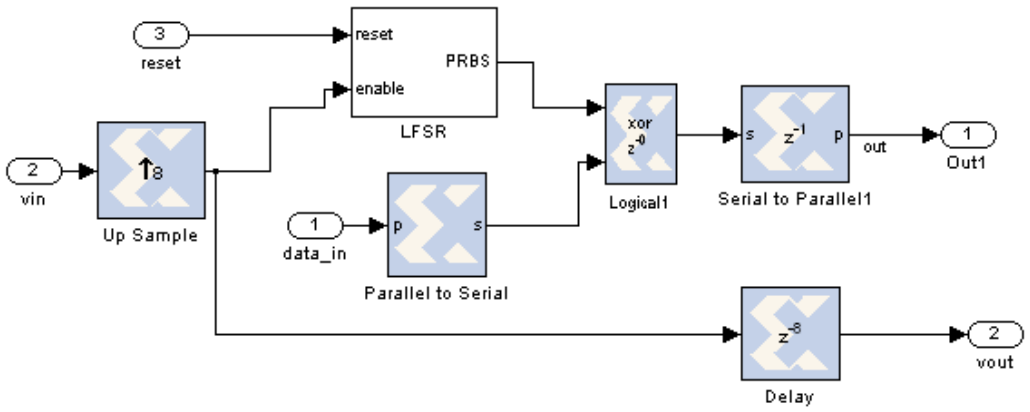

Figura 3.1 Diseño del Randomizer

Para el randomizer se tienen 3 entradas: data_in, vin y reset. La entrada data_in se refiere a los datos que vienen de la capa MAC en forma de byte

los cuales pasan por un convertidor de paralelo a serial convirtiendo el bit más significativo primero. Luego haciendo la operación lógica XOR con la secuencia PRBS se obtienen los datos aleatorizados. Dado que la siguiente parte del diseño trabaja a nivel de byte se agrupan los bits por medio del convertidor de serie a paralelo. Para poder sincronizar el siguiente bloque del diseño del codificador se utiliza la señal vout, donde el bloque Down Sample se utiliza para reducir el tiempo de muestreo que en la señal de salida cambio por la presencia del convertidor serie a paralelo.

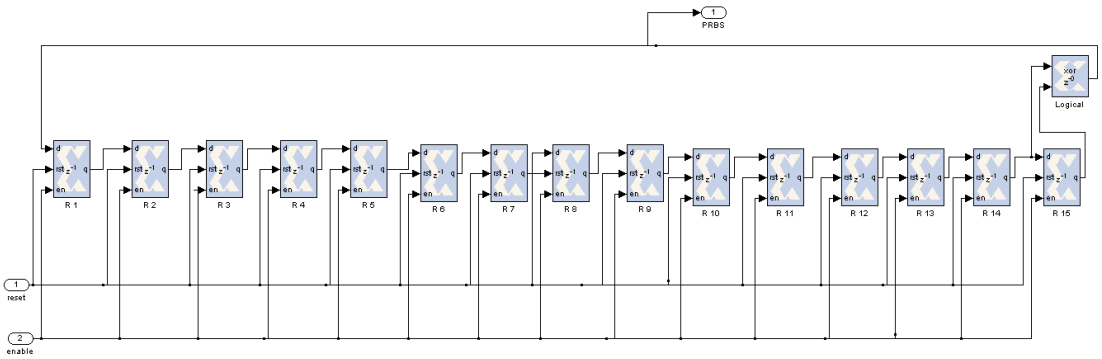

Figura 3.2 Bloque LFSR

La secuencia pseudoaleatoria es originada por el bloque LFSR el cual recibe una señal de habilitación vin y una señal de reset, las cuales deben tener un período de muestreo para manejar el bloque a nivel de bit. Como se observa en la figura 3.2, su implementación se realizó usando un registro que tiene la opción de poner un valor inicial a diferencia del bloque Delay. Los valores de inicialización dependerán de los valores de semilla especificados en las figuras 2.3 y 2.4. Siguiendo lo especificado en el estándar se coloco una

puerta de retroalimentación XOR, donde la salida retroalimentada será la salida del bloque.

Para preparar el bloque antes de pasar por el codificador de Reed-Solomon se utiliza el bloque Zero-Tailing. Su diseño variará dependiendo el perfil de codificación escogido. Como se muestra en la figura 3.3, se utiliza un comparador que seleccionara en el MUX el byte cero que será insertado al final del bloque.

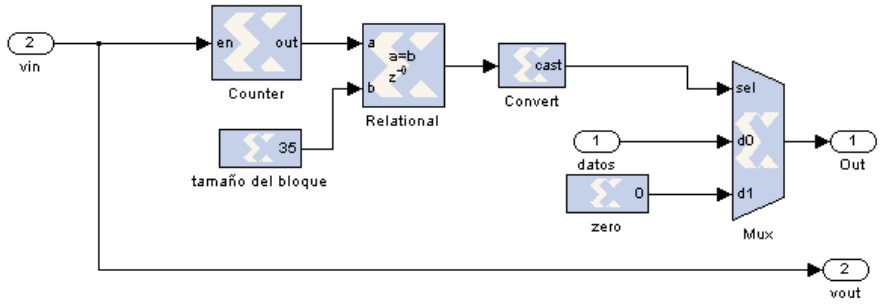

Figura 3.3 Inserción del byte cero al final del bloque

En la figura 3.3 se utiliza un bloque que representa el tamaño del bloque, en este caso 35, correspondiente al tamaño del bloque sin codificar menos uno para el perfil de codificación QPSK 3/4. Para poder utilizar el resto de perfiles se tiene que cambiar el valor límite en que funciona el contador y el tamaño del bloque usado en el comparador.

Reed-Solomon

El codificador de Reed-Solomon se compone de 3 etapas para poder cumplir las especificaciones del estándar.

- El codificador Reed-Solomon propiamente dicho.
- La etapa de puncturing.
- Reordenamiento de los datos.

La etapa de shortening no es necesaria ya que el core que implementa el codificador de Reed-Solomon cumple esta función con una señal externa que le indica el tamaño del bloque variable (señal n_in en la figura 3.4). El diagrama de bloques usado se muestra a continuación.

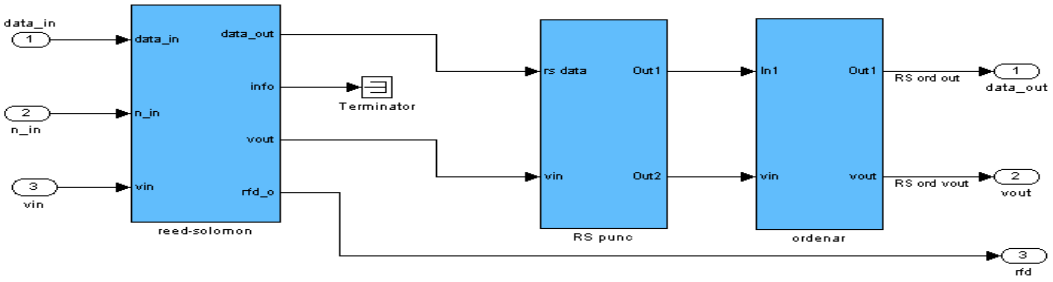

Figura 3.4 Diagrama de bloques del Codificador Reed-Solomon

La salida *data_out* tendrá el bloque con el tamaño n' especificado en el perfil de codificación, con los bytes de paridad al inicio del bloque y la señal vout será alta cuando se tenga el primer dato valido a la salida. La señal rfd se utiliza para indicar que el codificador está listo para aceptar nuevos datos.

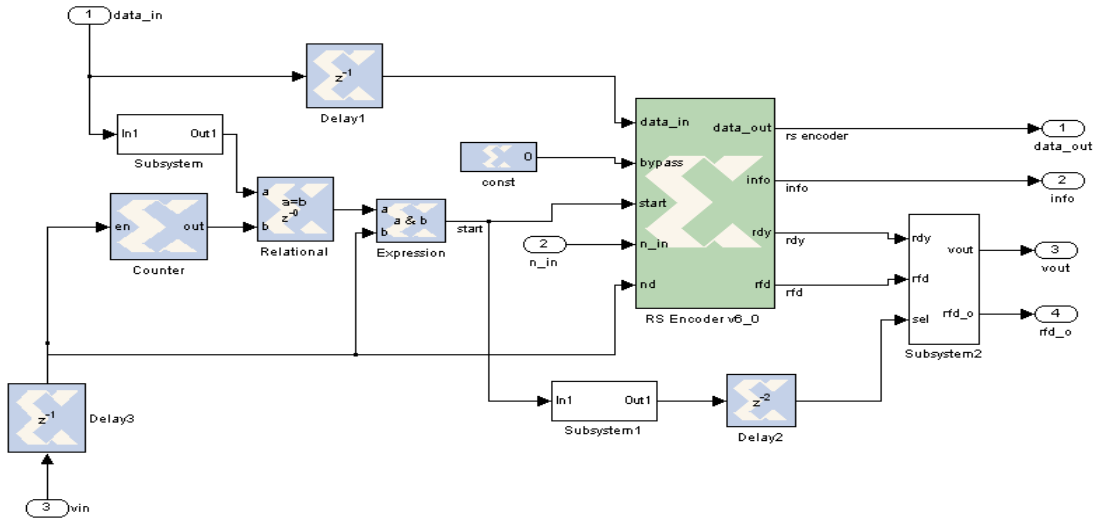

Figura 3.5 Etapa de codificación

La figura 3.5 muestra como se interconecta el core del codificador de Reed-Solomon. La señal *nd* sirve para habilitar el codificador y la señal *bypass* para hacer que la señal de salida sea siempre la señal de entrada. La señal *n_in* debe ser igual al tamaño del bloque de entrada más el número de símbolos de paridad utilizados en el código madre, que es una cantidad fija igual a 16; de esta forma se realiza el proceso de shortening especificado en el estándar. La señal *start* indicara al codificador cuando empezar a recibir nuevos símbolos y variara dependiendo el valor límite especificado en el contador. Los subsistemas de la figura 3.5 solo sirven para muestrear la señal de entrada y no tienen relevancia en el diseño. Los parámetros para el codificador de Reed- Solomon se describen en la figura 3.6.

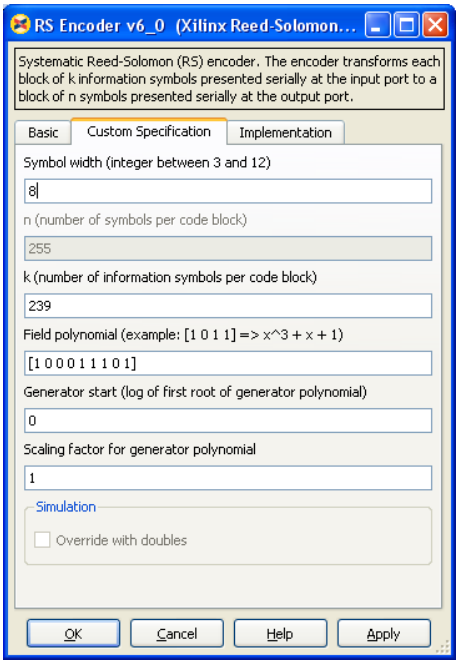

Figura 3.6 Parámetros del Codificador de Reed-Solomon

El número de símbolos de información es 239 como esta especificado en el estándar, lo que en conjunto con la señal n_in externa realiza el shortening. Los parámetros del polinomio primitivo (Field Polinomial) y el polinomio generador del campo (Generator Polinomial), son descritos tal como se especifico en el capitulo 2.1.2.

La señal *rfd* es baja cuando se está produciendo los símbolos de paridad, indicando que no está lista para aceptar nuevos datos de entrada. Los símbolos de paridad son especificados mediante la señal *info*, la cual es igual a cero en el instante de tiempo en que aparecen.

Hasta este punto del diseño todavía no se tiene el tamaño de bloque deseado. Este proceso es realizado por la etapa de puncturing que se

encarga de eliminar una cantidad especificada de símbolos de paridad. Su implementación se ilustra en la figura 3.7.

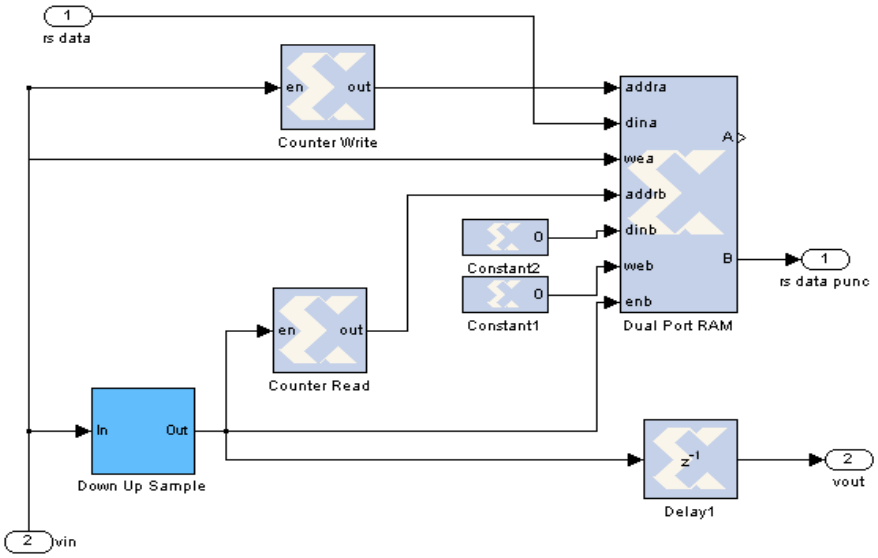

Figura 3.7 Puncturing usado en el Codificador Reed Solomon

Este proceso se realiza con una memoria RAM de doble puerto, la cual permite acceder al mismo contenido de memoria por 2 puertos, como su nombre lo indica. Esto se realiza manejando cada puerto a una frecuencia de reloj distinta. Dado que System Generator no presenta la opción de modificar la señal de reloj, lo que se hace es cambiar la tasa de muestreo al que entran los datos (Sample Rate). Para la escritura se usa la misma tasa de muestreo con el que se realizó la codificación y para la lectura será una tasa de muestreo variable dependiendo el perfil de codificación usado, tal como se muestra en la tabla siguiente:

Tabla 3.1 Tasas de muestro usadas en el Puncturing para el codificador de Reed-Solomon

Perfil de Codificación	Tasa de muestreo
1	5/4
2	13/10
3	No aplica
4	11/10
5	28/27
6	31/30

Los valores de la tasa de muestro se las obtiene de la razón entre el número de símbolos en el bloque a la salida del codificador (expresado como la señal n_in) dividido por el tamaño del bloque a la salida del puncturing.

Para cambiar la tasa de muestreo se utiliza un bloque Down Sample con el valor expresado en el numerador de la razón, seguido de un bloque Up Sample, con el valor expresado en el denominador. Tanto el contador de lectura como escritura deben tener valores límite diferentes, y además, el contador de lectura debe reflejar el cambio de la tasa de muestreo en la opción del Periodo de Muestreo especificado en el bloque. Se toma la salida del puerto B, la cual tendrá la tasa de muestreo con que se realizó la lectura, por lo que debe ser tomado en cuenta para utilizar otras etapas del codificador del canal.

Por último, para enviar primero los bytes de paridad, se aprovecha el bloque implementado para realizar el proceso de interleaver, descrito más adelante. Su implementación se muestra en la figura 3.8.

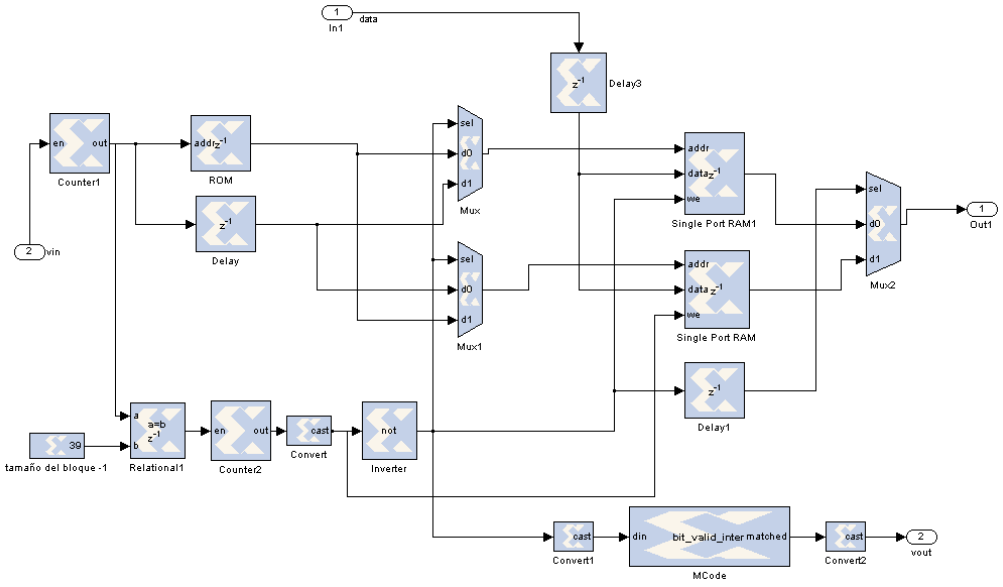

Figura 3.8 Reordenamiento de los símbolos de paridad

La idea básica es que cuando se realice la lectura de los datos, se utilice la dirección de memoria dada por la memoria ROM, que se inicializa con los vectores dados por la tabla 3.2. Estos valores son generados por Maltab previo a la implementación.

Tabla 3.2 Vectores de inicialización para la memoria ROM en el Codificador de Reed-Solomon

Perfil de Codificación	Vector de inicialización
1	[25:32 1:24]
2	[37:40 1:36]
3	[49:64 1:48]
4	[73:80 1:72]
5	[97:108 1:107]
6	[109:120 1:108]

Codificador Convolucional

El codificador convolucional es implementado usando un codificador nativo con tasa de codificación 1/2, seguido por una etapa de puncturing, el cual será variable dependiendo del perfil de codificación usado, tal como se describió en el capitulo 2.1.3., y cuyo diseño se presenta con el diagrama de bloques de la figura 3.9.

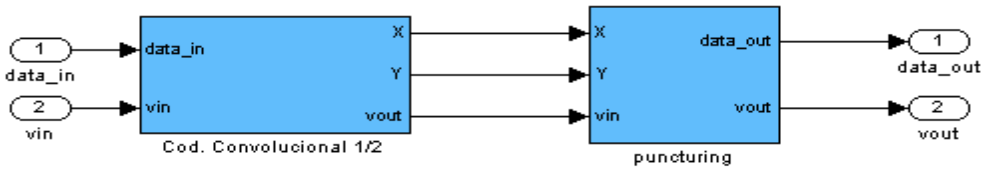

Figura 3.9 Diagrama de Bloques del Codificador Convolucional

El diseño del codificador convolucional nativo se muestra en la figura 3.10, donde el elemento principal es el bloque que realiza la codificación en sí, y cuyos parámetros se especifican en la figura 3.11. Se utiliza los bloques de retardo para mantener la sincronización con la etapa anterior del codificador del canal, el cual envía las señales en forma de byte, por lo que se necesita convertir la señal de paralelo a serial para que pueda funcionar en esta etapa. Dado que este bloque cambio la tasa de muestreo de los datos, se necesita hacer lo mismo con la vin que indica cuando comenzar a realizar la codificación de los datos.

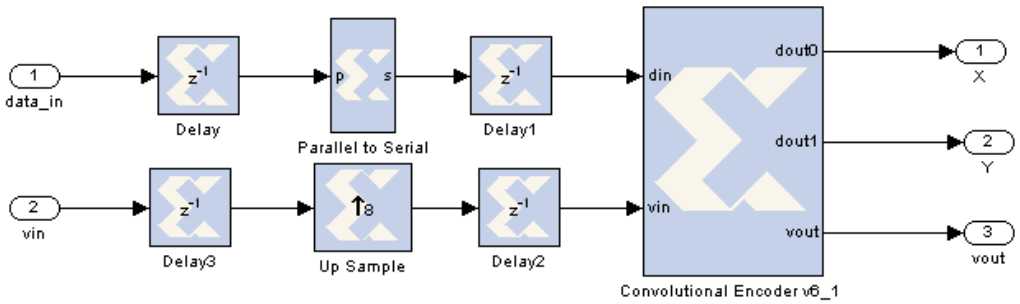

Figura 3.10 Implementación del codificador Convolucional nativo

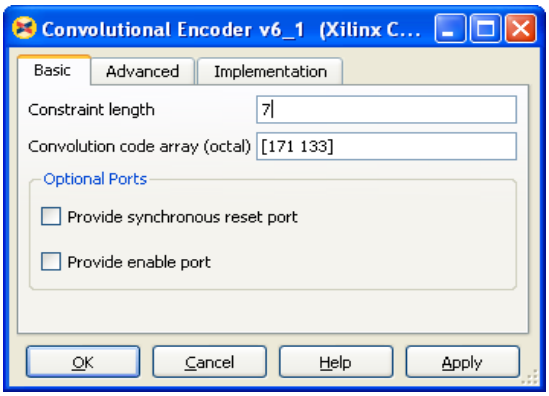

Figura 3.11 Parámetros del Codificador Convolucional nativo

Para el diseño del puncturing se utiliza el diseño de la figura 3.12. El primer paso es realizar la serialización de los datos enviados de la etapa anterior. Las señales X y Y son serializadas por medio del bloques Concat, y el convertidor de serial a paralelo se encarga de preparar los datos que van a ser procesados por el bloque de puncturing. El convertidor serial a paralelo realizará la conversión con un número de bits que dependerá de la longitud del vector de puncturing. Estos dependen del perfil de codificación usado y expresado en la tabla 2.5. Dado que el bloque Puncture cambia la tasa de

muestreo de los datos, se realiza lo mismo para obtener la señal vout, por medio de un bloque Down Sample seguido de un Up Sample, cuyos valores serán iguales al perfil de codificación usado.

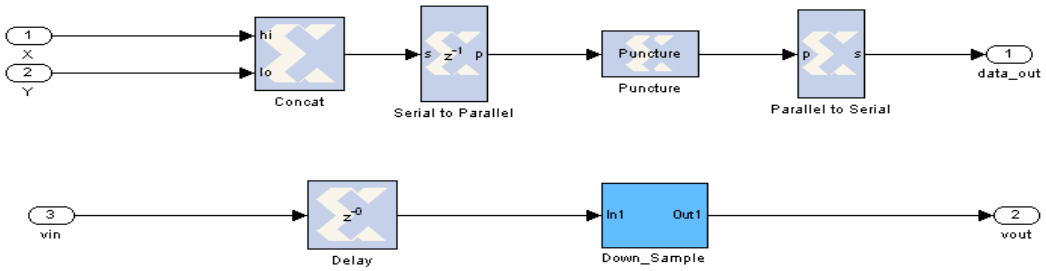

Figura 3.12 Implementación del Puncturing usado en el Codificador Convolucional

En base a la tabla 3.3, se describen los vectores de puncturing implementados en Matlab usados en el bloque Puncture.

Tabla 3.3 Vectores de Puncturing usados en la implementación

Tasa de codificación	Vector de Puncturing en forma serial
2/3	[1 1 0 1]
3/4	[1 1 0 1 1 0]
5/6	[1 1 0 1 1 0 0 1 1 0]

Interleaver

El diseño del interleaver se muestra en la figura 3.13.

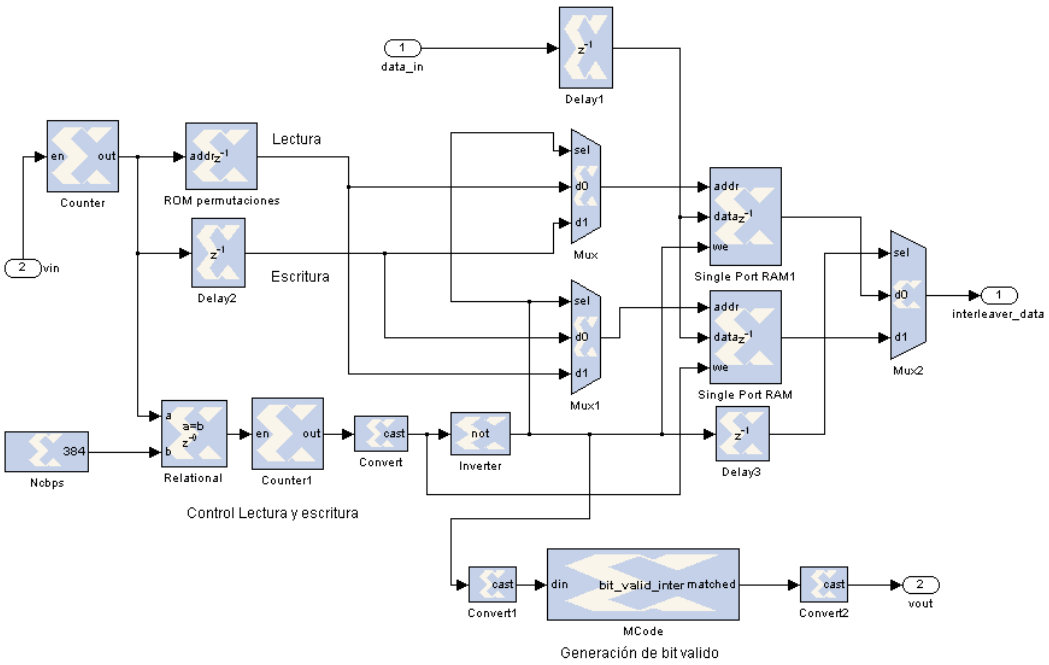

Figura 3.13 Diseño del Interleaver

Tal como se describió en la sección 2.1.3, el interleaver para el estándar IEEE 802.16-2004 se realiza por medio de 2 permutaciones. El resultado de realizar las 2 permutaciones se describe en un vector cuya función será inicializar la memoria ROM presente en la figura 3.13. El funcionamiento de este diseño del interleaver es grabar los datos de un bloque de datos en una memoria RAM, mientras se está realizando la lectura por otra memoria. Para lograr esto, se tienen que tomar 2 señales, la salida de la memoria ROM de las permutaciones será seleccionada para realizar la lectura de los datos, mientras que para la escritura se toman los datos del contador que cuenta hasta llegar al valor especificado como N_{cbps} que es la cantidad en bits del bloque codificado. El control de la lectura y escritura se realiza cambiando el

modo de escritura a lectura en la primera RAM, y de escritura a lectura en la segunda, lo cual se efectuará cuando el contador llegue al valor N_{cbps}, y el contador comenzara una nueva cuenta con lo que se procesara el siguiente bloque de datos.

Las permutaciones se generan con las siguientes ecuaciones en Matlab:

```
k = 0 : Ncbps - 1;
mk = (Ncbps/12) * mod(k,12) + floor(k/12);
s = ceil(Ncpc/2);
jk = s * floor(mk/s) + mod(s, mk + Ncbps - floor(12 * mk/Ncbps));
[s1,int_idx]=sort(jk);
```

(3.1)

El vector generado *int_idx* será el vector usado en la memoria ROM de permutaciones, y representa el índice de los valores calculados después de la segunda permutación.

Para generar la señal vout, se utiliza el bloque Mcode, el cual implementa una maquina de estados finitos escrita en código de Matlab, el cual generará la señal una vez que se termine se grabar el primer bloque de datos.

El periodo de las muestras que debe ser especificado en los bloques se basa en la siguiente ecuación:

$$T_s = \frac{(T_s\ RS)(Tasa\ de\ Codificación\ CC)}{8} \qquad \textbf{(3.2)}$$

Donde $T_s\,RS$, es el valor especificado en la tabla 3.1 y $Tasa\ de\ Codificación\ CC$, en la Tabla 1.3. Se lo divide para 8, debido a que los datos en esta etapa del diseño están a nivel de bit.

3.1.2. Implementación en FPGA del Codificador

El codificador del canal es implementado usando la tarjeta para FPGA Spartan 3E Starter Kit. La FPGA usada es la Spartan 3E xc3s500e-4fg320. El análisis será centrado en el área que ocupa en la FPGA cada una de las etapas mostradas en el diseño. Así mismo, la implementación se ha realizado de manera independiente para cada perfil de codificación.

Tabla 3.4 Recursos de la FPGA para el Codificador usando BPSK 1/2

Etapa	Slices	Slice FF	LUTs	IOBs	BRAMs
Randomizer	34	61	34	20	0
Reed-Solomon	0	0	0	0	0
Codificador Convolucional	25	42	22	12	0
Interleaver	17	23	34	5	3

Tabla 3.5 Recursos de la FPGA para el Codificador usando QPSK 1/2

Etapa	Slices	Slice FF	LUTs	IOBs	BRAMs
Randomizer	34	61	34	20	0
Reed-Solomon	189	277	325	19	4
Codificador Convolucional	51	87	54	12	0
Interleaver	23	34	46	5	3

Tabla 3.6 Recursos de la FPGA para el Codificador usando QPSK 3/4

Etapa	Slices	Slice FF	LUTs	IOBs	BRAMs
Randomizer	34	61	34	20	0
Reed-Solomon	214	285	285	20	4
Codificador Convolucional	66	115	74	12	0
Interleaver	26	36	51	5	3

Tabla 3.7 Recursos de la FPGA para el Codificador usando 16-QAM 1/2

Etapa	Slices	Slice FF	LUTs	IOBs	BRAMs
Randomizer	34	61	34	20	0
Reed-Solomon	159	235	279	20	3
Codificador Convolucional	45	78	36	12	0
Interleaver	21	32	41	5	3

Tabla 3.8 Recursos de la FPGA para el Codificador usando 16-QAM 3/4

Etapa	Slices	Slice FF	LUTs	IOBs	BRAMs
Randomizer	34	61	34	20	0
Reed-Solomon	204	299	352	20	4
Codificador Convolucional	66	114	71	12	0
Interleaver	29	44	57	19	3

Tabla 3.9 Recursos de la FPGA para el Codificador usando 64-QAM 2/3

Etapa	Slices	Slice FF	LUTs	IOBs	BRAMs
Randomizer	34	61	34	20	0
Reed-Solomon	203	294	349	20	4
Codificador Convolucional	62	105	71	12	0
Interleaver	28	40	55	5	4

Tabla 3.10 Recursos de la FPGA para el Codificador usando 64-QAM 3/4

Etapa	Slices	Slice FF	LUTs	IOBs	BRAMs
Randomizer	34	61	34	20	0
Reed-Solomon	205	297	352	20	4
Codificador Convolucional	67	119	75	12	0
Interleaver	28	41	56	5	4

3.2. Diagrama de Bloques del Decodificador

El diseño del decodificador de canal se realiza de forma inversa al codificador, por lo que de igual manera cuenta con 4 bloques:

- Deinterleaver.
- Decodificador de Viterbi
- Decodificador de Reed-Solomon.
- Derandomizer.

Así como se implemento el codificador, cada bloque a excepción del randomizer dependerá del perfil de codificación usado. Además para el diseño del randomizer se usa el mismo diseño usado para la codificación.

Para el diseño del decodificador, se asume que se tiene una señal demodulada por *soft decision*, usando 3 soft bits. Esta demodulación tiene que ser realizada usando un algoritmo LLR (Log-Likelihood Ratio), implementado en Matlab, usando la siguiente ecuación:

$$L(b) = log\left(\frac{P_r(b=0 \mid r=(x,y))}{P_r(b=1 \mid r=(x,y))}\right) \qquad \textbf{(3.3)}$$

De la ecuación 3.3, asumiendo igual probabilidad para todos los símbolos en un canal AWGN, se tiene:

$$L(b) = log\left(\frac{\sum_{s \in S_0} e^{-\frac{1}{\sigma^2}\left((x-s_x)^2+\left(y-s_y\right)^2\right)}}{\sum_{s \in S_1} e^{-\frac{1}{\sigma^2}\left((x-s_x)^2+\left(y-s_y\right)^2\right)}}\right) \qquad \textbf{(3.4)}$$

Las variables de la ecuación 3.4 son detalladas en la tabla 3.11.

Tabla 3.11 Variables para el algoritmo LLR en Matlab

Variable	Significado
R	Señal recibida con coordenadas (x,y)
B	Bit transmitido
S_0	Punto de constelación con bit 0
S_1	Punto de constelación con bit 1
s_x	Coordenada en fase
s_y	Coordenada en cuadratura
σ^2	Varianza del ruido de la señal en banda base

Un método alternativo (20), es realizar una aproximación, donde solo se toma en cuenta un solo punto de la constelación, el cual será el más cercano a la señal recibida.

$$L(b) = -\frac{1}{\sigma^2}\left(min_{s \in S_0}\left((x-s_x)^2+\left(y-s_y\right)^2\right) - min_{s \in S_1}\left((x-s_x)^2+\left(y-s_y\right)^2\right)\right) \qquad \textbf{(3.5)}$$

3.2.1. Descripción de los bloques del Decodificador

Deinterleaver

El diseño del deinterleaver es exacto al realizado por la etapa del codificador del canal. La única diferencia es que recibirá una señal con 3 soft bits, donde los 3 en conjunto tienen el mismo periodo de muestreo usado en la ecuación 3.2. En lugar de usar las ecuaciones de permutaciones especificadas en la estándar, se aprovechará los vectores generados para la etapa del deinterleaver, por lo que el único cambio adicional es agregar la siguiente ecuación en Matlab antes de la implementación.

$$[s2, dint\_idx] = sort(int\_idx) \qquad \textbf{(3.6)}$$

La idea de usar esta ecuación, es ordenar los valores usados en el vector de inicialización de la memoria ROM usada en el interleaver, y utilizar los índices generados en el arreglo s2, nombrado con la variable dint_idx, en la ROM de permutaciones del deinterleaver.

Decodificador de Viterbi

La implementación del decodificador de Viterbi, figura 3.14, se realiza primero mediante una etapa de depuncturing variable dependiendo el perfil de codificación seguido de un decodificador de Viterbi fijo, el cual tiene los mismos parámetros usados para el codificador convolucional.

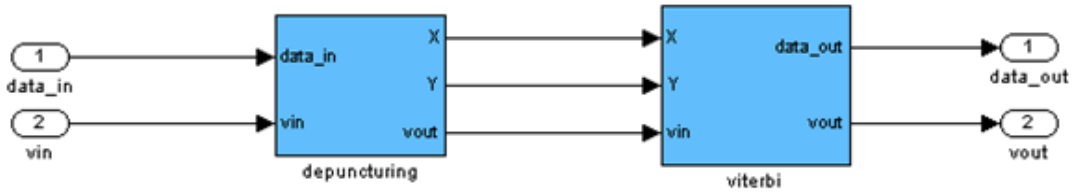

Figura 3.14 Diagrama de Bloques del Decodificador de Viterbi

El diseño del depuncturing, se muestra en la figura 3.15, el cual se compone de 3 etapas. La entrada al depuncturing serán datos de 3 soft bits. La primera etapa, expresado en el subsistema *add_erasure* se encarga se añadir un bit 0 a la parte baja de cada dato ingresado, para indicar que el dato entrante es válido y debe ser tomado en cuenta para la decodificación.

La segunda etapa se encarga de realizar la operación inversa al puncturing por lo que se necesita pasar los datos de forma serial a paralelo, cuyo número de datos dependerán del numero de 1's presentes en el vector de depuncturing, el cual debe ser el mismo que fue usado en el codificador. Dado que no se tiene conocimiento del símbolo que fue eliminado en la codificación, se inserta un símbolo de tal manera que el bit menos significativo sea igual a 1 para etiquetar este dato como símbolo *erasure* para el core del decodificador de Viterbi.

Finalmente, se separan los símbolos, 4 bits para la parte alta y 4 bits para la parte baja.

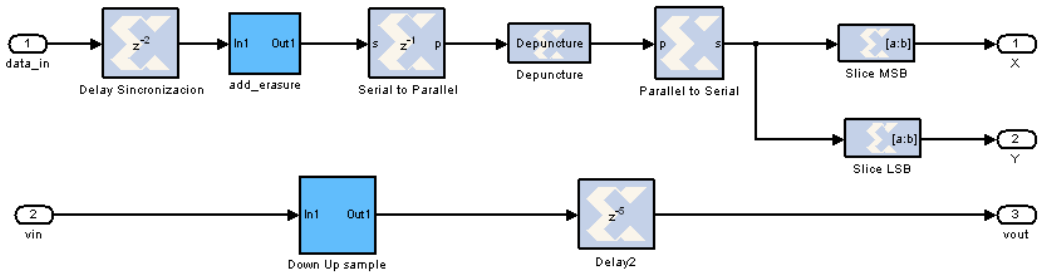

Figura 3.15 Implementación del Depuncturing

Para sincronizar este bloque, se usa el primer bloque de retardo, cuyo valor varía de acuerdo al perfil de codificación usado. Así mismo para generar la señal vout se utiliza un retardo para especificar el primer bit valido a la salida del bloque, y el subsistema *Down Up Sample*, para tener la tasa de muestro que se tenía antes de pasar por el puncturing.

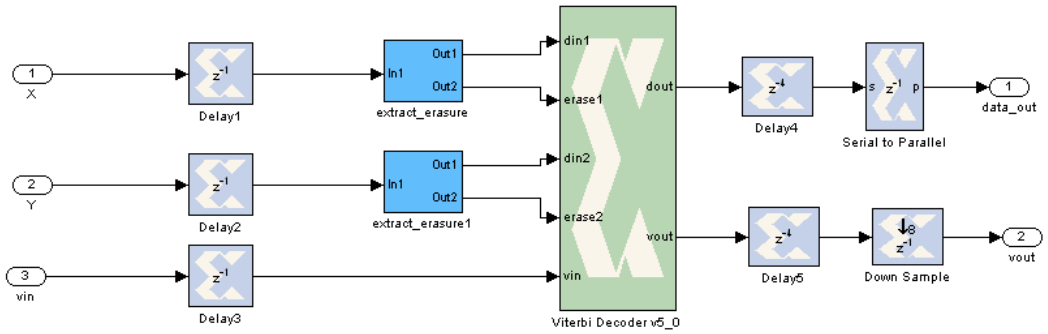

Figura 3.16 Implementación del Decodificador de Viterbi

La figura 3.16 muestra como esta interconectado el core que implementa la decodificación de Viterbi. Se utilizan los retardos a la entrada para sincronizar el bloque con el resto del sistema. Luego, al símbolo de 4 bits entrantes, se

separa en 3 bits para la entrada de datos del decodificador, y un bit de tipo bool para la entrada *erase*. Una vez realizada la decodificación, la señal se convierte de serial a paralelo, donde los bloques de retardo son usados para poder sincronizar la señal.

Al bloque del decodificador de Viterbi hay que habilitarle la opción de depuncturing externo. Como se puede observar en la figura 3.17, los parámetros son los mismos del codificador, además se le agrega el *traceback length,* que representa en la implementación el número de símbolos con valor cero antes de tener la primera salida decodificada y sirve como medio de truncación del algoritmo de Viterbi.

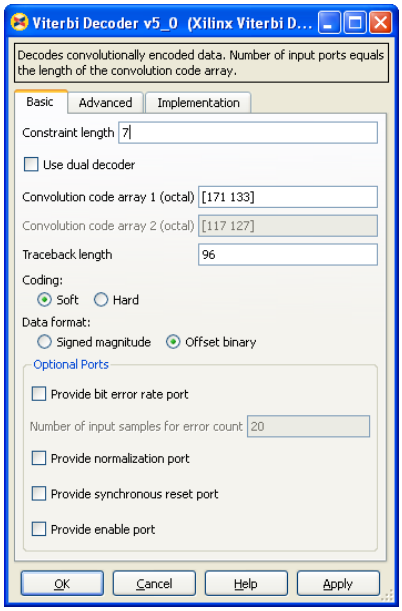

Figura 3.17 Parámetros del Core del Decodificador de Viterbi

Se elige el medio de decodificación Soft, ya que el método Hard no permite el uso de Puncturing en la codificación. El formato de los datos se elige como binario sin signo (*Offset Binary*) y se representa mediante la siguiente tabla 3.12.

Tabla 3.12 Formato de datos usado en la decodificación de Viterbi

Valor	Magnitud Binaria
1 más seguro	111
2do 1 más seguro	110
3er 1 más seguro	101
1 menos seguro	100
0 menos seguro	011
3er 0 más seguro	010
2do 0 más seguro	001
0 más seguro	000

La figura 3.18 muestra el funcionamiento del decodificador de Viterbi, donde se muestran cómo van cambiando las señales en el transcurso del tiempo.

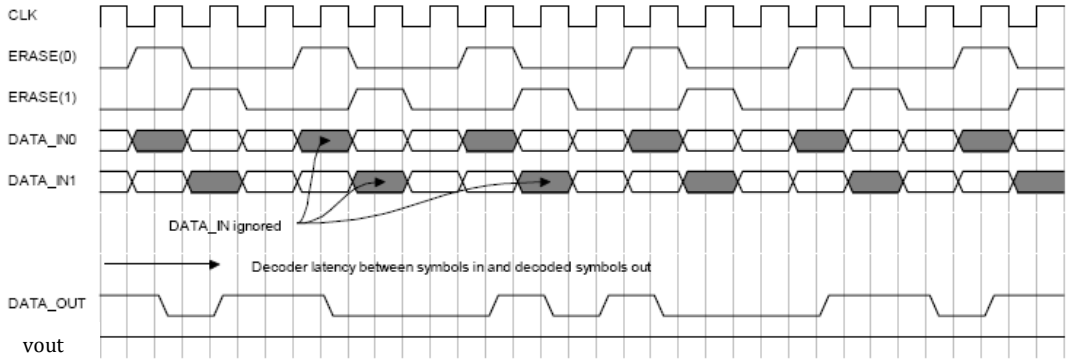

Figura 3.18 Diagrama de Tiempo del Decodificador de Viterbi

Decodificación de Reed-Solomon

El diagrama de bloques del decodificador de Reed-Solomon se muestra a continuación:

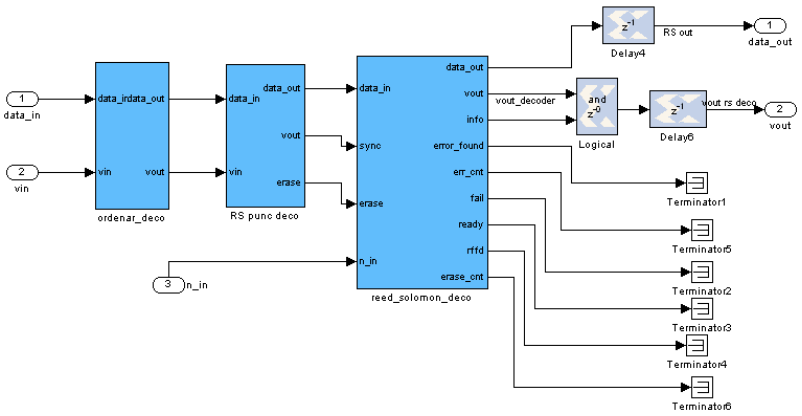

Figura 3.19 Diagrama de Bloques del Decodificador de Reed-Solomon

El proceso de decodificación se realiza en sentido inverso a la codificación. Primero se tienen que reordenar los datos, para volver a tener los símbolos de paridad al final del bloque. El diseño de esta etapa es igual a la realizada por la decodificación, donde lo único que hay que hacer es cambiar los vectores de inicialización para la memoria ROM como se muestra en la siguiente tabla.

Tabla 3.13 Vectores de inicialización para la memoria ROM en el Decodificador de Reed-Solomon

Perfil de Codificación	Vector de inicialización
1	[9:32 1:8]
2	[5:40 1:4]
3	[17:64 1:16]
4	[9:80 1:8]
5	[13:108 1:12]
6	[13:120 1:12]

El siguiente paso es realizar el depuncturing, el cual consiste en insertar ceros en el lugar donde los símbolos fueron eliminados. El diseño se muestra en la figura 3.20. El proceso es similar al realizado en la codificación, es decir la velocidad de lectura es diferente al de la escritura. El bloque *Down Up Sample,* debe volver el periodo de muestreo que tenía antes de la etapa del puncturing en la codificación, y se basa en los valores mostrados en la tabla 3.1. De igual manera, se envía una señal que indique el momento en que se inserten los ceros, realizado por el subsistema *Generar_Erase.* Para insertar los símbolos cero, la memoria RAM de doble puerto debe ser inicializada con cero en todos sus registros de memoria.

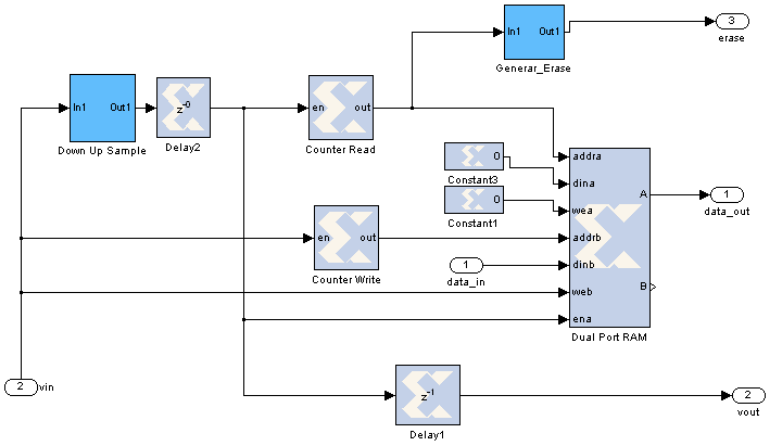

Figura 3.20 Diseño del Depuncturing del Decodificador de Reed-Solomon

Para terminar, se realiza la interconexión del core usado para la decodificación de Reed-Solomon, como se muestra en la figura 3.21. Se tienen los bloques Up Sample y Down Sample a la entrada y salida del core

respectivamente, cuyo valor es igual a la inverso del especificado como periodo del símbolo en Simulink. El bloque *info_generator* se encarga de generar la señal vout que indicara la presencia de un símbolo valido a la salida y de especificar cuáles son símbolos de información y cuales son de paridad.

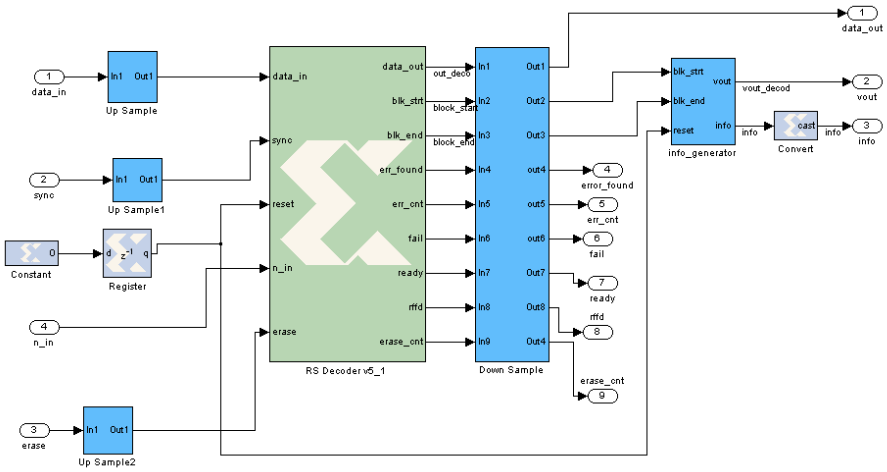

Figura 3.21 Diseño e interconexión del Decodificador de Reed-Solomon

Los parámetros del decodificador de Reed-Solomon son los mismos usados en la figura 3.6. Las señales más importantes a la salida del decodificador se muestran a continuación:

Data_out: Produce los símbolos de información y paridad.

Blk_start: Presenta un 1 al tener el primer símbolo del bloque.

Blk_end: Presenta un 1 al tener el ultimo símbolo del bloque.

Err_found: Presenta un 1 al final del bloque si se encontró un error.

Err_cnt: Presenta el número de errores que fueron corregidos en la decodificación en el instante que se presenta el ultimo símbolo del bloque.

Fail: Presenta un 1 en el instante que se presenta el ultimo símbolo del bloque si en la decodificación no se logro recuperar los símbolos de información.

3.2.2. Implementación en la FPGA del Decodificador

El análisis se realiza de igual manera al realizado en la parte del codificador donde se muestra el área usada por cada etapa del decodificador en la FPGA. Para el diseño del derandomizer se usa el mismo diseño usado en el codificador por lo que el área usada es la misma.

Tabla 3.14 Recursos de la FPGA para el Decodificador usando BPSK 1/2

Etapa	Slices	Slice Flip Flops	LUTs	IOBs	BRAMs
Deinterleaver	18	25	36	9	3
Viterbi	1604	1398	2297	7	4
Reed-Solomon	0	0	0	0	0
DeRandomizer	34	61	34	20	0

Tabla 3.15 Recursos de la FPGA para el Decodificador usando QPSK 1/2

Etapa	Slices	Slice Flip Flops	LUTs	IOBs	BRAMs
Deinterleaver	27	36	48	9	3
Viterbi	1635	1458	2337	7	4
Reed-Solomon	1604	1639	2863	12	9
DeRandomizer	34	61	34	20	0

Tabla 3.16 Recursos de la FPGA para el Decodificador usando QPSK 3/4

Etapa	Slices	Slice Flip Flops	LUTs	IOBs	BRAMs
Deinterleaver	27	38	53	9	3
Viterbi	1667	1516	2379	7	4
Reed-Solomon	1627	1678	2907	12	9
DeRandomizer	34	61	34	20	0

Tabla 3.17 Recursos de la FPGA para el Decodificador usando 16-QAM 1/2

Etapa	Slices	Slice Flip Flops	LUTs	IOBs	BRAMs
Deinterleaver	22	34	43	9	3
Viterbi	1629	1448	2323	7	4
Reed-Solomon	810	996	1359	12	6
DeRandomizer	34	61	34	20	0

Tabla 3.18 Recursos de la FPGA para el Decodificador usando 16-QAM 3/4

Etapa	Slices	Slice Flip Flops	LUTs	IOBs	BRAMs
Deinterleaver	26	39	52	3	1
Viterbi	1665	1512	2373	7	4
Reed-Solomon	1630	1680	2907	12	9
DeRandomizer	34	61	34	20	0

Tabla 3.19 Recursos de la FPGA para el Decodificador usando 64-QAM 2/3

Etapa	Slices	Slice Flip Flops	LUTs	IOBs	BRAMs
Deinterleaver	29	42	57	9	4
Viterbi	1650	1481	2359	7	4
Reed-Solomon	1647	1706	2935	12	9
DeRandomizer	34	61	34	20	0

Tabla 3.20 Recursos de la FPGA para el Decodificador usando 64-QAM 3/4

Etapa	Slices	Slice Flip Flops	LUTs	IOBs	BRAMs
Deinterleaver	29	43	58	9	4
Viterbi	1666	1519	2379	7	4
Reed-Solomon	1669	1746	2980	12	9
DeRandomizer	34	61	34	20	0

3.3. Diagrama de Bloques General

El diseño general del codificador del canal es presentada en la figura 3.22. Se utiliza una entrada de datos que pueden ser aleatorios o fijos. La señal rfd, la cual es una salida del codificador se utiliza para validar la salida de los datos de entrada. Así mismo se describe el tipo de datos usado en Simulink según cómo va transcurriendo las señales.

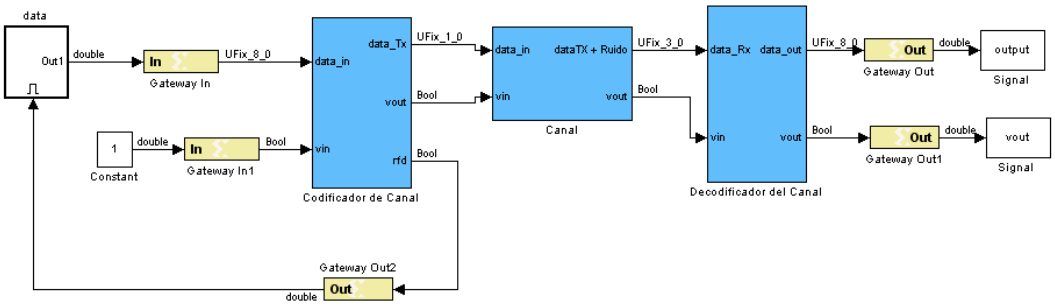

Figura 3.22 Diagrama de Bloques General para la Simulación

El diseño del canal se muestra en la figura 3.23. Este bloque sirve para poder realizar todas las simulaciones necesarias. Para poder usar esta etapa es necesario convertir las señales de punto a fijo a punto flotante para poder usar los bloques de las librerías de Simulink.

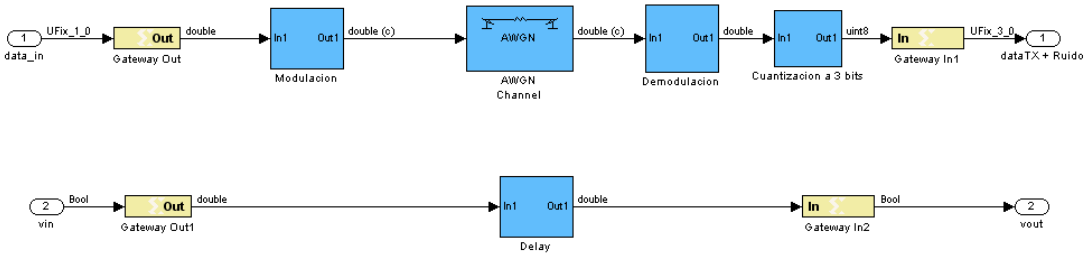

Figura 3.23 Diseño del Canal usado en la simulación del sistema

La modulación y demodulación usada depende del perfil de codificación especificado. Dado que los datos están en forma de bit, es decir, uno o cero, entonces hay que configurar los bloques para poder trabajar a ese nivel. Se utiliza codificación Gray y se normalizan los datos mapeados con la potencia promedio para cumplir con las características del estándar. Para la demodulación se tiene que configurar los bloques para usar detección por *soft decision* acorde a la ecuación 3.3.

Es necesario trabajar con el periodo de muestreo de la salida del codificador. El bloque de retardo usado para generar el bit valido depende de la modulación usada, siendo de 1, 2, 4 y 6 unidades, para BPSK, QPSK, 16QAM y 64QAM respectivamente. El canal será simulado usando ruido aditivo gaussiano.

3.4. Diseño del Plan de Pruebas

Una vez terminado el diseño, tenemos que verificar que cada etapa funcione correctamente de acuerdo al estándar. Los pasos a tener en cuenta son los siguientes:

- Verificación de cada etapa del codificador de acuerdo a los vectores de prueba especificados en el estándar.
- Verificar si el decodificador recupera los datos enviados por el codificador para cada etapa del diseño.
- Verificar todo el diseño en conjunto.
- Analizar la probabilidad de error de bit para cada perfil de codificación. Esta debe mejorar comparándolo para un diseño sin codificación.
- Verificar el diseño en Hardware usando cosimulación. Los datos a la salida del codificador del canal usando la FPGA deben ser los mismos.

Los vectores de prueba especificados en el estándar se dan para el perfil de codificación 2, es decir, usando modulación QPSK y una tasa de codificación 3/4. Además para el vector de inicialización del Randomizer se usa lo siguiente: Número de símbolo sin ráfaga: 1, UUIC: 7, BSID: 1, número de trama: 1.

Entrada de datos en notación Hexadecimal:

45 29 C4 79 AD 0F 55 28 AD 87 B5 76 1A 9C 80 50 45 1B 9F D9 2A 88 95 EB AE B5 2E 03 4F 09 14 69 58 0A 5D

Datos a la Salida del Randomizer

D4 BA A1 12 F2 74 96 30 27 D4 88 9C 96 E3 A9 52 B3 15 AB FD 92 53 07 32 C0 62 48 F0 19 22 E0 91 62 1A C1

Datos a la Salida del Codificador de Reed-Solomon

49 31 40 BF D4 BA A1 12 F2 74 96 30 27 D4 88 9C 96 E3 A9 52 B3 15 AB FD 92 53 07 32 C0 62 48 F0 19 22 E0 91 62 1A C1 00

Datos a la Salida del Codificador Convolucional

3A 5E E7 AE 49 9E 6F 1C 6F C1 28 BC BD AB 57 CD BC CD E3 A7 92 CA 92 C2 4D BC 8D 78 32 FB BF DF 23 ED 8A 94 16 27 A5 65 CF 7D 16 7A 45 B8 09 CC

Datos a la Salida del Interleaver

77 FA 4F 17 4E 3E E6 70 E8 CD 3F 76 90 C4 2C DB F9 B7 FB 43 6C F1 9A BD ED 0A 1C D8 1B EC 9B 30 15 BA DA 31 F5 50 49 7D 56 ED B4 88 CC 72 FC 5C

Como método de prueba, el vector usado debe repetirse para cada ráfaga de datos para probar que el codificador funciona para un flujo de datos continuos. Para cada etapa se utilizan diagramas de tiempo para analizar cómo van cambiando las señales de acuerdo a la señal de reloj.

Una vez realizadas estas pruebas, se procede a verificar el codificador en hardware, según como se especifica en la figura 3.24. En este caso se realiza una cosimulación, donde los datos de entrada pasan a la FPGA y esta los regresa al computador para que puedan ser procesados por el decodificador.

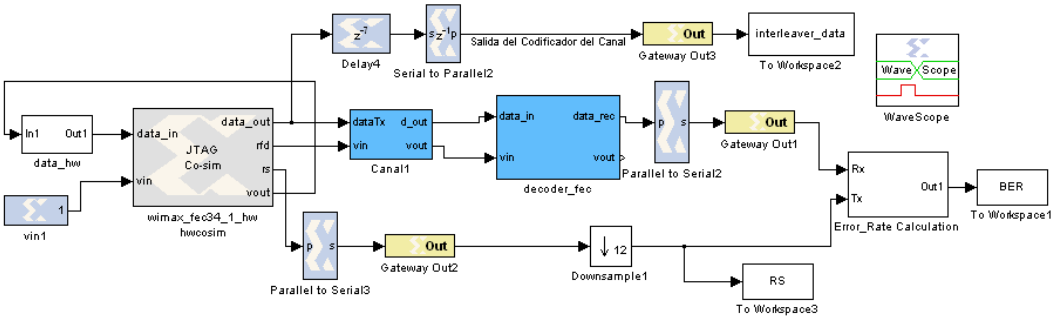

Figura 3.24 Verificación del Codificador del Canal en Hardware

De la figura 3.24, se observa el bloque del codificador, el cual tiene las 3 salidas explicadas con anterioridad, y la señal RS. Esta señal es la salida del decodificador de Reed-Solomon antes de pasar por la etapa de puncturing y el reordenamiento de los bytes de paridad. Esta señal debe ser comparada con la salida del decodificador para obtener la probabilidad de error del sistema que debe ser igual a cero para una relación de señal a ruido alta. Se compara con la señal RS ya que de esa manera de verifica si el decodificador de Reed-Solomon puede recuperar los símbolos de paridad

eliminados por el puncturing. Además para verificar que en hardware el codificador funciona según el estándar, se ingresa como entrada la señal de prueba y se verifica la salida del codificador del canal, el cual es la salida del interleaver.

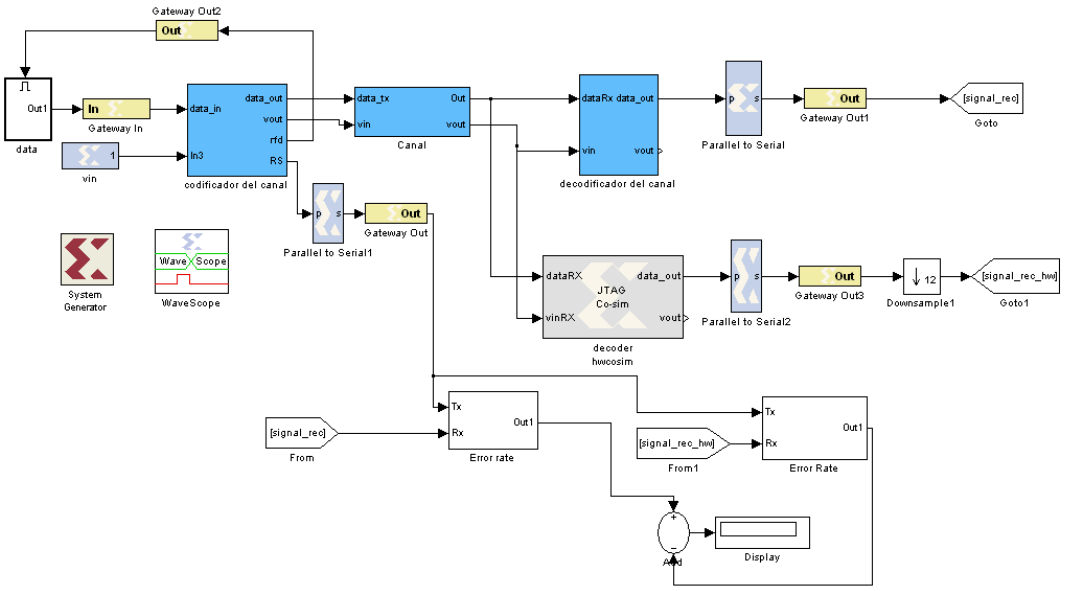

Figura 3.25 Verificación del Decodificador del Canal en Hardware

De igual manera como se verifico el codificador del canal en hardware, hay que hacerlo para el decodificador. En la figura 3.25 se muestra el diseño realizado para obtener estos datos. Se utiliza la misma señal transmitida más ruido para el decodificador en hardware y el usado en la simulación. Para probar el funcionamiento del decodificador y consecuentemente la capacidad de corrección de errores del sistema, se calcula el BER para diferentes

valores de relación de señal a ruido. Los dos resultados tienen que ser similares, lo cual se refleja en el display mostrado en la grafica. En base a este grafico, se tienen que comparar todos los perfiles de codificación y modulación especificados en el estándar y desarrollados en este trabajo.

CAPÍTULO IV

4. EXPERIMENTACIÓN Y ANÁLISIS DE RESULTADOS

4.1. Variación del BER según el SNR

La mejor forma de verificar la fiabilidad del sistema diseñado es midiendo la probabilidad de error de bit (BER) para diferentes valores de relación señal a ruido (SNR). Según lo expresado en el capítulo 2, utilizando codificación de canal se puede detectar y corregir errores causados por el canal de comunicaciones, por lo que el BER tiene que mejorar comparado a un sistema que no utilice codificación.

Para verificar que existe una mejora en el BER para un sistema con codificación, se realizan pruebas usando todo el sistema en conjunto, y

además se analizará las variaciones que pueden ocasionar cada una de las etapas usadas en el diseño.

En cada grafica se muestra el BER como una función de la energía de bit por radio de ruido (Eb/No), el cual es una medida de la eficiencia de energía en cada esquema de modulación y codificación.

4.1.1. Comparación entre un sistema sin codificación y uno con codificación según su modulación

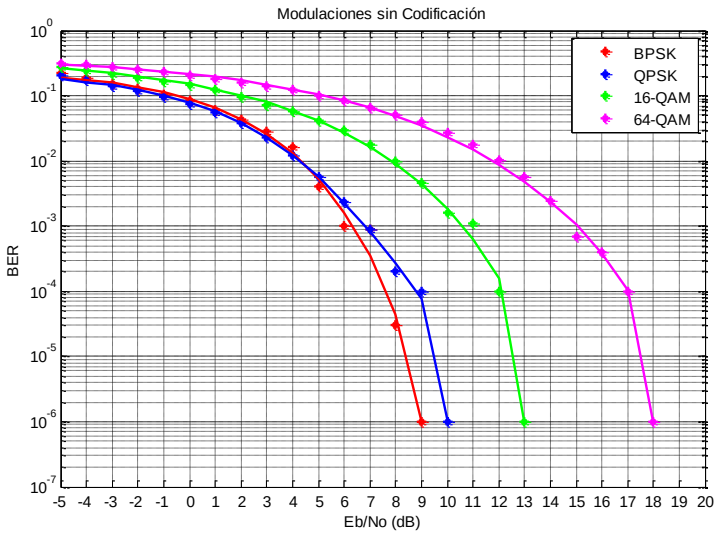

Figura 4.1 Comparación de las modulaciones usadas en Wimax sin usar codificación

Antes de realizar el análisis del sistema con codificación, se muestra en la figura 4.1 las diferentes modulaciones usadas en el estándar de Wimax. La simulación fue realizada utilizando la misma trama de datos aleatoria y la demodulación usando detección Hard.

De la grafica podemos concluir que mientras mayor es el número de puntos en la constelación, el desenvolvimiento de la señal con presencia de ruido disminuye. Además usando modulación BPSK y QPSK se obtienen comportamientos similares donde se observa que BPSK tiene una mejora en 1 dB, pero QPSK presenta un mejoramiento en la velocidad de los datos al poder transmitir una mayor cantidad de datos.

Para 16 QAM y 64-QAM las curvas están desplazadas más a la derecha, lo que demuestra que mientras mayor es la distancia entre los puntos de la codificación, habrá mayor tolerancia al ruido.

La siguiente tabla muestra los valores de Eb/No a los que se obtiene una probabilidad de error de bit de 10^{-3}.

Tabla 4.1 Eb/No requerido para tener BER de $\mathbf{10^{-3}}$

Modulación	**BPSK**	**QPSK**	**16-QAM**	**64-QAM**
E_b/N_o **con BER=** $\mathbf{10^{-3}}$**.**	6	7	10.5	15
E_b/N_o **con BER=** $\mathbf{10^{-6}}$**.**	9	10	13	18

Modulación BPSK

La primera usando codificación será para modulación BPSK. El estándar IEEE 802.16-2004 especifica un sistema con tasa de codificación 1/2. Este sistema es el más sencillo en el diseño ya que solo usa el codificador

convolucional nativo, ya que al ser BPSK la modulación que menos ancho de banda ocupa no hay necesidad de usar puncturing.

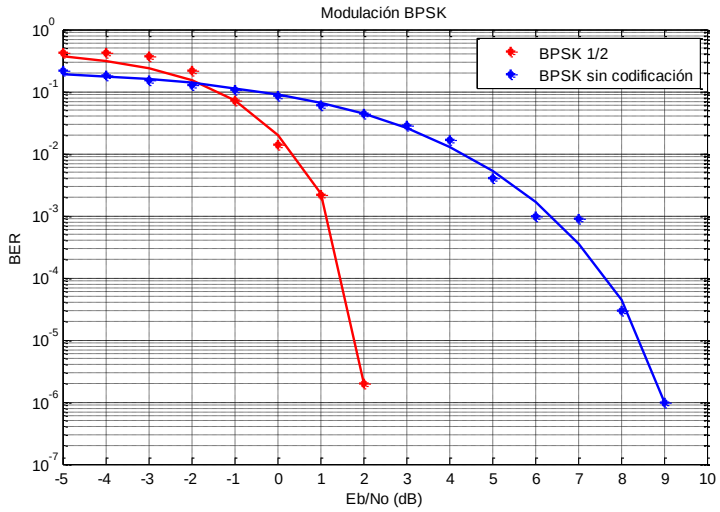

Figura 4.2 Comparación de la Modulación BPSK con codificación y sin codificación

En la figura se muestra como mejora la curva del BER vs EbNo al utilizar codificación, siendo la mejora de 5 dB. Esto se logra al usar el nivel más alto de codificación convolucional. Cabe destacar que cuando se llega a una probabilidad de error de 10^{-6} la simulación ya no mostraba ningún error y etiquetaba el BER como cero.

Modulación QPSK

Para la modulación QPSK se tienen 2 perfiles de codificación según lo especificado en el estándar IEEE 802.16-2004. Se utilizan tasas de codificación 1/2 y 3/4, ambas usando codificación concatenada RS-CC con un interleaver. Los resultados se muestran en la siguiente figura:

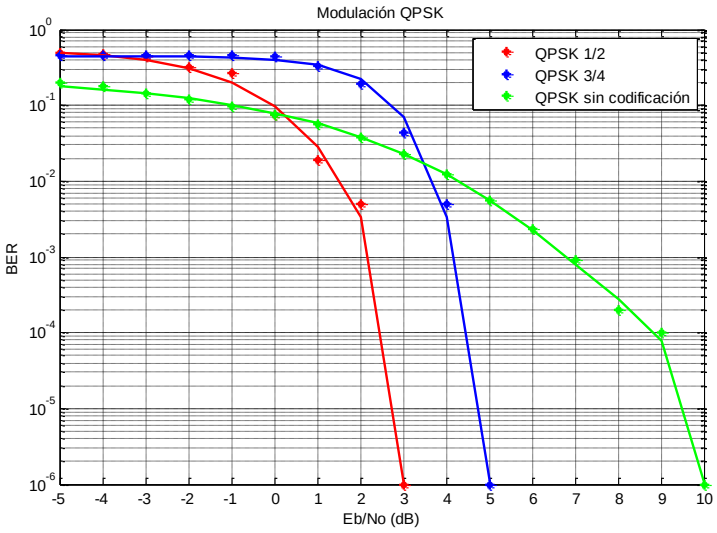

Figura 4.3 Comparación de la Modulación QPSK con codificación y sin codificación

La figura 4.3 muestra la comparación de los sistemas usados en Wimax cuya modulación sea QPSK para un sistema sin codificación y con codificación. Los resultados muestran que se logra una mejora usando codificación, siendo esto más notorio cuando se utiliza una tasa de codificación de 1/2, donde se tiene una mejora con una ganancia de codificación de 5 dB comparado con los 3 dB que se tienen usando una tasa de codificación 3/4.

Para los dos sistemas de codificación se tiene el mismo tamaño de bloque al final, y esto se logra para el caso de QPSK 3/4 disminuyendo la capacidad de corrección del codificador convolucional y del codificador de Reed-Solomon.

Para la tasa de codificación ½ hay un comportamiento similar al usado con BPSK, ya que en ambos existe una ganancia de codificación de 5 dB, por lo que resulta el perfil adecuado cuando se quiere tener un sistema fiable.

Modulación 16-QAM

Para la modulación 16-QAM se tienen 2 perfiles de codificación según lo especificado en el estándar IEEE 802.16-2004. Se utilizan tasas de codificación 1/2 y 3/4, ambas usando codificación concatenada RS-CC con un interleaver. Los resultados se muestran en la siguiente figura:

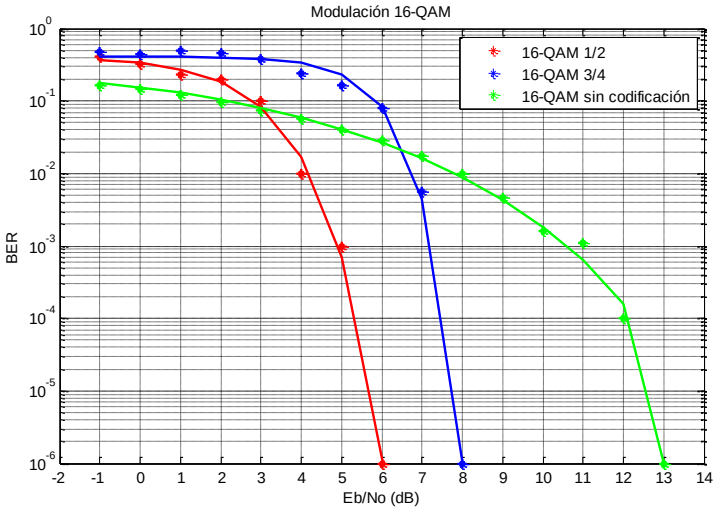

Figura 4.4 Comparación de la Modulación 16-QAM con codificación y sin codificación

Los resultados son similares a los que se obtuvieron usando una codificación, aunque aclarando que las curvas están desplazadas a la derecha debido a los efectos de la modulación 16-QAM, donde los símbolos

en la constelación están más cercanos entre sí lo que dificulta la detección en presencia de ruido.

Entre los dos perfiles de codificación existe una diferencia de cerca de 2 dB, y se obtuvo una ganancia de codificación de 5.5 dB para 16-QAM 1/2, y de 4 dB para 16-QAM ¾, mediciones hechas para un BER de 10^{-3}. De igual manera cuando se llega a 10^{-6} se tiene un sistema libre de errores.

Para este sistema se utilizan las mismas tasas de codificación para el codificador convolucional usadas para QPSK y se aumento el tamaño del bloque codificado, además que para el perfil 16-QAM ½ se usa en el codificador de Reed-Solomon todos los símbolos de paridad, siendo este el único perfil en que no se utiliza el puncturing para el codificador exterior.

Modulación 64-QAM

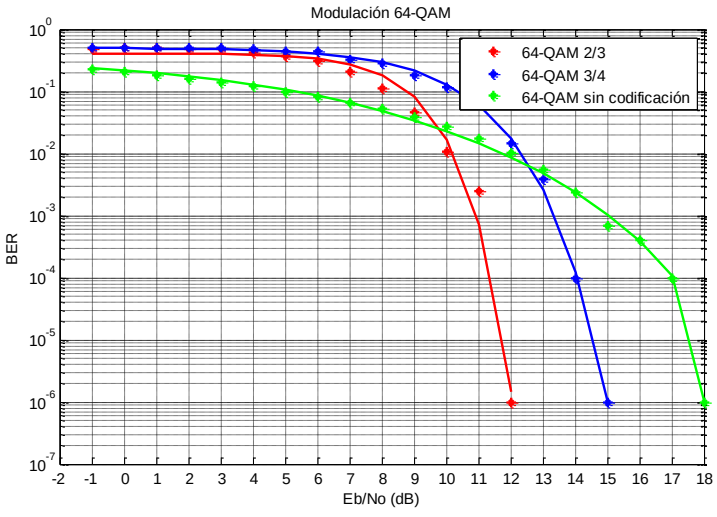

Figura 4.5 Comparación de la Modulación 64-QAM con codificación y sin codificación

Para la modulación 64-QAM se tienen 2 perfiles de codificación según lo especificado en el estándar IEEE 802.16-2004. Se utilizan tasas de codificación 2/3 y 3/4, ambas usando codificación concatenada RS-CC con un interleaver.

Las curvas, como se puede observar en la figura 4.5, se desplazan más a la derecha al tener un sistema de modulación donde la detección es mucho más complicada al tener en este caso 64 puntos en la constelación, por lo que este sistema se tiene que usar cuando se necesita tener velocidades de datos superiores.

A diferencia de las modulaciones QPSK o 16-QAM, aquí se usa para el primer perfil de codificación una tasa de codificación 2/3, el cual ofrece una vulnerabilidad contra el ruido inferior pero que se ajusta dado las características de este tipo de modulación.

Entre los dos perfiles de codificación existe una diferencia de cerca de 2 dB, y se obtuvo una ganancia de codificación de 4 dB para 64-QAM 2/3, y de 2 dB para 64-QAM 3/4, mediciones hechas para un BER de 10^{-3}. Sin embargo cuando la curva está más estabilizada se encuentra una ganancia de codificación de 6 dB para el primer perfil de codificación usado y de 3 dB para el segundo. Comparado a las simulaciones realizadas para las otras modulaciones, se comprueba que con una modulación 64-QAM el

sistema se vuelve más vulnerable al ruido, pero aun así fue posible mejorar el sistema usando un sistema con codificación del canal.

Para el diseño del codificador de Reed-Solomon se usaron 12 bytes de paridad para cada sistema, y para el codificador convolucional se uso para el primer perfil una tasa de codificación de 3/4 y para el segundo se disminuye la capacidad de corrección usando una tasa de codificación de 5/6.

Al comparar las graficas para cada modulación se puede observar que para valores bajos de Eb/No no se observa una mejoría para el BER. Esto se debe a que para estos valores el demodulador Soft Decision usado en el diseño recupera la señal con muchos errores y el codificador de canal no puedo recuperarlos. Por tal motivo, el análisis se realiza cuando la curva tiende a valores con probabilidad bien bajos, es decir que pasen de 10^{-3} **,** tal como están contemplados en trabajos afines (11), (7).

Para terminar el análisis hecho para el diseño de cada perfil de codificación y modulación del estándar de Wimax IEEE 802.16-2004, se presenta en la figura 4.6 una comparación de cada uno de los perfiles

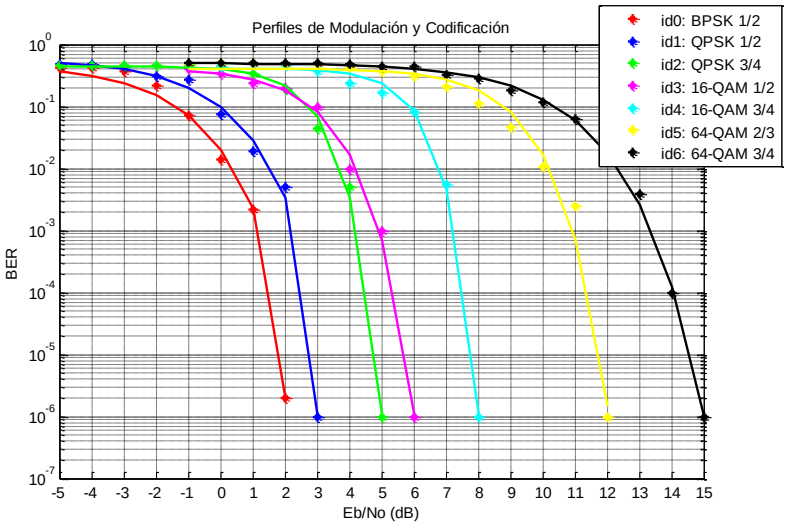

Figura 4.6 Resultados de la Simulación de los Perfiles de Codificación y Modulación

De este grafico, podemos concluir que mientras va aumentando el id especificado en el estándar, que especifica el tipo de codificación y la modulación usada, la curva del BER en función de la relación de la señal a ruido va desplazándose a la derecha. Como resultado de la simulación se tomaron los valores de la energía de bit a ruido (Eb/No) cuando se tiene un BER de 10^{-3} en la tabla 4.2:

Tabla 4.2 Eb/No requerido para tener BER de 10^{-3} usando Codificación

Modulación	**BPSK**	**QPSK**		**16-QAM**		**64-QAM**	
Tasa de Codificación	**1/2**	**1/2**	**3/4**	**1/2**	**3/4**	**2/3**	**¾**
E_b/N_o**(dB) con una BER=**10^{-3}	1	2	4.2	5	7.2	11	13.5
E_b/N_o**(dB) con una BER=**10^{-6}	2	3	5	6	8	12	15

4.1.2. Simulación del Codificador de Reed-Solomon

Luego de haber analizado el comportamiento de la codificación en su totalidad, se muestra el funcionamiento de las etapas del codificador que tienen influencia en la probabilidad de error. En este caso se va a analizar la funcionalidad de la codificación de Reed-Solomon según los datos obtenidos en la simulación.

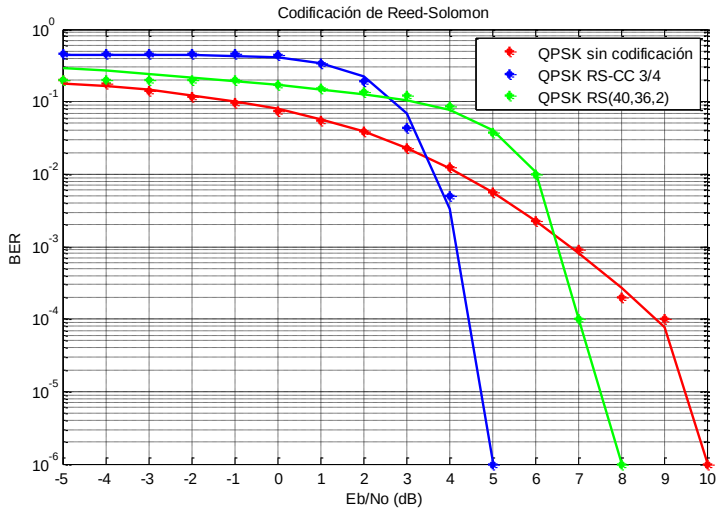

Figura 4.7 Comparación de los efectos del Codificador Reed-Solomon con un Codificador Concatenado RS-CC y un sistema sin codificación

En la figura 4.7 se muestran los resultados de los datos obtenidos usando el diseño del codificador de Reed-Solomon del perfil de codificación 2, el cual usa modulación QPSK. Para analizar el comportamiento del codificador, en la grafica se pueden encontrar también los resultados usando una codificación

concatenada RS-CC y los resultados de la simulación cuando no se utiliza ninguna codificación.

De la grafica se observa que hay una ganancia de codificación de 2 dB medido para un BER de 10^{-6}, ya que recién cuando la energía de bit a densidad ruido está en 6 dB, recién comienza notarse los efectos del codificador de Reed-Solomon. Así mismo, comparado con el sistema de codificación concatenado RS-CC, se tiene una diferencia de 3 dB entre las 2 curvas, con lo que se demuestra la ventaja de usar este sistema, además que por las características de la simulación se notan más los efectos del codificador convolucional al tener un ruido aleatorio.

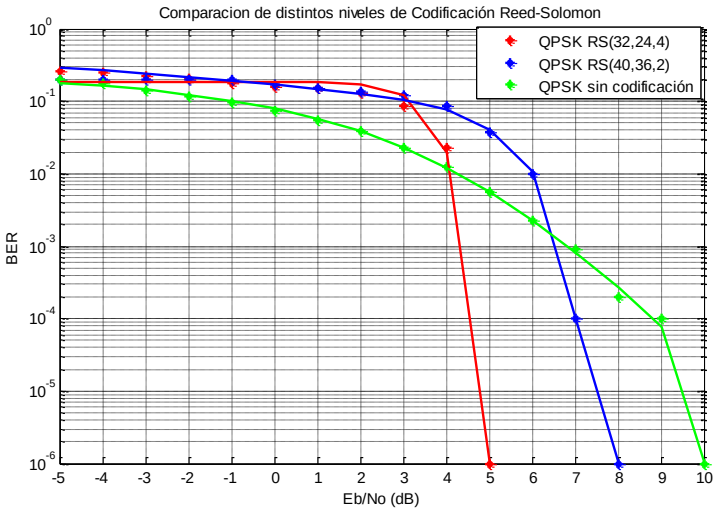

Figura 4.8 Comparacion de los dos perfiles de Codificación Reed-Solomon presentes en el estándar Wimax

Cabe mencionar que dada las características que este codificador RS(40,36,2) posee, este solo puede corregir 2 bytes. Para poder comparar las características se muestra en la figura 4.8 las características de la codificación Reed-Solomon presente en el estándar para la modulación QPSK.

En esta grafica podemos comprobar que cambiando el número de bytes que pueden ser corregidos, en este caso 8 bytes (usando codificación RS (32,24,4)), se logra mejorar la probabilidad de error en cerca de 3 dB.

4.1.3. Simulación del Codificador Convolucional

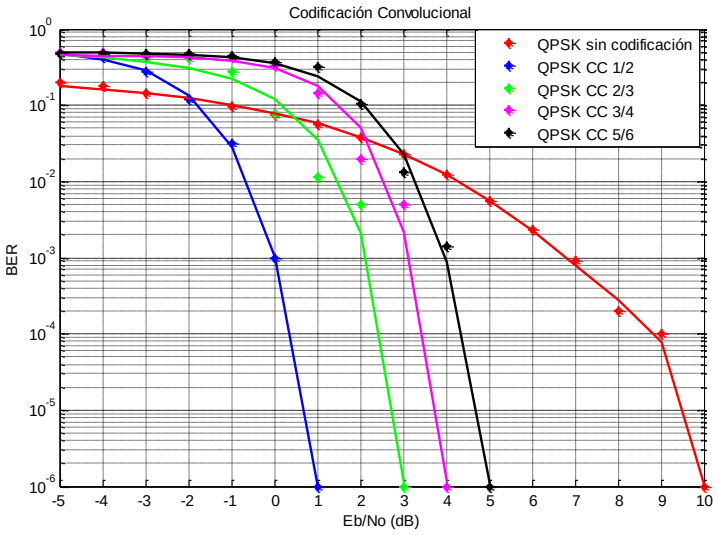

Figura 4.9 Comparación de las distintas tasas de codificación para el Codificador Convolucional

Para analizar los efectos del codificador convolucional se analizan todas las tasas de codificación especificadas en el estándar y las cuales han sido implementadas y simuladas.

En la figura 4.9 se muestran la grafica del BER vs EbNo usando modulación QPSK y usando solo el codificador convolucional.

Con los resultados de la simulación mostrada en la figura 4.9 se puede comprobar que mientras la tasa de codificación sea cercana a 1/2, mejor va a ser el desempeño del codificador convolucional frente al ruido. Además para el caso del codificador convolucional nativo, cuya tasa de codificación es 1/2, existe una considerable diferencia con respecto a los que diseños que utilizan puncturing. En la figura se observa que la ganancia de codificación es de 7 dB para este caso, superior a los 5,4 y 3 dB de ganancia de código para las tasas de codificación 2/3, 3/4 y 5/6 respectivamente.

El motivo por lo que el sistema se vuelve más vulnerable a ruido es que se han eliminado varios bits los cuales en el momento de la decodificación ingresan con un valor nulo el cual no es tomado en cuenta para volver a tener la señal original

4.1.4. Efectos del Interleaver en la Codificación del Canal

Los efectos del interleaver han sido también analizados.. Los efectos del interleaver en el codificador concatenado RS-CC y cuyos datos son

modulados por QPSK, 16-QAM y 64-QAM y pasados por un canal AWGN son mostrados en las siguientes figuras:

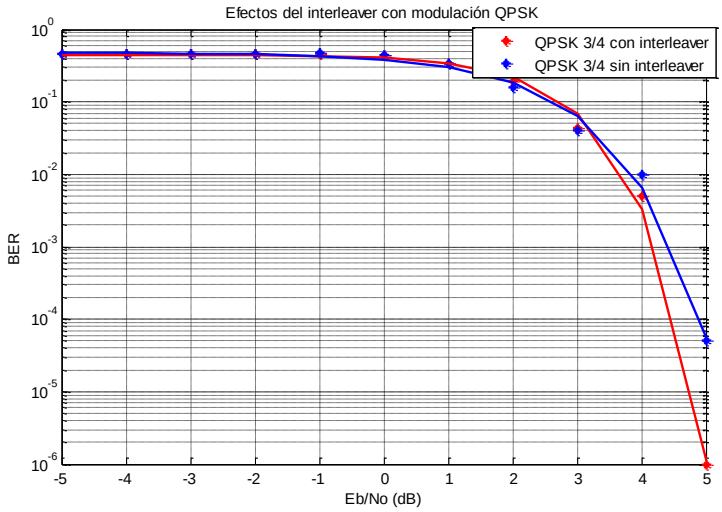

Figura 4.10 Efectos del Interleaver usando QPSK 3/4

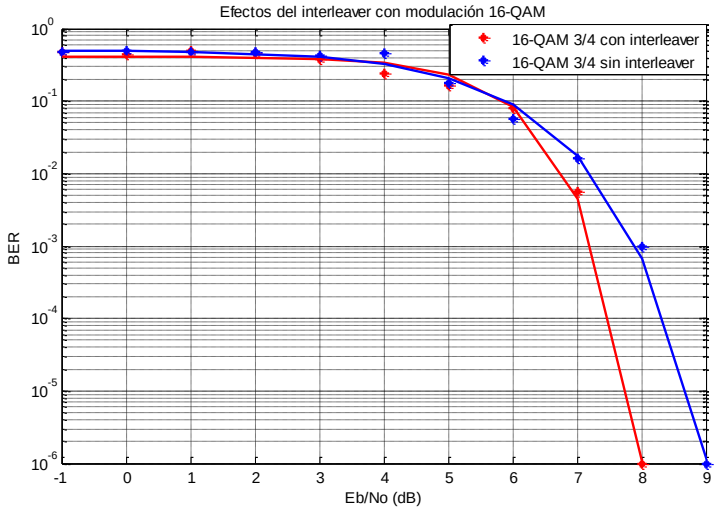

Figura 4.11 Efectos del Interleaver usando 16-QAM 3/4

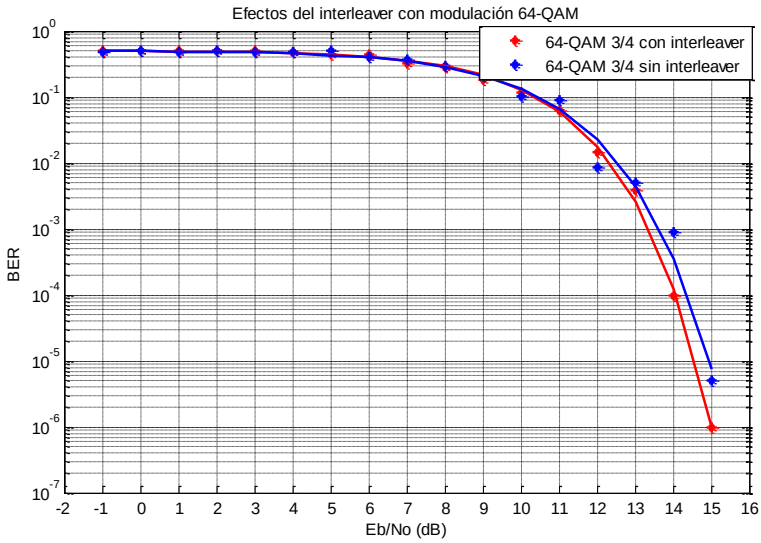

Figura 4.12 Efectos del Interleaver usando 64-QAM 3/4

Los resultados obtenidos presentan una ligera mejoría de la probabilidad de error, lo que demuestra que es importante su uso para tener un mejor desempeño frente al ruido comparado al usado cuando se tiene solo un codificador concatenado RS-CC. En los casos en que el interleaver va a tener un mejor desempeño será cuando se presenten mayores ráfagas de ruido y que afecten a una mayor cantidad de bits.

4.2. Comparación de Constelaciones para valores extremos de SNR a la salida del canal

Para analizar las constelaciones de las señales para cada modulación, se toman en cuenta los valores de la energía a bit por densidad de ruido

tomados de las tablas 4.1 y 4.2. El objetivo de mostrar estas graficas será tener una visualización de cómo es la señal recibida, la que tiene que pasar por el demodulador y el decodificador, y al tomar los valores de la tabla 4.2, quiere decir que la probabilidad de error para esa señal recibida es de 10^{-3}.

Las señales transmitidas son las mismas que han sido especificadas por el estándar de Wimax y que se describen en la Figura 1.8

4.2.1. Constelaciones para la Modulación BPSK

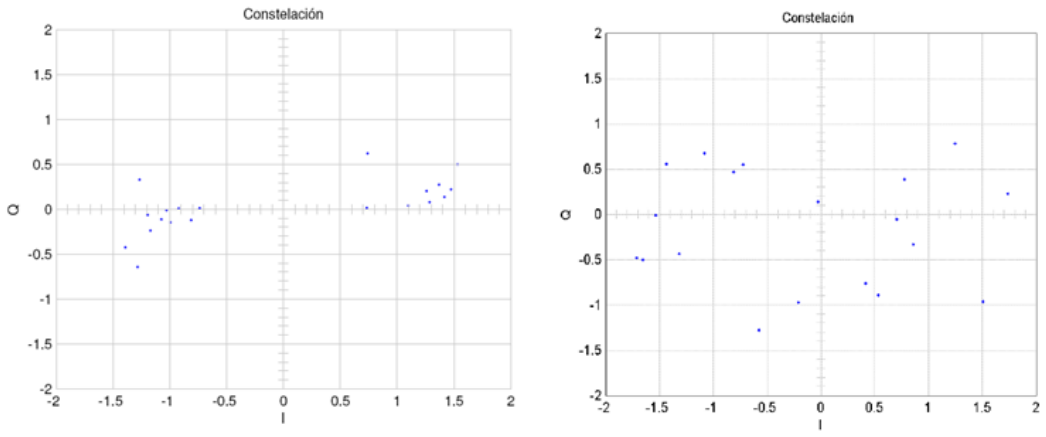

Figura 4.13 a) Señal BPSK sin codificación **b)** Señal BPSK con codificación 1/2

Usando modulación BPSK sin codificación se necesita un Eb/No de 6 dB para lograr una probabilidad de error de 10^{-3} , mientras que usando codificación de ½ se encontró que con un Eb/No de 1 dB se logra la misma probabilidad de error.

Al comparar la figura 4.14, donde la constelación de la derecha representa el caso con codificación, podemos notar que para tener el mismo efecto sin usar codificación los puntos en la constelación deben ser cercanos entre sí, lo cual se logra con una relación de señal a ruido alta, por lo que las ventajas de usar codificación se pueden comprobar en esta grafica.

4.2.2. Constelaciones para la Modulación QPSK

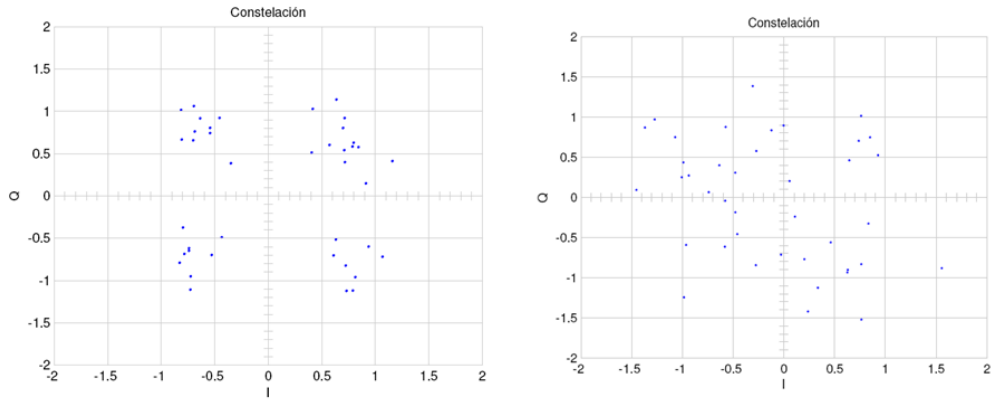

Figura 4.14 a) Señal QPSK sin codificación **b)** Señal QPSK con codificación 1/2

En la figura 4.17 podemos observar la señal QPSK recibida para un sistema sin codificación y otro con codificación 1/2. En el primer caso se uso 7 dB y en el segundo 2 dB, teniéndose un ahorro de 5 dB usando codificación y cuyos resultados en la calidad de la señal pueden ser apreciados.

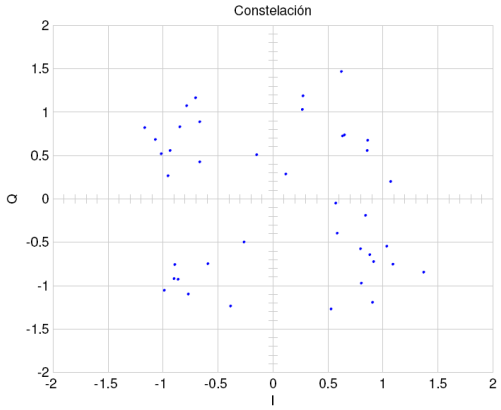

Figura 4.15 Señal QPSK con codificación 3/4

Para el perfil de codificación 2, el cual usa una tasa de codificación 3/4 se muestran los resultados en la figura 4.15. Esta constelación tiene una mejor calidad que la anterior, pero se necesitan de 4.2 dB para lograrla, lo cual aun es mejor al caso en que no se usa codificación.

4.2.3. Constelaciones para la Modulación 16-QAM

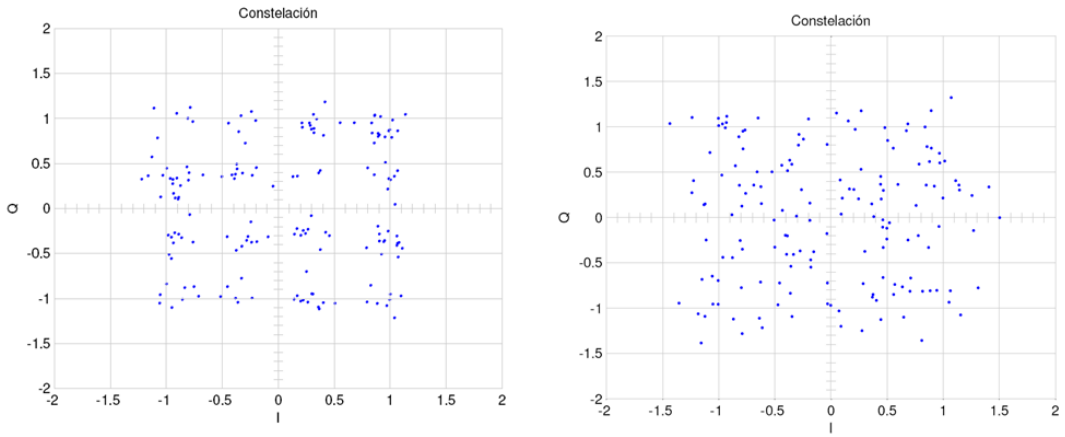

Figura 4.16 a) Señal 16-QAM sin codificación **b)** Señal 16-QAM con codificación 1/2

En la figura 4.16 podemos observar la señal 16-QAM recibida para un sistema sin codificación y otro con codificación 1/2. En el primer caso se uso 10.5 dB y en el segundo 5 dB, teniéndose un ahorro de 5.5 dB usando codificación y cuyos resultados en la calidad de la señal recibida pueden ser apreciados.

Para el perfil de codificación 4, el cual usa una tasa de codificación 3/4 se muestran los resultados en la figura 4.17. Esta constelación tiene una mejor calidad que la anterior, pero se necesitan de 7.2 dB para lograrla, lo cual aun es mejor al caso en que no se usa codificación.

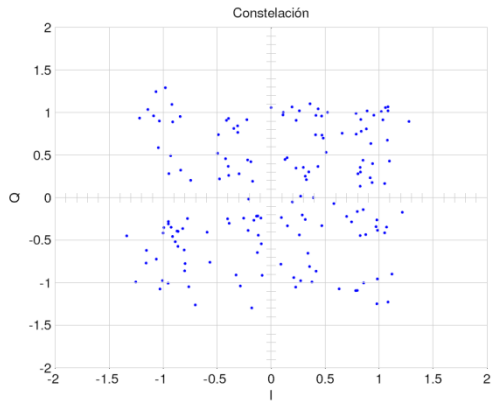

Figura 4.17 Señal 16-QAM con codificación 3/4

4.2.4. Constelaciones para la Modulación 64-QAM

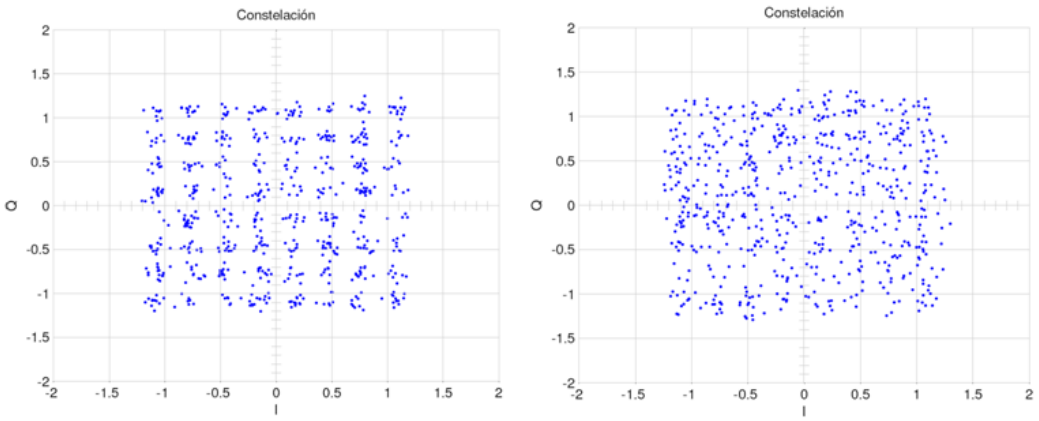

Figura 4.18 a) Señal 64-QAM sin codificación **b)** Señal 64-QAM con codificación 2/3

En la figura 4.18 podemos observar la señal 64-QAM recibida para un sistema sin codificación y otro con codificación 2/3. En el primer caso se uso 15 dB y en el segundo 11 dB, teniéndose un ahorro de 4 dB usando codificación y cuyos resultados en la calidad de la señal recibida pueden ser apreciados.

Para el perfil de codificación 6, el cual usa una tasa de codificación 3/4 se muestran los resultados en la figura 4.19. Esta constelación tiene una mejor calidad que la anterior, pero se necesitan de 13.5 dB para lograrla, lo cual aun es mejor al caso en que no se usa codificación.

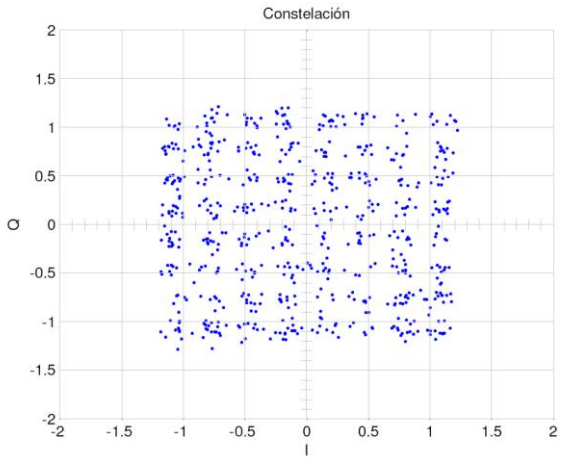

Figura 4.19 Señal 64-QAM con codificación 3/4

4.3. Pruebas y Análisis de Resultados

Luego de realizar las pruebas especificadas en la sección 3.4, se encuentra que todas las etapas del codificador y decodificador funcionan en base al estándar IEEE 802.16-2004. Las pruebas realizadas en Hardware demuestran que el diseño tiene la misma funcionalidad que el realizado en la simulación de Simulink, en cual en conjunto con Matlab resultaron muy eficaces para el análisis, al poder ingresar fuentes de datos y poder visualizar las señales en cada instante de tiempo.

En la figura 4.20 se muestran los resultados al enviar las señales de prueba especificadas en el estándar donde se está repitiendo la misma ráfaga de datos.

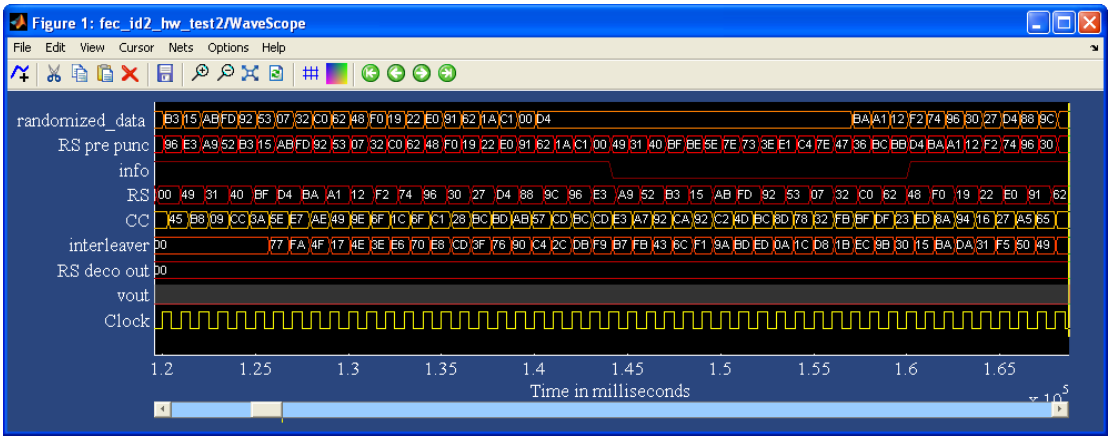

Figura 4.20 Diagrama de tiempo con los resultados

Los gráficos del BER vs SNR reflejan las características de corrección de errores presentes en el diseño, donde para poder realizar las simulaciones se convirtieron los datos enviados por la FPGA a Simulink para realizar ahí las modulaciones tal como están especificadas en el estándar y además con un canal AWGN para simular la presencia de ruido. Así mismo se verificaron las características de cada etapa, por lo que se pueden apreciar las virtudes de cada etapa usada en el diseño. De la simulación realizada para el codificador convolucional se pudo comprobar que sin utilizar puncturing se obtiene el mejor rendimiento del codificador frente al ruido, el cual disminuye conforme se utiliza puncturing y la tasa de codificación se va acercando a 1, por lo que las tasas de codificación que tienen una mayor distancia libre (especificado en Tabla 2.4) podrán corregir una mayor cantidad de errores. La mayoría de los errores ingresados a la señal son corregidos por esta etapa.

En la simulación de la etapa de codificación de Reed-Solomon, se comprobó la capacidad de corregir errores de la señal al mejorar la curva del BER conforme disminuye la relación de señal a ruido. De igual manera se comprobó la capacidad de corrección de errores usando las modificaciones presentes en el estándar (Tabla 2.3), comprobando que mientras mayor sea el valor de t, la capacidad de corrección de errores aumenta.

El interleaver también fue verificado, haciéndose un análisis en tiempo para comprobar que las permutaciones eran realizadas correctamente y utilizando el deinterleaver para aceptar un solo bit en lugar de los 3 bits. Además se demostró su funcionalidad para ayudar al decodificador de Viterbi al poder aleatorizar la posición de los errores causados por el canal de comunicaciones.

Las pruebas realizadas al Randomizer incluyeron demostrar que funciona para el estándar en base a los vectores de prueba y utilizando el mismo diseño para el Derandomizer para verificar la decodificación de los datos. Además se diseño un circuito de control para inicializar el valor de semilla usado para el LFSR para cada ráfaga de datos por medio de la señal reset.

CONCLUSIONES Y RECOMENDACIONES

1. Con el desarrollo de este trabajo se ha podido comprobar las ventajas al utilizar un ambiente de desarrollo de Hardware de alto nivel, el cual es de gran utilidad al momento del diseño ya que evita el uso de tener que escribir el código usando VHDL, quedando este pasó destinado al momento de la generación al hardware.

2. En base a las simulaciones realizadas se concluye que el diseño del codificador cumple con las características especificadas con el estándar, al obtener la secuencia de bits deseados a la salida en base al vector de prueba utilizado como entrada. Dado que el estándar solo especifica un vector de prueba usando codificación QPSK 1/2, este diseño fue el primero en realizarse y se tomo como base para el resto

de de perfiles. El diseño del codificador del canal está listo entonces para realizar el resto de etapas especificadas en el estándar de Wimax.

3. Con las graficas del BER vs SNR obtenidas, se comprueba la capacidad de detección y corrección de errores del sistema, con lo que se demuestra la utilidad del codificador del canal y sus ventajas comparándolo con un sistema sin codificación. Así mismo al visualizar las constelaciones de las señales a la salida del canal, se puede tener una idea de que tan eficaz es el codificador usado.

4. La generación del código VHDL para poder usar la FPGA fue realizado de manera separada tanto para el codificador como para el decodificador, por lo que se recomienda usar una FPGA de mayores prestaciones para generar las dos partes en conjunto y para poder agregar la etapa de la generación del símbolo OFDM en el diseño.

5. Con el desarrollo de este trabajo se podrá realizar investigaciones futuras en el campo de redes de acceso fijo inalámbrico de banda ancha sin línea de vista, del cual ya existen varios trabajos, los cuales

podrán usarse en conjunto con el diseño realizado en este proyecto, y así poder tener el diseño de la capa física de Wimax.

6. Para futuros trabajos es recomendable realizar ip cores propios para las etapas de FEC, especialmente del codificador de Reed-Solomon y el decodificador de Viterbi, los cuales deben ajustarse a las características del diseño del codificador del canal diseñado.

7. Así mismo el desarrollo de este codificador tiene finalidades académicas, por lo que puede ser usado para entender esta área de las comunicaciones digitales, ya que el diseño realizado ha sido hecho para ser usado en Hardware en contraparte de los modelos de los programas de simulación.

BIBLIOGRAFÍA

1. Instituto de Ingenieros Eléctricos y Electrónicos, Estándar IEEE-802.16-2004, http://standards.ieee.org, 2004
2. Andrews J. Fundamentals of WiMAX, Understanding Broadband Wireless Networking, Prentice Hall, 2007.
3. Bahai A. MultiCarrier Digital Communications Theory and Applications of OFDM. Segunda Edición, Springer, 2004.
4. Sklar B. Digital Communications Fundamentals and Applications – Sklar, Segunda Edición, Prentice Hall, 2001
5. Lopez F. Diseño de Transmisor y Receptor para redes Inalámbricas WMAN, Universidad de Málaga, 2005.

6. Stamplecoskie S, A Study of the Concatenated Reed Solomon – Convolutional Coding Performance used in WiMAX, Defence Research and Development Canada, 2006.
7. Roca A. Implementation of a WiMAX Simulator in Matlab, Eingereicht an der Technischen Universität Wien, 2006.
8. Langton C: Coding and Decoding with Convolutional Codes, www.complextoreal.com, 1999
9. Xilinx Inc, Xilinx Reference Guide, www.xilinx.com, 2009
10. Xilinx Inc, Xilinx Solutions for WiMAX/WiBro System Design, www.xilinx.com, 2005.
11. Azizul M, Performance Evaluation of WiMAX/IEEE 802.16 OFDM Physical, Helsinki University of Technology, 2007.
12. Serra M. Prototipado Rápido de la Capa Física de OFDM: Hiperlan2, Universidad Autónoma de Barcelona, 2005.
13. Sghaier A, Areibi S, Dony R. Implementation Approaches Trade-offs for WiMax OFDM Functions on Reconfigurable, University of Guelph, 2008.
14. Rodriguez A, Maroto O. Decodificador de Viterbi, http://www.contrib.andrew.cmu.edu/~albertor/Academics/Projects/Viterbi.pdf, 2003

15. Sandoval S, Codificador y Decodificador Digital Reed-Solomon Programados para Hardware Configurable, http://dialnet.unirioja.es/servlet/articulo?codigo=2343061, 2007.
16. Pless V, Huffman W, Fundamentals of Error-correcting Codes, Cambridge University Press, UK, 2003.
17. Wicker S. Error Control Systems, Prentice Hall, 1994.
18. Chien R.T. Cyclic Decoding Procedure for the Bose-Chaudhuri-Hocquenghem Codes," IEEE Transactions on Information Theory, Vol. IT-10, 1964.
19. Houghton A. Error Codig for Engineers, Springer, 2001,
20. Viterbi, A. J., "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes," IEEE Journal on Selected Areas in Communications, vol. 16, No. 2", 1998.

ANEXOS

```
%Datos de Configuración iniciales
%datos
input_data = sscanf(['45 29 C4 79 AD 0F 55 28 AD 87 B5 76 1A 9C 80
50 45 1B 9F D9 2A 88 95 EB AE B5 2E 03 4F 09 14 69 58 0A 5D FF'],
'%x');
randomized_data_1 = sscanf(['D4 BA A1 12 F2 74 96 30 27 D4 88 9C 96
E3 A9 52 B3 15 AB FD 92 53 07 32 C0 62 48 F0 19 22 E0 91 62 1A
C1'],'%x');
randomized_data = sscanf(['D4 BA A1 12 F2 74 96 30 27 D4 88 9C 96 E3
A9 52 B3 15 AB FD 92 53 07 32 C0 62 48 F0 19 22 E0 91 62 1A C1
00'],'%x');
rs_encoded_data = sscanf(['49 31 40 BF D4 BA A1 12 F2 74 96 30 27 D4
88 9C 96 E3 A9 52 B3 15 AB FD 92 53 07 32 C0 62 48 F0 19 22 E0 91 62
1A C1 00'],'%x');
conv_encoded_data = sscanf(['3A 5E E7 AE 49 9E 6F 1C 6F C1 28 BC BD
AB 57 CD BC CD E3 A7 92 CA 92 C2 4D BC 8D 78 32 FB BF DF 23 ED 8A 94
16 27 A5 65 CF 7D 16 7A 45 B8 09 CC'],'%x');
interleaved_data = sscanf(['77 FA 4F 17 4E 3E E6 70 E8 CD 3F 76 90
C4 2C DB F9 B7 FB 43 6C F1 9A BD ED 0A 1C D8 1B EC 9B 30 15 BA DA 31
F5 50 49 7D 56 ED B4 88 CC 72 FC 5C'],'%x');


ord_reed_sol_id1=[25:32 1:24];
ord_reed_sol_deco_id1=[9:32 1:8];

ord_reed_sol2=[37:40 1:36 41:52];
ord_reed_sol=[37:40 1:36];
ord_reed_sol_deco=[5:40 1:4];

ord_reed_sol_id3=[49:64 1:48];
ord_reed_sol_deco_id3=[17:64 1:16];

ord_reed_sol_id4=[73:80 1:72];
ord_reed_sol_deco_id4=[9:80 1:8];

ord_reed_sol_id5=[97:108 1:107];
ord_reed_sol_deco_id5=[13:108 1:12];

ord_reed_sol_id6=[109:120 1:108];
ord_reed_sol_deco_id6=[13:120 1:12];

%interleaver
%ncbps=192(BPSK),384(QPSK),768(16-QAM),1152(64-QAM)

Ncbps = 384;
Ncpc = 2;
```

```
k = 0 : Ncbps - 1;
mk = (Ncbps/12) * mod(k,12) + floor(k/12);

s = ceil(Ncpc/2);
jk = s * floor(mk/s) + mod(s, mk + Ncbps - floor(12 * mk/Ncbps));

[s1,int_idx]=sort(jk);

s = ceil(Ncpc/2);

%deinterleaver:
[s2,dint_idx] = sort(int_idx);
```