HSDPA

High Speed Download Packet Acces

Proyecto de Fin de Carrera
Facultad de Ingeniería
Universidad de la República

Patricia Rodríguez
Fernando Caamaño

Tutor: José Acuña

Tabla de Contenidos

Glosario

3GPP	3rd Generation Partnership Project
AICH	Acquisition Indication Channel
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BS	Base Station
CCPCH	Common Control Physical Channel
CRC	Cyclic Redundancy Channel
CQI	Channel Quality Information
DPCH	Dedicated Physical Channel
DL	Downlink
FDD	Frequency Division Duplex
FEC	Forward Error Correction
HS-DPCCH	High Speed Dedicated Physical Control Channel
HS-DSCH	High Speed Downlink Shared Channel
HS-PDSCH	High Speed Physical Downlink Shared Channel
HS-SCCH	High Speed Shared Control Channel
HS-DSCH	High Speed – Downlink Shared Channel
HARQ	Hybrid Automatic Repeat request
ISI	Inter-Symbol Interference
MRC	Maximum Ratio Combining
Nodo B	Estación Base
PCCC	Parallel Concatenated Convolutional Code
PhCH	Physical Channel
SNR	Signal to Noise Ratio

SUI	Stanford University Interim
TTI	Transmission Timing Interval
UL	Uplink
UE	User Equipment

1 Introducción

El mundo de las telecomunicaciones ha evolucionado ampliamente en los últimos años, fenómenos como la globalización, el desarrollo de nuevas tecnologías y las crecientes necesidades de comunicación derivadas a su vez del vertiginoso aumento de las diferentes actividades e interrelaciones sociales, se han retroalimentado.

En este contexto, es de destacar la enorme penetración que ha tenido la tecnología celular en el mundo en un período no demasiado prolongado. La utilización cada vez mayor del celular es por razones que van desde las meramente económicas para el usuario a el acceso a más y mejores servicios.

A su vez, la otra área que ha crecido notablemente en los últimos años es Internet. Su utilización y difusión a nivel de toda la población es creciente en forma exponencial. Internet se ha integrado en el avance de la telefonía móvil, potenciándose mutuamente ya que los terminales actuales permiten su acceso.

Como todo fenómeno, este crecimiento del mercado tiene múltiples causales y efectos. Se encuentran asociados: el vertiginoso avance de la tecnología, las oportunidades comerciales vinculadas con la creciente movilidad personal, y la disminución del precio de terminales y de tarifas.

Un crecimiento tan espectacular y rápido, lleva aparejado el desarrollo e implantación de diferentes tecnologías, muchas veces coexistiendo en una misma plaza. A partir de tal realidad resulta complicado y normalmente costoso, dotar de un servicio de movilidad universal a los usuarios que permita cubrir sus desplazamientos y que pueda estar disponible en todo momento y lugar. Surge así la ineludible necesidad de crear un estándar.

Con este objetivo se forma el grupo 3GPP, 3rd Generation Partnership Project. 3GPP es un acuerdo de colaboración en la tecnología de telefonía celular establecido en diciembre de 1998. El objetivo del 3GPP es hacer especificaciones técnicas para los sistemas móviles 3G que permitan su aplicabilidad a nivel global.

Los sistemas de comunicaciones móviles de tercera generación, surgen tras el enorme desarrollo y crecimiento de los sistemas de segunda generación, con el objetivo de ofrecer comunicación de cualquier tipo de información para todos los ciudadanos del mundo, desde y hacia cualquier lugar y en cualquier momento.

EL 3GPP está basado en la evolución de los sistemas GSM, en la actualidad comúnmente conocidos como sistemas UMTS: Servicios Universales de Telecomunicaciones Móviles. Los sistemas UMTS, son miembros de la familia IMT-2000 del sistema de comunicaciones móviles de *“tercera generación*” de la UIT (Unión Internacional de Telecomunicaciones).

UMTS busca extender las tecnologías móviles, inalámbricas y satelitales para proporcionar mayor capacidad y una gama de servicios mucho más extensa. Aventaja a los sistemas móviles de segunda generación (2G) por su potencial para soportar mayores velocidades de transmisión de datos. Esta capacidad sumada al soporte inherente del Protocolo de Internet (IP), se combinan permitiendo prestar servicios multimedia interactivos y nuevas aplicaciones de banda ancha, tales como servicios de video telefonía y video conferencia.

En 1999, se aprobaron las primeras especificaciones del 3GPP, conocidas como '*releasse 99*' o *W-CDMA 99*, que incluían los servicios básicos compatibles entre las redes GSM y las futuras en ese momento de UMTS. Estas tecnologías 3G, alcanzan picos de 2Mbps. A los efectos de cubrir la creciente demanda de servicios multimedia de alta velocidad, 3GPP, crea luego la *'release 5'*, en la cual se incluye la nueva tecnología HSDPA, High-Speed Downlink Packet Access, o 3.5G. HSDPA se introdujo en el 3GPP para proveer un canal de datos de bajada de alta velocidad. Con esta innovación la tasa de datos en el link de bajada podría alcanzar hasta 14 Mbps.

HSDPA tiene mejoras impresionantes respecto a la versión 99 de WCDMA para la bajada. El efecto de la misma resulta en una mejor experiencia para el usuario final para aplicaciones de descarga de datos, con tiempos de conexión y respuesta mucho menores.

1.1 Motivación del proyecto

Los servicios 3G se están utilizando cada vez más por lo tanto nos pareció sumamente interesante explorar esta área. En el momento en que elegimos este proyecto, en Uruguay aún no se hacia mención pública de los servicios 3G. A poco tiempo de comenzar el proyecto (marzo del 2007) dos compañías locales ofrecían las prestaciones de las comunicaciones 3.5G. Evidentemente los comentarios que se han hecho en la introducción al presente trabajo sobre lo vertiginoso de los avances, no son mera literatura sino que la plaza local es un ejemplo más de cómo estas tecnologías se desarrollan rápidamente.

Más precisamente nos referimos a la tecnología HSDPA, cuya velocidad máxima teórica (14 Mbps) es muy alta si la comparamos con las velocidades que se venían alcanzando en teléfonos móviles. Sin embargo no han salido al mercado receptores capaces de trabajar a esta tasa de datos, lo que genera una incógnita al respecto.

Por otro lado el proyecto implicaba el estudio y la profundización acerca de una tecnología nueva, dentro de un área de interés como lo es la telefonía celular.

Lo novedoso de la tecnología y la cantidad de servicios multimedia asociados llevan a la conveniencia de intentar desarrollar simulaciones que permitan evaluar el comportamiento del sistema. Adicionalmente se busca aportar conocimiento y experiencia sobre el tema y fundamentalmente sentar bases sobre las cuales nuevas investigaciones puedan avanzar en el futuro.

Entendemos que éste es un proyecto que debe formar parte de un proceso de desarrollo e investigación y que mediante sucesivas etapas y aproximaciones permita alcanzar los mejores rendimientos de la tecnología analizada.

1.2 Objetivos del proyecto

Se pretende analizar el desempeño de diferentes implementaciones en sistemas de transmisión inalámbrica basadas en el sistema HSDPA. Se toma como base el Standard del 3GPP, release 5. El punto es estudiar las diferentes implementaciones a nivel de capa física y trabajando a la velocidad máxima teórica de transmisión, 14,4 Mbps, de manera de ver las mejoras de cada una de las opciones presentadas.

Las diferentes opciones serán comparadas por medio de simulaciones, utilizando la herramienta Simulink de Matlab. Las implementaciones serán extraídas de la literatura y seleccionadas de acuerdo a su conveniencia en cuanto a la mejora de la performance y a su practicidad.

1.3 HSDPA

La tecnología HSDPA es la optimización de la tecnología espectral UMTS/WCDMA (3G).

Las metas de HSDPA son las siguientes:

- Incrementar la capacidad
- Reducir retardos
- Obtener mayores picos en la tasa de datos
- Planificación eficiente de usuarios

Los servicios de datos como el acceso a Internet móvil, requieren redes asimétricas para utilizar de la mejor manera el espectro disponible en un entorno multiusuario.

HSDPA provee canales de bajada de alta velocidad que pueden ser compartidos eficientemente por múltiples usuarios. Logra teóricamente tasas de datos de 14,4Mbps y una mejora en el control de errores, además de otras técnicas, como H-ARQ[1], modulación adaptiva o una planificación eficiente de usuarios, lo que resulta en una mejora del rendimiento de la red.

La figura muestra la evolución de las tecnologías 2G a 3.5G:

[1] Hybrid Automatic Repeat request, presentado en 3.1

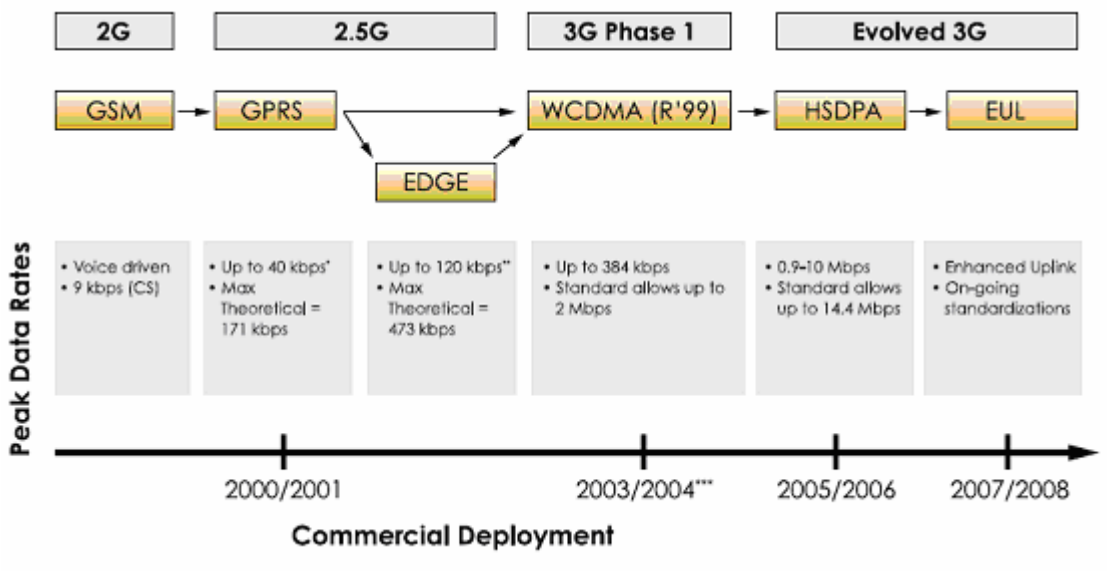


Figura 1.1: Evolución de la tecnología celular [1]

Para mejorar el sistema w-cdma 99 y pasar a HSDPA, se introducen los siguientes cambios en la interfase de radio, que afectan principalmente a la capa física:

- Trama de radio más corta
- Modulación 16QAM además de QPSK
- Nuevo canal de control
- Modulación adaptiva
- H-ARQ: Hybrid Automatic Repeat Request
- Planificación de acceso al medio (MAC) en la estación base (Nodo B)

Estas mejoras son explicadas en el punto 3

Un beneficio adicional importante de HSDPA que debe mencionarse, es su compatibilidad hacia atrás con la release 99. Esto provoca que su despliegue sea muy suave y gradual basándose en las necesidades de operadores y clientes. Al tener la posibilidad de ser compatible hacia atrás, las aplicaciones de voz y datos desarrolladas para W-CDMA pueden seguir corriendo en las redes actualizadas, y el mismo canal de radio puede soportar servicios W-CDMA y HSDPA simultáneamente.

1.4 Simulink

Simulink es una herramienta para el modelaje, análisis y simulación de una amplia variedad de sistemas físicos y matemáticos. Este software Incluído en Matlab, es la herramienta que se utilizará para llevar a cabo las simulaciones que se incluyen en el proyecto

Utiliza diagramas de bloques para representar sistemas dinámicos. El software permite mediante una interfaz gráfica con el usuario que se puedan arrastrar

los componentes desde una librería de bloques existentes y luego interconectarlos

Simulink utiliza un ambiente gráfico interactivo lo que hace sencilla la creación de los modelos de sistemas. Provee un grupo '*customizable*' de librerías de bloques que permiten diseñar, simular, implementar y testear una variedad de sistemas incluyendo comunicaciones, control, procesamiento de señales y procesamiento de video e imágenes.

2 Conceptos teóricos

A los efectos de la disponibilidad integral de la información relevante en el texto del Proyecto, se describirán algunos conceptos teóricos que se utilizan en el proyecto, sin que esto implique pretender sustituir desarrollos teóricos específicos mencionados en la bibliografía.

2.1 Canal multicamino

Refiere a un modelo de propagación aplicable a señales transmitidas en entornos urbanos. Este bloque se agrega para modelar el hecho que la señal transmitida, debido a la reflexión, dispersión o difracción de la misma al atravesar un entorno con obstáculos, pueda tomar diferentes caminos para llegar desde el transmisor hasta el receptor.

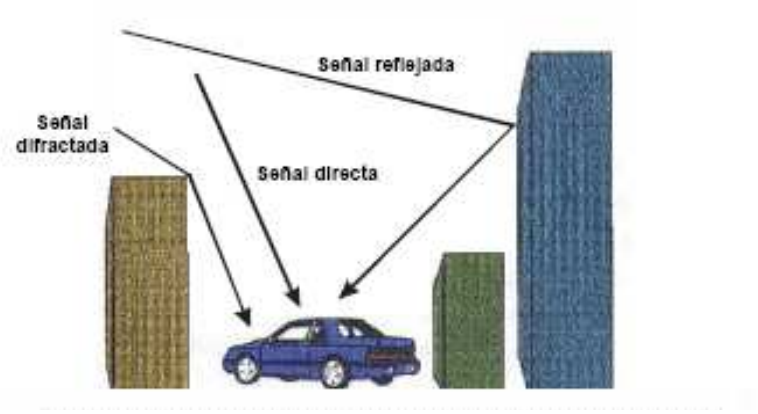


Figura 2.1: Canal multicamino

En este caso, llegan al receptor múltiples copias de la señal con diferente fase y amplitud. Esta circunstancia puede provocar una interferencia constructiva o destructiva según el recorrido de cada una de las copias. Adicionalmente puede generar ISI (interferencia ínter simbólica) en el caso que un pulso llegue retrasado al receptor al mismo tiempo que llega otro pulso, como muestra la figura 2.2.

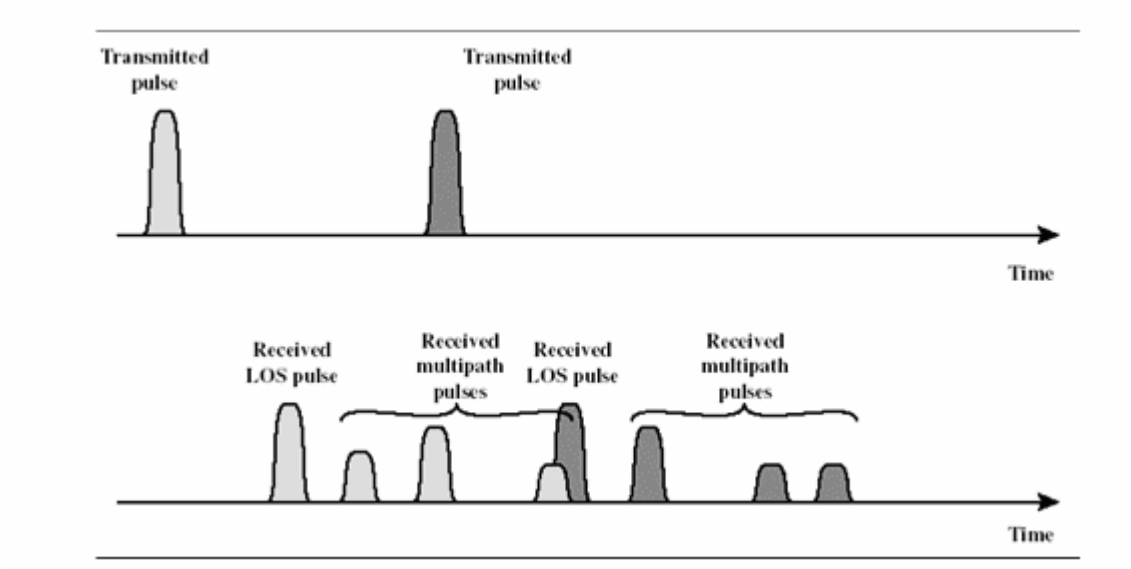


Figura 2.2: Efectos de canal multicamino

El siguiente esquema es un ejemplo de modelo del canal multicaminos: la señal se transmite por diferentes caminos, cada uno con un retardo tao y una atenuación diferentes. En la figura además se agrega el ruido blanco y la interferencia por acceso múltiple [2].

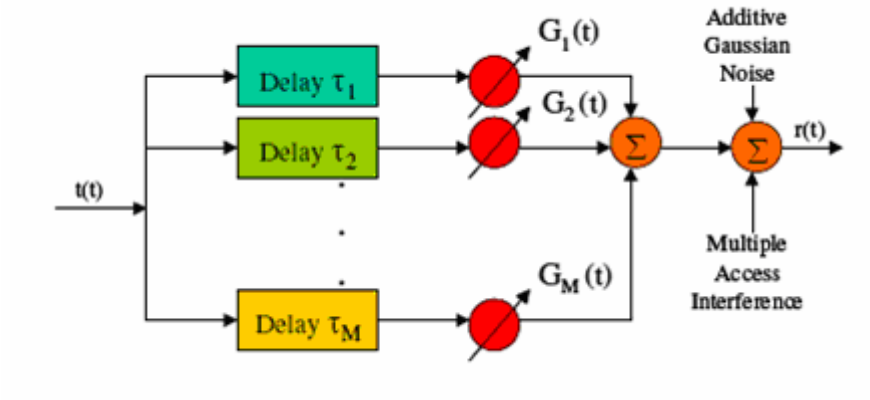


Figura 2.3: Modelo de canal multicamino

Las características de estos desvanecimientos varían con el entorno de propagación y su repercusión en la calidad de las señales depende, adicionalmente, de la velocidad de la estación móvil respecto a la estación base.

Se hace evidente la necesidad de contar con un modelo de canal en el momento de diseñar un sistema de comunicación inalámbrica. En el punto 4 se presentan el modelo de canal inalámbrico que será de aplicación en el proyecto.

2.2 AWGN

En comunicaciones, el modelo de canal de ruido aditivo blanco gaussiano (AWGN) representa la adición lineal de ruido blanco, con una densidad espectral constante y una distribución de amplitud Gaussiana. El modelo no tiene en cuenta fenómenos tales como desvanecimiento, selectividad de frecuencias, interferencia, no linealidades o dispersión. Sin embargo, produce modelos matemáticos simples y tratables que resultan sumamente útiles para ganar entendimiento sobre el comportamiento subyacente de un sistema antes que esos otros fenómenos sean considerados.

El ruido blanco viene de varias fuentes naturales, tales como las vibraciones térmicas de los átomos en las antenas (referido como ruido térmico o ruido de Johnson-Nyquist), radiación de cuerpo negro desde la tierra y otros objetos cálidos y desde fuentes celestiales tales como el Sol.

El canal AWGN es un modelo adecuado para enlaces de varios satélites y de comunicación de espacio profundo. No es un buen modelo para la mayoría de los enlaces terrestres debido a tópicos tales como multicamino, bloqueo terrestre, interferencia, etc. Sin embargo, para el modelado de caminos terrestres, AWGN se utiliza comúnmente para simular ruido de fondo del canal bajo estudio.

2.3 Codificación Turbo

Los códigos Turbo son un método de corrección de errores tipo FEC, Forward Error Correction. Es uno de los métodos de corrección de errores que se sitúa más cerca de alcanzar el límite de Shannon: el límite teórico de máxima transferencia de información sobre un canal ruidoso. Desde 1993, fecha en que fueron anunciados, hasta la actualidad se han utilizado en varias aplicaciones, principalmente en comunicaciones inalámbricas.

En [3] y [4] se presentan estudios sobre estos codificadores. En este documento se incluye un pequeño resumen de estos trabajos a modo de breve introducción sobre la codificación Turbo.

Un codificador de FEC agrega redundancia a los datos como información de paridad. El codificador de FEC toma k bits a la vez y produce una salida (o palabra de código) de n bits, donde $n>k$. El cociente k/n se denomina la tasa de código. En el receptor, un decodificador de FEC utiliza esa redundancia para poder corregir los errores de canal. Con un código FEC se logra tolerar más errores de canal, por lo que los sistemas codificados de esta manera pueden funcionar con menos potencia en la transmisión, transmitir a largas distancias y tolerar más interferencia.

La performance de los FEC crece con el tamaño de k y se aproxima al límite de Shannon para valores de k muy grandes, pero también crece la complejidad de la decodificación en estos casos. Esto llevó a pensar en la idea de construir un código complejo y largo pero que estuviera compuesto por

componentes más cortos, de forma que éstos fueran más fáciles de decodificar. De aquí surge la concatenación como método principal para estos codificadores

Un turbo código está formado por la concatenación paralela de dos códigos separados por un intercalador. Los dos codificadores son idénticos y de tipo convolucional, recursivos y sistemáticos, lo que significa que una de las salidas del codificador convolucional corresponde directamente a la entrada a dicho codificador (salida sistemática) además de los bits de paridad, y recursivos porque internamente sus registros tienen realimentación.

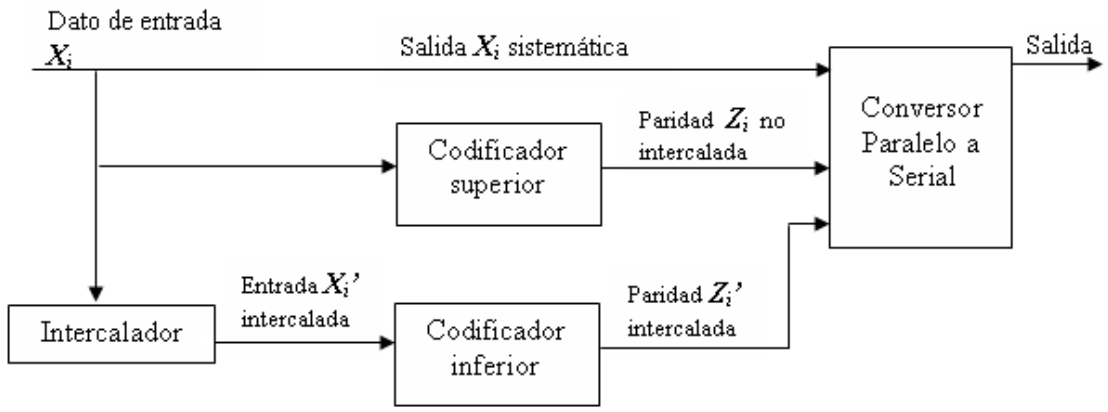


Figura 2.4: Estructura genérica para un turbo código

La secuencia de datos de entrada (Xi) y las salidas de la paridad de los dos codificadores paralelos (Zi y Zi') pasan a ser concatenadas para formar la salida del turbo código.

En el anexo se presenta una introducción a la codificación convolucional.

2.4 RAKE

Estos receptores están diseñados para contrarrestar los efectos del multipath fading[2]

El receptor RAKE permite que los diferentes componentes de una señal individual producidos por retardos de multicamino sean detectados por separado y coherentemente combinados. Esto no sólo tiende a prevenir pérdidas graduales, si no que también proporciona un efecto de diversidad de canales resultando en enlaces más robustos.

El receptor RAKE intenta recoger las versiones retrasadas de la señal original por medio de un receptor de correlación separado para cada una de las señales de multicamino, cada uno será una rama del RAKE. Esto se puede

[2] Efectos del canal multicamino

hacer debido a que las componentes de multicamino prácticamente no son correlacionadas si su retraso de propagación relativo excede el tiempo de chip.

Se utiliza la autocorrelación de una señal de referencia que es transmitida junto con la señal original. Esta señal se genera en el transmisor, y luego en el receptor de genera nuevamente para compararla con la señal recibida.

En la figura 3.4 se puede ver el modelo de un receptor RAKE de M ramas. El mismo utiliza múltiple receptores de correlación para descubrir por separado los M componentes de multicamino más fuertes. Las salidas de cada *correlator* son ponderadas (alfa) según la estimación del canal para proporcionar mejor la estimación de la señal transmitida. La desmodulación y decisiones de bit están entonces basadas en las salidas ponderadas de los M receptores de correlación

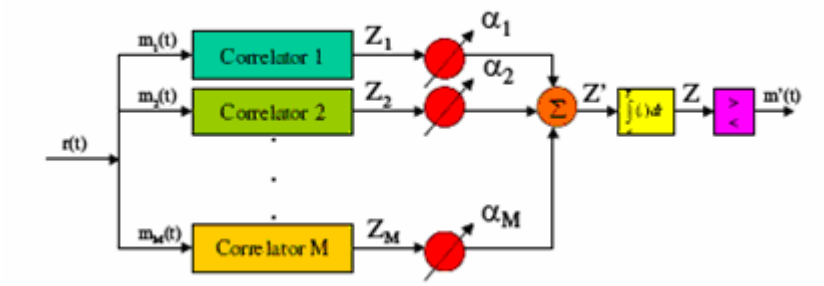


Figura 2.5: Recepor RAKE de M ramas

El número ideal de ramas del RAKE es un compromiso entre complejidad y performance. Depende principalmente de la tasa de chips y del grado de dispersión del canal. A mayor tasa de chips, mayor cantidad de caminos se podrán distinguir, pero crecería el ancho de banda utilizado. También para recoger más señales provenientes de diferentes caminos sería mejor utilizar más ramas en el RAKE, pero trabajar con muchas ramas puede generar pérdidas al combinar los datos, además de problemas prácticos de implementación. En la mayoría de los ejemplos que hemos estudiados no se utilizan más de cuatro ramas en un receptor RAKE, excepto en el caso de una estación base donde se utilizan ocho ramas para el receptor.

2.5 EQ

Los ecualizadores son filtros adaptativos, esto significa que es un sistema que intenta ajustar sus parámetros con el objetivo de alcanzar alguna meta bien definida que depende del estado del sistema así como de los alrededores. Dependiendo del tiempo requerido para alcanzar la meta final del proceso de adaptación, que se denomina tiempo de convergencia, y de la complejidad y recursos disponibles para llevar a cabo la adaptación, existen una variedad de algoritmos y estructuras de filtros.

El proceso de seleccionar los coeficientes del filtro para lograr la mejor correspondencia entre la señal deseada y la salida del filtro se logra generalmente optimizando una función de performance apropiadamente definida. La función de performance puede ser definida en un esquema estadístico o determinístico.

En el enfoque estadístico, la función de performance más utilizada es el valor medio cuadrático de la señal de error, o sea, la diferencia entre la señal deseada y la salida del filtro. Para entradas estacionarias y señales deseadas, minimizar el error medio cuadrático resulta en el filtro de Wiener[3], óptimo en el sentido medio-cuadrático.

En el enfoque determinístico, la elección usual de función de performance es una suma ponderada de la señal de error cuadrada. Minimizar esta función resulta en un filtro que es óptimo para el conjunto dado de datos. Sin embargo, bajo algunas suposiciones en ciertas propiedades estadísticas de los datos, la solución determinística se acercará a la solución estadística, o sea, el filtro de Wiener, para longitudes grandes de datos.

- **Método de los mínimos cuadrados**

Los algoritmos de filtrado adaptativo cuyas derivaciones son basados en la teoría de filtros de Wiener tienen su origen en una formulación estadística del problema. En contraste con esto, el método de los mínimos cuadrados enfoca el problema de la optimización del filtro desde un punto de vista determinístico. En el método de mínimos cuadrados, el índice de performance es la suma de los errores cuadrados ponderados para datos dados, o sea una cantidad determinística. Una consecuencia de ese enfoque determinístico es que los algoritmos basados en mínimos cuadrados, en general, convergen mucho más rápido que los algoritmos basados en LMS. También son insensibles a la densidad espectral de potencia de la señal de entrada. El precio que se paga por lograr esta mejora en la convergencia es mayor complejidad computacional y estabilidad numérica más pobre.

En el contexto de los filtros adaptativos, se prefieren formulaciones recursivas del método de mínimos cuadrados que actualicen los coeficientes del filtro luego de la llegada de cada muestra de la entrada.

Hay tres grandes clases de algoritmos de filtrado adaptativo de mínimos cuadrados recursivos (RLS – recursive least-squares):

- El algoritmo RLS estándar
- El algoritmo de descomposición QR basado en RLS (QRD-RLS)
- Algoritmos RLS rápidos

---

[3] Filtro propuesto por Norbert Wiener publicado en 1949. Su propósito es reducir la cantidad de ruido presente en una señal al compararla con una estimación de la señal deseada sin ruido.

Se incluyen breves comentarios sobre los mismos:

El algoritmo estándar RLS

La derivación de este algoritmo involucra el uso de un resultado bien conocido del algebra lineal conocido como el lema de inversión de matriz. Consecuentemente, la implementación del algoritmo estándar RLS involucra manipulaciones matriciales que resultan en una complejidad computacional proporcional al cuadrado del largo del filtro.

El algoritmo de descomposición QR basado en RLS (QRD-RLS)

Esta formulación del algoritmo RLS también involucra manipulaciones matriciales que llevan a una complejidad computacional que crece con el cuadrado del largo del filtro. Sin embargo, las operaciones involucradas aquí son tales que pueden ser puestas en estructuras regulares conocidas como arrays sistólicos[4]. Otra característica importante del algoritmo QRD-RLS es su robustez a errores numéricos comparado con otros tipos de algoritmos RLS.

Algoritmos RLS rápidos

En el caso de filtros transversales, las entradas a los taps son muestras sucesivas de la señal de entrada x(n). Los algoritmos RLS rápidos usan esta propiedad de la entrada del filtro y solucionan el problema de los mínimos cuadrados con una complejidad computacional que es proporcional al largo del filtro, por lo tanto el nombre RLS rápido. Dos tipos de algoritmos RLS rápidos pueden ser reconocidos:

1. Algoritmos RLS lattice: Estos algoritmos lattice involucran el uso de ecuaciones que actualizan el orden asi como el tiempo. Una consecuencia de esta característica es que resulta en estructuras modulares que son apropiados para implementaciones de hardware. Otra característica deseable de estos algoritmos es que ciertas variantes de ellos son muy robustos contra errores numéricos que surgen del uso de largos de palabra finita al computar.

2. Algoritmo RLS rápido transveral: En términos de cantidad de operaciones por iteración, el algoritmo RLS rápido transversal es menos complejo que los algoritmos RLS lattice. Sin embargo, sufre de problemas de inestabilidad numérica que requieren atención para prevenir un comportamiento indeseable en la práctica

[4] Arreglo de procesadores en un array donde los datos fluyen sincrónicamente a través del array entre los vecinos, usualmente con diferentes datos fluyendo en diferentes direcciones. Cada procesador en cada paso toma datos de uno o más vecinos, lo procesa y, en el siguiente paso, las salidas resultan en dirección opuesta.

3 Descripción de HSDPA

3.1 Introducción

HSDPA apunta a incrementar la capacidad, reducir retardos y a obtener mayores picos en la tasa de datos.

Una de las innovaciones para lograr estos objetivos es la introducción de nuevos canales de transporte y nuevos canales físicos. Los nuevos canales físicos son:

- HS-PDSCH, High Speed Physical Downlink Shared Channel, que se encarga de transportar los datos
- HS-SCCH, High Speed Shared Control Channel, que lleva la identidad del usuario y parámetros de los HS-PSDSCH asociados.
- HS-DPCCH, High Speed Dedicated Physical Control Channel, un canal de subida que lleva los ACK e información de CQI[5]

Y se agrega el nuevo canal de transporte, HS-DSCH (High Speed – Downlink Shared Channel), un canal de bajada que es compartido por varios UE.

Las técnicas más destacadas que agrega HSDPA son:

- Fast Link Adaptation (Adaptación rápida de enlace)
- HARQ (Hybrid Automatic Repeat reQuest)
- Fast scheduling

Las técnicas de Fast Link Adaptation permiten el uso de una modulación de mayor orden (16QAM) en condiciones favorables del canal y se revierte a la modulación robusta QPSK en condiciones de canal menos favorables. La capa MAC en el Nodo B se encarga de seleccionar el tipo de modulación de acuerdo a: reportes del UE, tamaños de los buffer de espera, la potencia instantánea del DPCH y la calidad de servicio requerida.

Los algoritmos de HARQ piden rápidamente la retransmisión de datos perdidos y combinan la información de la transmisión original con cualquier retransmisión subsiguiente antes de decodificar el mensaje. Así la probabilidad de decodificar el mensaje exitosamente aumenta. Compensa errores realizados por el link adaptation. Como el mecanismo HARQ reside en el Nodo B, las retransmisiones son más rápidas.

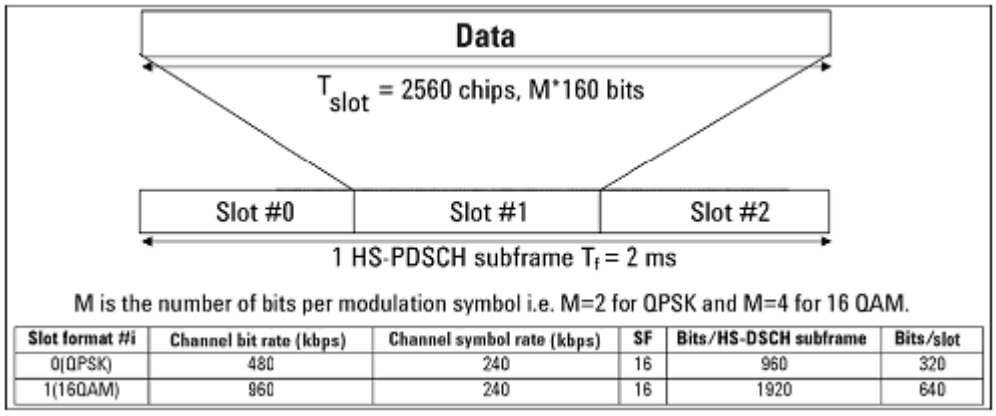
Fast scheduling comparte el HS-DSCH entre los usuarios. Esta técnica, que explota la diversidad multi-usuario, se esfuerza para transmitir a usuarios con condiciones de radio favorables.

[5] Channel Quality Information

3.2 Capa Física HS-DSCH

Con la creación de los nuevos canales, la capa dos puede mapear los canales lógicos existentes en WCDMA en el nuevo canal HS-DSCH. La capa uno se encarga de mapear el canal de transporte (HS-DSCH) en uno o más (hasta quince) canales físicos (HS-PDSCH). Luego crea los canales HS-SCCH y HS-DPCCH para controlar y asistir en la transmisión del HS-DSCH.

En la figura 3.1 se puede observar la estructura del canal físico de bajada, HS-PDSCH. En la norma 25.213, se fija la tasa de chips en 3.84 Mcps. La tasa de bits depende del tipo de modulación que se esté utilizando. La tasa de símbolo por segundos, ksps, es fija y tiene un valor de 240 ksps, lo que resulta en una velocidad de 480 kbps o 960 kbps según se esté trabajando con QPSK o 16QAM respectivamente. El ancho de banda que utiliza es de 5MHz.



M is the number of bits per modulation symbol i.e. M=2 for QPSK and M=4 for 16 QAM.

Slot format #i	Channel bit rate (kbps)	Channel symbol rate (kbps)	SF	Bits/HS-DSCH subframe	Bits/slot
0(QPSK)	480	240	16	960	320
1(16QAM)	960	240	16	1920	640

Figura 3.1: Canal físico de bajada

Una sub-trama es el intervalo de tiempo básico para la transmisión del HS-DSCH y la señalización en la capa física. La longitud de una sub-trama (o longitud de trama HSDPA) o TTI_{HSDPA} corresponde a 3 slots que son 2 ms o 7680 chips.

El canal de control lleva información de codificación del canal, como el tamaño de bloque de transporte (6 bits), información de HARQ (3 bits), versión de constelación y redundancia (3 bits) y el indicador de datos nuevos (1 bit). La tasa de bits del HS-SCCH es de 60 kbps.

Como muestra la figura 3.2, el canal de control HS-SCCH se comienza a transmitir previo a la transmisión de datos en HS-DSCH, de esta forma los parámetros necesarios para la obtención de los datos ya son conocidos en el momento que llegan los datos por lo que se pueden comenzar a procesar inmediatamente.

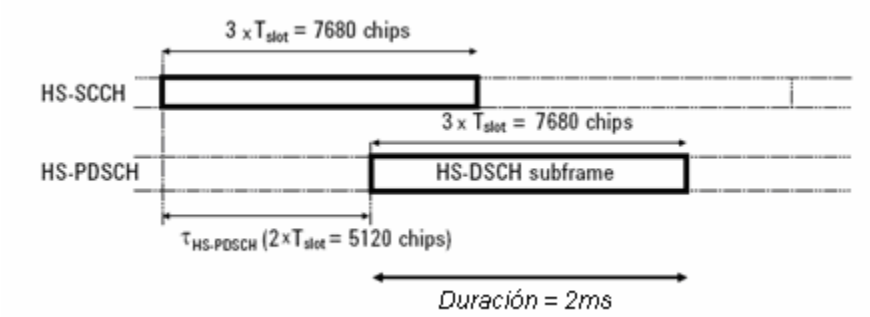


Figura 3.2: Tiempos de los canales físicos de bajada (3gpp TS 25.211 7.8)

La figura siguiente es un ejemplo de codificación de canal. En ella se pueden ver algunos de los bloques de la capa física y la cantidad de bits que maneja cada bloque en este caso. En este ejemplo se usan cuatro canales físicos y un bloque de transporte de 4664 bits.

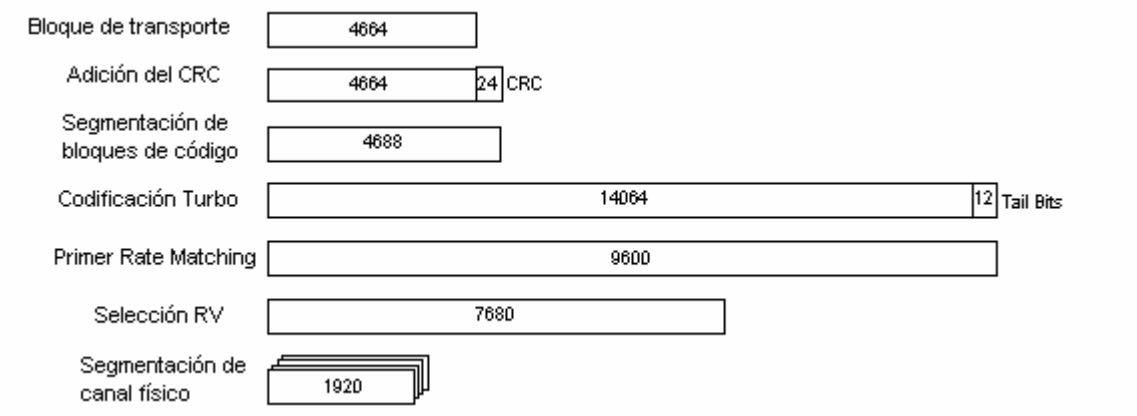


Figura 3.3: Ejemplo de codificador de canal

El siguiente es un esquema del conjunto de bloques de multiplexado y codificación de canal que componen la capa física.

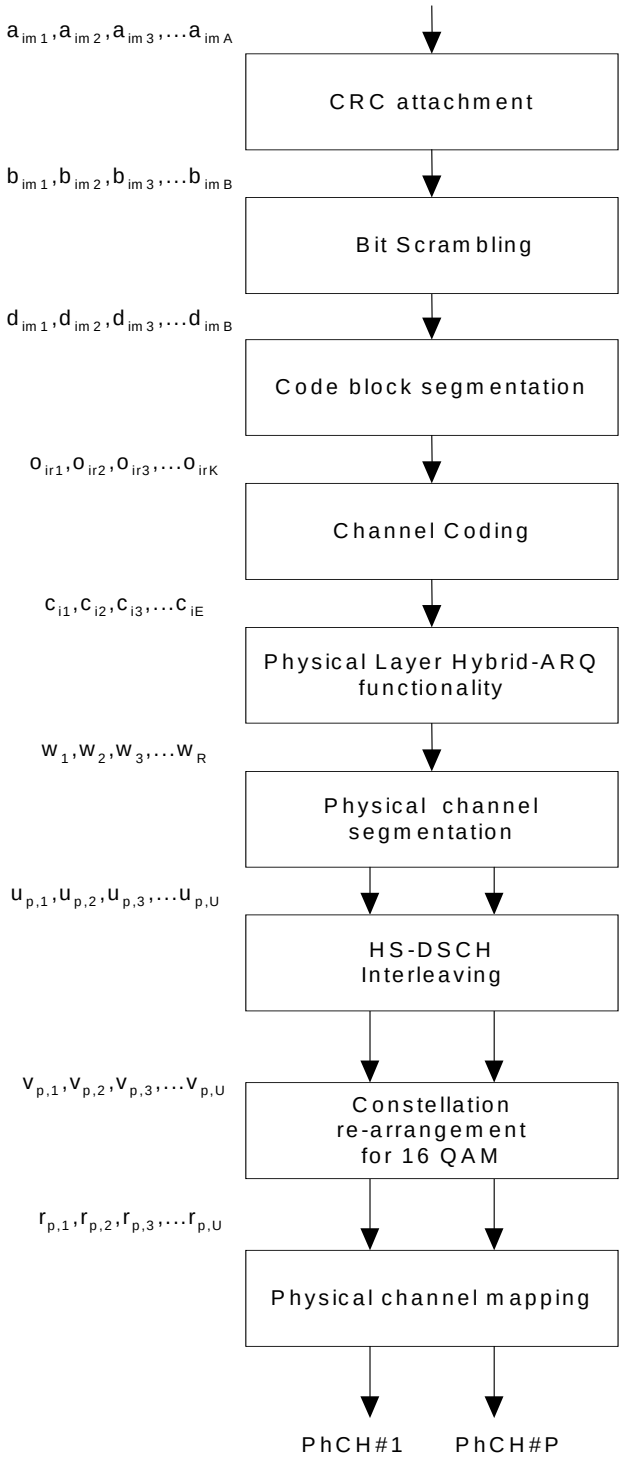


Figura 3.4: Cadena de bloques para HS-DSCH (3GPP TS 25.212 V5.10.0)

Los bits $a_{im1}, a_{im2}, \ldots, a_{imA}$ provienen de las capas superiores. Es la información a ser transmitida por el canal de datos.

A este diagrama se le debe agregar la sección de spreading y modulación, completando así los bloques necesarios para hacer todo el tratamiento de la información proveniente de las capas superiores.

A continuación se presenta información sobre el funcionamiento de cada uno de bloques anteriores, definidos principalmente en las normas [5] y [6] de 3GPP. Estas normas especifican el funcionamiento de la capa física para el canal de datos.

3.2.1 Código de Redundancia Cíclica (CRC)

La primera operación que se aplica a los datos que ingresan a la capa física es agregarle un CRC (código de redundancia cíclica). En [5] se determina que el tamaño del CRC sea de 24 bits.

El polinomio generador utilizado es: $g_{CRC24}(D) = D^{24} + D^{23} + D^6 + D^5 + D + 1$

Para hallar los bits de paridad, se divide:

$$a_{im1}D^{A+23} + a_{im2}D^{A+22} + \ldots + a_{imA}D^{24} + p_{im1}D^{23} + p_{im2}D^{22} + \ldots + p_{im23}D^1 + p_{im24}$$

entre g_{CRC24} y los bits de paridad p_{imj} con j=1,…,24 son tales que esta división da resto igual a cero.

Los bits luego del bloque se denotan por $b_{im1}, b_{im2}, b_{im3}, \ldots, b_{imB_i}$, donde $B = A + L$. La relación entre a_{imk} y b_{imk} es:

$b_{imk} = a_{imk}$ $\quad k = 1, 2, 3, \ldots, A$

$b_{imk} = p_{im(L_i+1-(k-A))}$ $\quad k = A + 1, A + 2, A+ 3, \ldots, A+ L \quad L=24$

O sea que los 24 bits de paridad se agregan a continuación de la entrada.

3.2.2 Bit Scrambling

La funcionalidad de bit scrambling fue introducida para evitar tener secuencias que repiten el mismo símbolo (secuencias largas de ‘1s’ y ‘0s’). Esto podría ocurrir para cierto tipo de contenido y especialmente si no se usa cifrado en capas superiores. En tal caso, el terminal tendría dificultades para estimar el nivel de potencia. La operación es la misma para todos los usuarios y es puramente para asegurar buenas propiedades de la señal para su demodulación.

El bit scrambling se define por la siguiente relación:

$d_{im,k} = (b_{im,k} + y_k) \bmod 2$ $\quad k = 1,2,\ldots,B$

Donde $d_{im,1}, d_{im,2}, d_{im,3}, ..., d_{im,B}$ son los bits a la salida del bloque e y_k resulta de la siguiente operación:

$$y'_\gamma = 0 \qquad -15 < \gamma < 1$$

$$y'_\gamma = 1 \qquad \gamma = 1$$

$$y'_\gamma = \left(\sum_{x=1}^{16} g_x \cdot y'_{\gamma - x} \right) \bmod 2 \qquad 1 < \gamma \leq B ,$$

donde $g = \{g_1, g_2, \ldots, g_{16}\} = \{0,0,0,0,0,0,0,0,0,0,1,0,1,1,0,1\}$,

$y_k = y'_k \qquad k = 1,2,\ldots,B.$

3.2.3 Segmentación

La segmentación de la secuencia de bits en los bloques de transporte se realiza si B > Z, donde Z es el tamaño máximo de los bloques de código y B es la longitud de la secuencia de entrada.

Luego de la segmentación los bloques de código son del mismo tamaño. El número de bloques de código es denotado por *C*. Si el número de bits de entrada a la segmentación, *B*, no es múltiplo de *C*, bits de relleno deben ser agregados al comienzo del primer bloque. Si la codificación turbo es elegida y *B* < 40, bits de relleno son agregados al comienzo del bloque de código. Los bits de relleno son transmitidos y son fijados siempre en 0.

El tamaño máximo de bloque de código definido es: Z = 5114 para codificación turbo.

El algoritmo de segmentación se puede encontrar en el anexo.

3.2.4 Codificación de canal

Los bloques de código son entregados al bloque de codificación de canal. Después de ser codificados los bits son denotados por $y_{ir1}, y_{ir2}, y_{ir3}, \ldots, y_{irY_i}$, donde Y_i es la cantidad de bits codificados. La relación entre o_{irk} (secuencia de entrada) y y_{irk} y entre *K* y Y_i depende del esquema de codificación de canal.

La norma [5] define que se utilizará la codificación Turbo con tasa 1/3 para el canal de datos. De esta forma $Y_i = 3*K + 12$

3.2.4.1 Codificación Turbo

3.2.4.1.1 Codificador Turbo

El esquema del codificador Turbo es un Código Convolucional Paralelo Concatenado (PCCC) con dos codificadores constituyentes de 8 estados y un intercalador interno de código Turbo. La tasa de codificación del codificador Turbo es 1/3. La estructura del codificador Turbo es ilustrada en la figura 3.5.

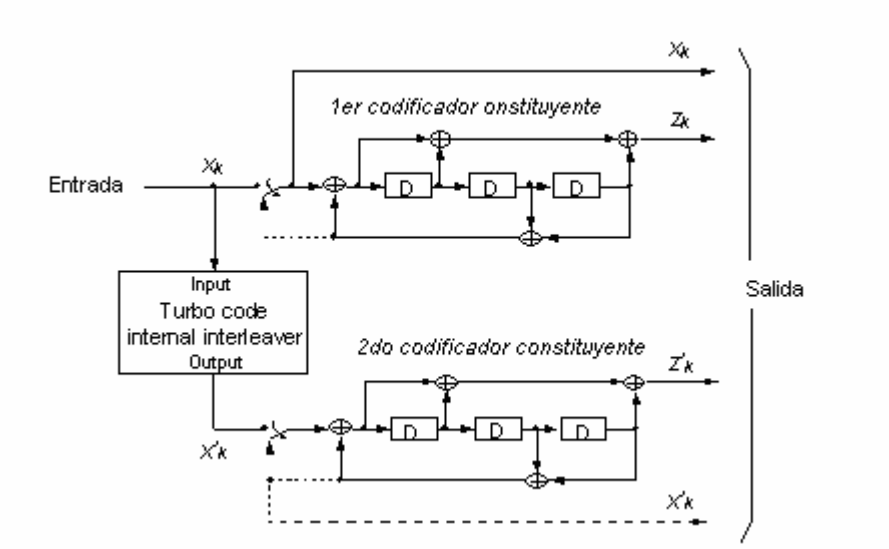


Figura 3.5: Estructura de codificador turbo de tasa 1/3 (las líneas punteadas aplican sólo para terminación

La función de transferencia del código constituyente de 8 estados para PCCC es:

$$G(D) = \left[1, \frac{g_1(D)}{g_0(D)}\right],$$

donde

$$g_0(D) = 1 + D^2 + D^3,$$

$$g_1(D) = 1 + D + D^3.$$

El valor inicial de los shift registers de los codificadores constituyentes de 8 estados deben ser todos ceros cuando se comienza a codificar los bits de entrada.

La salida del codificador Turbo es

$x_1, z_1, z'_1, x_2, z_2, z'_2, \ldots, x_K, z_K, z'_K,$

donde $x_1, x_2, \ldots, x_K$ son los bits de entrada al codificador Turbo, K es la cantidad de bits, y $z_1, z_2, \ldots, z_K$ y $z'_1, z'_2, \ldots, z'_K$ son los bits de salida del primer y segundo codificador de 8 estados, respectivamente. Los bits de salida del primer codificador son denominados bits de paridad uno y los bits de salida del segundo decodificador serán bits de paridad dos.

Los bits de salida del intercalador interno de código Turbo son denotados por $x'_1, x'_2, \ldots, x'_K$, y esos bits serán entrada del segundo codificador de 8 estados.

3.2.4.1.2 Terminación Trellis para codificador Turbo

La terminación Trellis es realizada tomando los bits de cola de la retroalimentación de los shift register después que todos los bits de

información son codificados. Los bits de cola son rellenados después de la codificación de los bits de información. Los primeros tres bits de cola deben ser usados para terminar el primer codificador (llave superior en la figura 4 en la posición inferior) mientras el segundo codificador es deshabilitado. Los últimos tres bits de cola deberían ser usados para terminar el segundo codificador (llave inferior en la figura 4 en la posición inferior) mientras el primero es deshabilitado.

Los bits transmitidos para terminación trellis deberían ser entonces:

x_{K+1}, z_{K+1}, x_{K+2}, z_{K+2}, x_{K+3}, z_{K+3}, x'_{K+1}, z'_{K+1}, x'_{K+2}, z'_{K+2}, x'_{K+3}, z'_{K+3}.

3.2.4.1.3 Intercalador interno de código Turbo

El intercalador interno de código Turbo consiste de bits de entrada a una matriz rectangular con relleno, permutaciones intra e inter fila de la matriz rectangular y bits de salida de la matriz rectangular con borrado.

En el Anexo I se presentan los detalles de cada uno de estos pasos para obtener el intercalado correcto según la especificación.

3.2.5 Funcionalidad HARQ – Rate Matching

La funcionalidad HARQ de la capa física es una extensión del rate matching (adaptación de tasas) de la release 99, que se ocupa de igualar la cantidad de bits a la salida del codificador turbo a la cantidad total de bits de los canales físicos HS-DSCH. Esta funcionalidad es controlada por el parámetro RV[6] (Versión de Redundancia), o sea el conjunto exacto de bits a la salida de la funcionalidad HARQ depende de la cantidad de bits de entrada, la cantidad de bits de salida y el parámetro RV.

HARQ puede ser operado de dos formas distintas, con retransmisiones idénticas y no-idénticas. En el caso de las retransmisiones idénticas, la funcionalidad de rate matching es idéntica entre transmisiones y siempre permanecen los mismos bits (o sea se repiten o se borran los mismos bits) para ser enviados luego del rate matching. Sin importar el número de retransmisiones, la operación de rate matching permanece sin cambios para cada transmisión del mismo paquete. Las retransmisiones no-idénticas, también conocidas como “redundancia incremental”, usan un rate matching distinto entre retransmisiones. La cantidad relativa de bits de paridad a bits sistemáticos varía entre retransmisiones.

[6] El parámetro RV o versión de redundancia es el que indica que tipo de redundancia se utilizará, ya que, en caso de requerirse retransmisiones, se puede decidir borrar los mismos bits o borrar otros distintos, como en el caso de redundancia incremental.

La funcionalidad HARQ consiste de dos etapas de rate matching como se muestra en la figura siguiente:

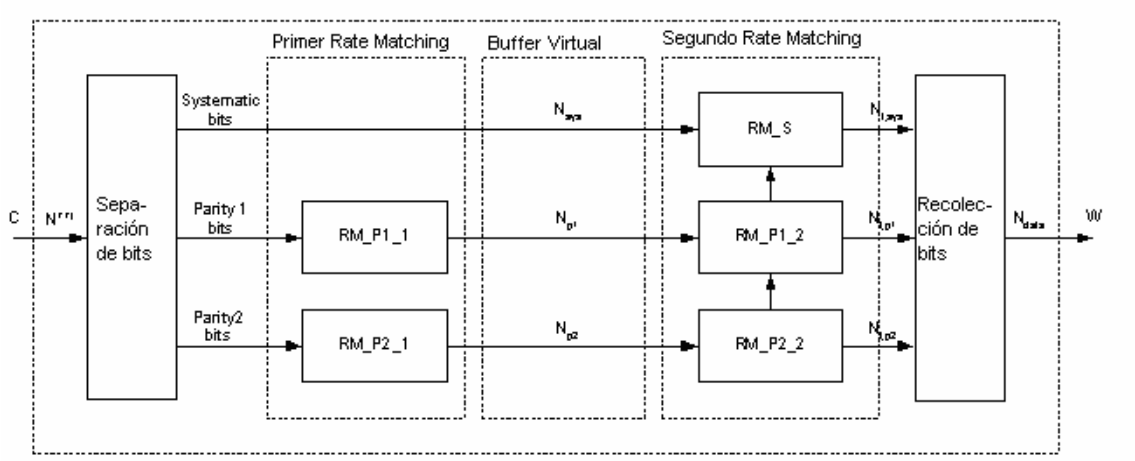


Figura 3.6: Funcionalidad HARQ

El hecho de que sea en dos etapas permite sintonizar la versión de redundancia de distintas retransmisiones cuando se usan retransmisiones no-idénticas.

El buffer mostrado en la figura anterior es virtual, ya que una implementación real puede consistir de un solo bloque de rate matching.

La primera etapa de rate matching es idéntica a la funcionalidad de rate matching de la release 99 excepto que la cantidad de bits de salida no es igual a la cantidad de bits de canal físico disponibles en el TTI de HS-DSCH. En vez de eso, la cantidad de bits de salida es igual a la capacidad de buffer del UE, información que es provista por capas superiores. Si la cantidad de bits de entrada no supera la capacidad de buffer del UE, la primera etapa es transparente. El buffer virtual simula ser el buffer del UE por lo que a partir de ahora se hablará de la capacidad de buffer virtual.

La segunda etapa iguala la cantidad de bits luego del primer rate matching a la cantidad de bits de canal físico disponibles en un TTI de HS-DSCH. El segundo rate matching también usa el algoritmo de la release 99. Sin embargo, sólo considera bits que no han sido borrados por la primera etapa de rate matching.

3.2.5.1 Primera etapa de Rate matching

Se define la cantidad de bits disponibles en el buffer virtual como N_{IR}, esta cantidad es señalada por capas superiores para cada proceso HARQ. La cantidad de bits codificados en un TTI antes del rate matching, o sea la cantidad de bits que ingresan al bloque H-ARQ, es N^{TTI}, esto es deducido de

información indicada por capas superiores y parámetros señalados en el HS-SCCH para cada TTI.

Si N_{IR} es mayor o igual que N^{TTI}, o sea todos los bits codificados turbo del correspondiente TTI pueden ser guardados, la primera etapa de rate matching debe ser transparente.

Si N_{IR} es menor que N^{TTI}, los flujos de bits de paridad son borrados de acuerdo al algoritmo de rate matching, presentado en el Anexo I donde se pueden encontrar mas detalles sobre este bloque. Para cada TTI el patrón de rate-matching se calcula con este algoritmo.

3.2.5.2 Segunda etapa de Rate matching

La segunda etapa también deberá ser realizada por medio del algoritmo anterior pero con parámetros de inicialización distintos.

Los parámetros de la segunda etapa de rate matching dependen del valor de los parámetros RV[7], s y r. El parámetro s puede tomar el valor 0 o 1 para distinguir entre transmisiones que prioricen los bits sistemáticos (s=1) y bits no sistemáticos (s=0).

Para decidir si es necesario borrar o repetir bits, se definen las siguientes variables:

N_{sys}: cantidad de bits sistemáticos
N_{p1}: cantidad de bits paridad 1
N_{p2}: cantidad de bits de paridad 2
P: cantidad de canales físicos
N_{data} = PX3XN_{data1}, la cantidad de bits disponibles para el HS-DSCH en un TTI con N_{data1} obtenido de la tabla siguiente:

Formato de ranura#i	**Tasa de bits del canal (kbps)**	**Tasa de símbolos del canal (ksps)**	**SF**	**Bits/ subtrama HS-DSCH**	**Bits/ Ranura**	**Ndata1**
0(QPSK)	480	240	16	960	320	320
1(16QAM)	960	240	16	1920	640	640

Tabla 3.1: Norma 25.211

Para $N_{data} \leq N_{sys} + N_{p1} + N_{p2}$, se realizará borrado en la segunda etapa de rate matching.

[7] El parámetro RV o versión de redundancia es el que indica que tipo de redundancia se utilizará, ya que, en caso de requerirse retransmisiones, se puede decidir borrar los mismos bits o borrar otros distintos, como en el caso de redundancia incremental.

La cantidad de bits sistemáticos transmitidos en una transmisión es $N_{t,sys} = \min\{N_{sys}, N_{data}\}$ para una transmisión que prioriza los bits sistemáticos y $N_{t,sys} = \max\{N_{data} - (N_{p1} + N_{p2}), 0\}$ para una transmisión que prioriza los bits no sistemáticos.

Para $N_{data} > N_{sys} + N_{p1} + N_{p2}$, se realizará repetición en la segunda etapa de rate matching. Una tasa de repetición similar en todos los flujos de bits es lograda fijando el número de bits sistemáticos transmitidos a $N_{t,sys} = \left\lfloor N_{sys} \cdot \frac{N_{data}}{N_{sys} + 2N_{p1}} \right\rfloor$.

La cantidad de bits de paridad en una transmisión es: $N_{t,p1} = \left\lfloor \frac{N_{data} - N_{t,sys}}{2} \right\rfloor$ y $N_{t,p2} = \left\lceil \frac{N_{data} - N_{t,sys}}{2} \right\rceil$ para los bits de paridad 1 y 2, respectivamente.

3.2.6 Segmentación de canales físicos:

Cuando se utiliza más de un HS-DPSCH, la segmentación de canales físicos divide los bits entre los distintos canales. Los bits de entrada a la segmentación de canales físicos son denotados w_1, w_2, w_3,…w_R, donde R es la cantidad de bits que entran al bloque. La cantidad de canales físicos (PhCHs) es denotada por P.

Los bits a la salida de este bloque son denotados $u_{p1}, u_{p2}, u_{p3}, \ldots, u_{pU}$, donde *p* es el número de canal físico y *U* es la cantidad de bits en un TTI por cada PhCH o sea: $U = \frac{R}{P}$

La relación entre w_k y $u_{p,k}$ es la siguiente:

Bits en el primer PhCH luego de la segmentación de canales físicos:

$u_{1,k} = w_k$ $\quad k = 1, 2, \ldots, U$

Bits en el segundo PhCH luego de la segmentación de canales físicos:

$u_{2,k} = w_{k+U}$ $\quad k = 1, 2, \ldots, U$

…

Bits en el P-ésimo PhCH luego de la segmentación de canales físicos:

$u_{P,k} = w_{k+(P-1)\times U}$ $\quad k = 1, 2, \ldots, U$

3.2.7 Intercalado (Interleaving) HS-DSCH:

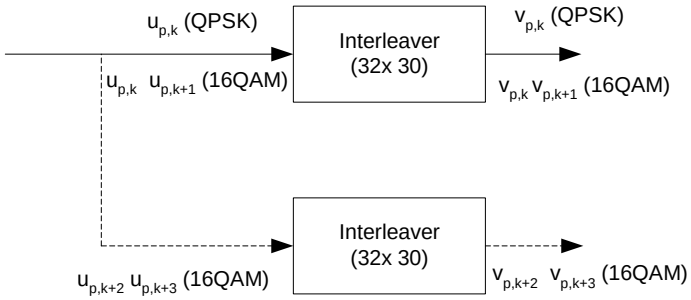


Figura 3.7: Intercalador 16QAM

El intercalado se hace por separado para cada canal. Para QPSK U = 960 y para 16QAM U = 1920. El intercalador básico es de tamaño fijo: R2=32 filas y C2=30 columnas.

Para 16QAM, hay 2 intercaladores idénticos del mismo tamaño fijo R2×C2 = 32×30. Los bits de salida de la segmentación de canales físicos se dividen dos por dos entre los intercaladores: los bits $u_{p,k}$ y $u_{p,k+1}$ van al primer intercalador y los bits $u_{p,k+2}$ y $u_{p,k+3}$ van al segundo intercalador. Los bits se juntan dos por dos de los intercaladores: los bits $v_{p,k}$ y $v_{p,k+1}$ son obtenidos del primer intercalador y los bits $v_{p,k+2}$ y $v_{p,k+3}$ son obtenidos del segundo intercalador donde k mod 4=1.

3.2.7.1 Funcionamiento de los intercaladores:

Se escribe la secuencia de bits de entrada $u_{p,1}, u_{p,2}, u_{p,3}, \ldots, u_{p,U}$ en la matriz de tamaño R2×C2 = 32×30, fila por fila, comenzando con el bit $y_{p,1}$ en la columna 0 de la fila 0:

$$\begin{bmatrix} y_{p,1} & y_{p,2} & y_{p,3} & \cdots & y_{p,\mathrm{C2}} \\ y_{p,(\mathrm{C2}+1)} & y_{p,(\mathrm{C2}+2)} & y_{p,(\mathrm{C2}+3)} & \cdots & y_{p,(2\times\mathrm{C2})} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ y_{p,((\mathrm{R2}-1)\times\mathrm{C2}+1)} & y_{p,((\mathrm{R2}-1)\times\mathrm{C2}+2)} & y_{p,((\mathrm{R2}-1)\times\mathrm{C2}+3)} & \cdots & y_{p,(\mathrm{R2}\times\mathrm{C2})} \end{bmatrix}$$

donde $y_{p,k} = u_{p,k}$ para k = 1, 2, …, U

Si R2 × C2 > U, se ponen bits de relleno tal que $y_{p,k}$ = 0 ó 1 para $k = U + 1, U + 2, \ldots,$ R2 × C2. Estos bits de relleno se quitan de la salida de la matriz luego de la permutación inter-columna.

Se realiza la permutación inter-columna para la matriz basada en el patrón que se muestra en la tabla siguiente.

Cantidad de columnas C2	**Patrón de permutación Inter-columna** **< P2(0), P2(1), …, P2(C2-1) >**
30	<0, 20, 10, 5, 15, 25, 3, 13, 23, 8, 18, 28, 1, 11, 21, 6, 16, 26, 4, 14, 24, 19, 9, 29, 12, 2, 7, 22, 27, 17>

Luego de la permutación de las columnas, los bits son denotados $y'_{p,k}$.

$$\begin{bmatrix} y'_{p,1} & y'_{p,(\mathrm{R2}+1)} & y'_{p,(2\times\mathrm{R2}+1)} & \cdots & y'_{p,((\mathrm{C2}-1)\times\mathrm{R2}+1)} \\ y'_{p,2} & y'_{p,(\mathrm{R2}+2)} & y'_{p,(2\times\mathrm{R2}+2)} & \cdots & y'_{p,((\mathrm{C2}-1)\times\mathrm{R2}+2)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ y'_{p,\mathrm{R2}} & y'_{p,(2\times\mathrm{R2})} & y'_{p,(3\times\mathrm{R2})} & \cdots & y'_{p,(\mathrm{C2}\times\mathrm{R2})} \end{bmatrix}$$

La salida del bloque intercalador es la secuencia de bits leída columna por columna de la matriz con permutación inter-columnas. Se eliminan los bits de relleno en caso que se hayan agregado, o sea, los bits $y'_{p,k}$ que corresponden a los bits $y_{p,k}$ con *k>U* son eliminados de la salida.

3.2.8 Reordenamiento de constelación para 16QAM

Este bloque aplica solamente a 16QAM, por lo tanto es transparente para QPSK. En 16QAM, dos de los cuatros bits de un símbolo tienen una probabilidad de error mayor que los otros dos. El reordenamiento es aplicado durante la retransmisión y provee igual probabilidad de error a todos los bits en promedio después de la retransmisión.

La secuencia de bits que entra es mapeada en grupos de cuatro bits tal que $v_{p,k}$, $v_{p,k+1}$, $v_{p,k+2}$, $v_{p,k+3}$ son los bits de entrada al bloque con k mod 4 = 1.

La tabla siguiente describe la operación que producen los diferentes reordenamientos. El parámetro de constelación b, que forma parte de los parámetros que generan la versión de redundancia RV, es quien indica que tipo de reordenamiento se le realiza a la constelación.

Parámetro de constelación *b*	**Secuencia de salida**	**Operación**
0	$v_{p,k}\, v_{p,k+1} v_{p,k+2}\, v_{p,k+3}$	Ninguna
1	$v_{p,k+2} v_{p,k+3} v_{p,k}\, v_{p,k+1}$	Intercambiar MSBs con LSBs
2	$v_{p,k}\, v_{p,k+1} \overline{v_{p,k+2}}\, \overline{v_{p,k+3}}$	Inversión de los valores lógicos de LSBs
3	$v_{p,k+2} v_{p,k+3} \overline{v_{p,k}}\, \overline{v_{p,k+1}}$	Intercambiar MSBs con LSBs e inversión de los valores lógicos de LSBs

Tabla 3.2: Reordenamiento de constelación para 16QAM

Los bits de salidas son $r_{p,k}$, $r_{p,k+1}$, $r_{p,k+2}$, $r_{p,k+3}$, donde k mod 4 = 1.

3.2.9 Mapeo de canales físicos:

Los bits que entran al mapeo de canales físicos son denotados por $r_{p,1}$, $r_{p,2}$,...,$r_{p,U}$, donde *p* es el número de PhCH y U es la cantidad de bits en una trama de radio por un PhCH. Los bits $r_{p,k}$ son mapeados a los canales físicos tal que los bits de cada PhCH son transmitidos por el aire en orden ascendente respecto de *k*.

3.2.10 Spreading

La figura muestra el bloque Spreading para los canales físicos (menos para el SCH). Este bloque consiste básicamente en dos operaciones fundamentales: canalización y scrambling.

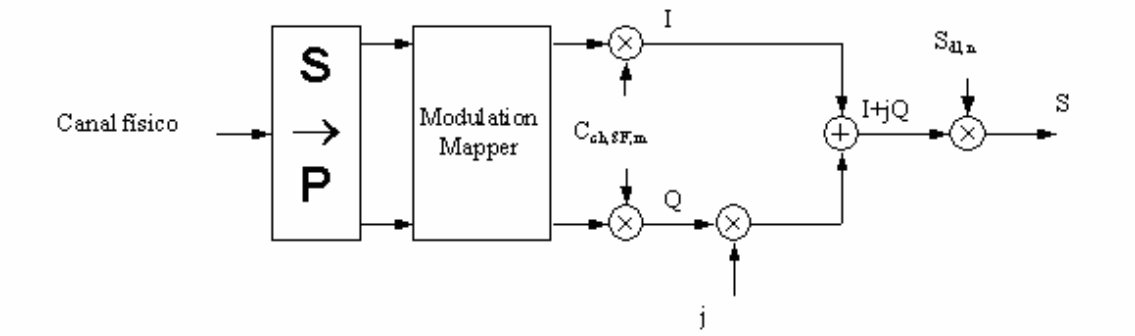


Imagen 3.8 Spreading de canales físicos

En primer lugar los bits que llegan desde los canales físicos pasan por un conversor serial-paralelo y luego por el Modulation Mapper. La salida de este sub-bloque es diferente según se esté utilizando QPSK o 16QAM.

Para el caso de QPSK, los bits son convertidos de serial a paralelo y el Modulation mapper se encarga del mapeo de los bits pares e impares a las ramas I y Q respectivamente.

En el caso de modulación 16QAM, se toma un grupo de cuatro bits que son convertidos a dos bits en la rama I y dos en la rama Q, y luego mapeados a 16QAM según la tabla siguiente.

$i_1q_1i_2q_2$	**Rama I**	**Rama Q**
0000	0.4472	0.4472
0001	0.4472	1.3416
0010	1.3416	0.4472
0011	1.3416	1.3416
0100	0.4472	-0.4472
0101	0.4472	-1.3416
0110	1.3416	-0.4472
0111	1.3416	-1.3416
1000	-0.4472	0.4472
1001	-0.4472	1.3416
1010	-1.3416	0.4472
1011	-1.3416	1.3416
1100	-0.4472	-0.4472
1101	-0.4472	-1.3416
1110	-1.3416	-0.4472
1111	-1.3416	-1.3416

Tabla 3.3: Símbolos 16QAM

Luego de este sub-bloque de mapeado según la modulación, a cada rama se le aplica primero la operación de canalización, aumentando su ancho de banda al multiplicar en cada rama por un mismo código de canalización. Estas secuencias de chips en cada rama I y Q son convertidas a una secuencia simple compleja a la cual se le aplica la operación de scrambling, una multiplicación compleja por un 'scrambling code'.

Para la canalización se utilizan códigos OVSF (Orthogonal Variable Spreading Factor). Estos códigos son ortogonales, o sea que no interfieren unos con otros, solamente si son sincronizados en el tiempo. Por lo tanto pueden ser utilizados en el enlace de bajada para separar diferentes usuarios de una misma celda, pero no para identificar una celda ya que las celdas no están sincronizadas y por lo tanto podrían tener códigos no ortogonales entre sí. Para el enlace de subida los códigos de canalización separan diferentes servicios de un mismo usuario. No se pueden usar para diferenciar usuarios de una misma celda ya que al igual que las celdas en el enlace de bajada, los usuarios no están sincronizados entre sí en el tiempo y por lo tanto sus códigos pueden no ser ortogonales.

Como la operación de canalización no es suficiente para identificar totalmente la comunicación, se utilizan además los códigos de scrambling. Con ellos se identifica la celda y el usuario. El scrambling utiliza códigos de pseudos-ruido.

En la tabla siguiente se pueden ver algunas características y las diferencias de estos códigos:

	Código de canalización	Código de scrambling
Uso	UL: Separa canales físicos y de control de un mismo usuario DL: Separa diferentes usuarios en una celda	UL: Identifica terminales DL: Identifica celdas
Tipo	OVSF	Códigos de pseudos-ruido
Expande espectro	Si	No
Nro de códigos	=SF	512 primarios + 15 secundarios para cada uno
Longitud	4-512	38400 chips

Tabla 3.4: Códigos utilizados en el bloque spreading

A cada HS-PDSCH le es asignado un código de canalización con un spreading factor fijo igual a 16. Como la tecnología HSDPA permite la transmisión de multicódigo, a un UE se le puede asignar múltiples códigos de canalización (múltiple HS-PDSCHS) en la mismo subtrama de HS-PDSCH.

La definición de los códigos OVSF y códigos de scrambling utilizados puede encontrarse en el Anexo.

3.2.11 Modulación

La secuencia de chips resultante del bloque de spreading es modulada como indica la figura:

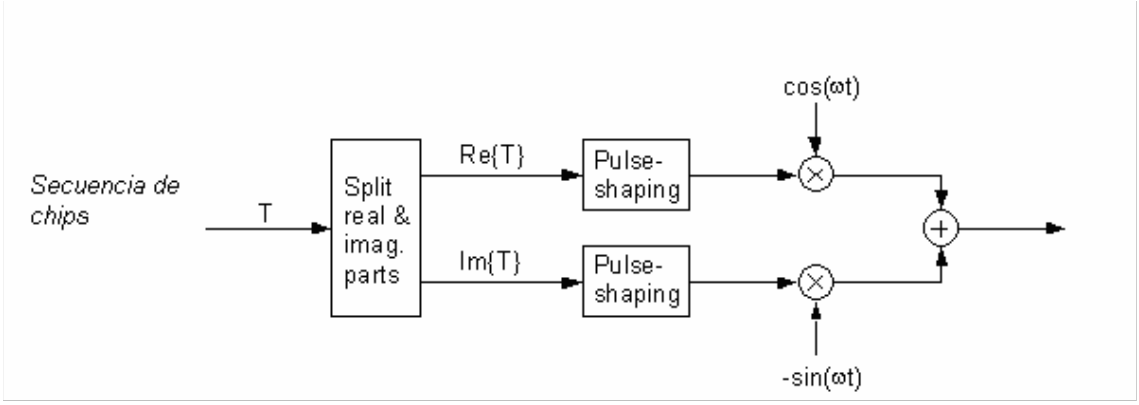


Imagen 3.9 Modulación

La ecuación para el pulse shaping es la siguiente:

$$RC_0(t) = \frac{\sin\left(\pi \frac{t}{T_C}(1-\alpha)\right) + 4\alpha \frac{t}{T_C}\cos\left(\pi \frac{t}{T_C}(1+\alpha)\right)}{\pi \frac{t}{T_C}\left(1-\left(4\alpha \frac{t}{T_C}\right)^2\right)}$$

Donde:

factor de roll-off $\alpha = 0.22$ y el tiempo de chip:

$$T_c = \frac{1}{chiprate} \approx 0.26042\mu s$$

3.3 Canal CPICH (Common Pilot Channel)

El Canal de pilots común primario es usado por los UEs para identificar primero el código de scrambling primario usado para las transmisiones del canal físico de control común primario (P-CCPCH) desde el Nodo B. Posteriores canales CPICH permiten estimaciones de fase y potencia, asi como ayudar a descubrir otros caminos de radio.

La norma TS 25.211 define el canal CPICH primario con una tasa fija de 30kbps y un spreading factor de 256, código 0, y se modula en QPSK.

La figura muestra la estructura de la trama del canal CPICH.

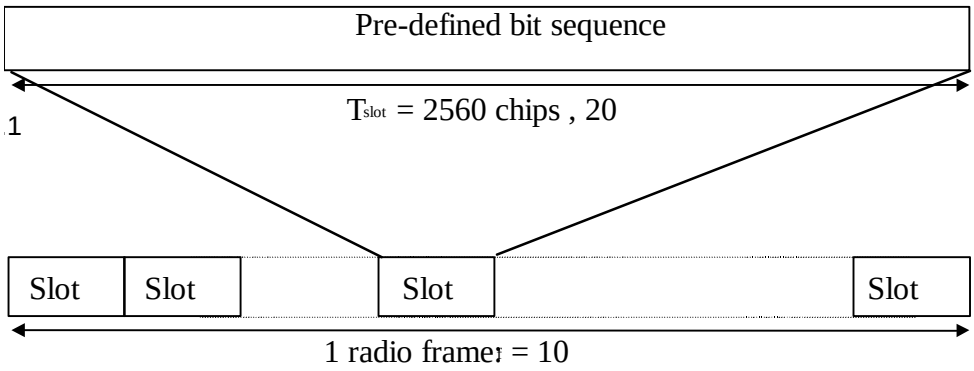


Figura 3.10: Estructura de trama de CPICH

Esta señal se transmite junto al canal de datos y será utilizada como señal de referencia en el receptor.

Opcionalmente un Nodo B puede transmitir uno o más canales de pilotos comunes secundarios (S-CPICH), que usan 256 códigos elegidos arbitrariamente, que se notan como $C_{ch,256,n}$ donde $0 < n < 256$.

Las señales piloto no son un requisito de CDMA, sin embargo, hacen que el receptor del UE sea más sencillo y mejoran la fiabilidad del sistema.

4 Modelo a implementar

El modelo que se va a simular corresponde a un solo usuario utilizando los quince canales físicos para obtener la máxima tasa de datos posible.

4.1 Transmisor

El modelo del transmisor está definido por la norma como se describió en el punto anterior. En el mismo se describe la tecnología a nivel de capa física a modo general sin entrar en el caso particular de estudio, el de transmitir a la máxima tasa de datos. En este capítulo nos centramos en presentar la información necesaria para implementar nuestro caso.

Lo que resta definir es el flujo de bits que se va a transmitir

4.1.1 Flujo de bits

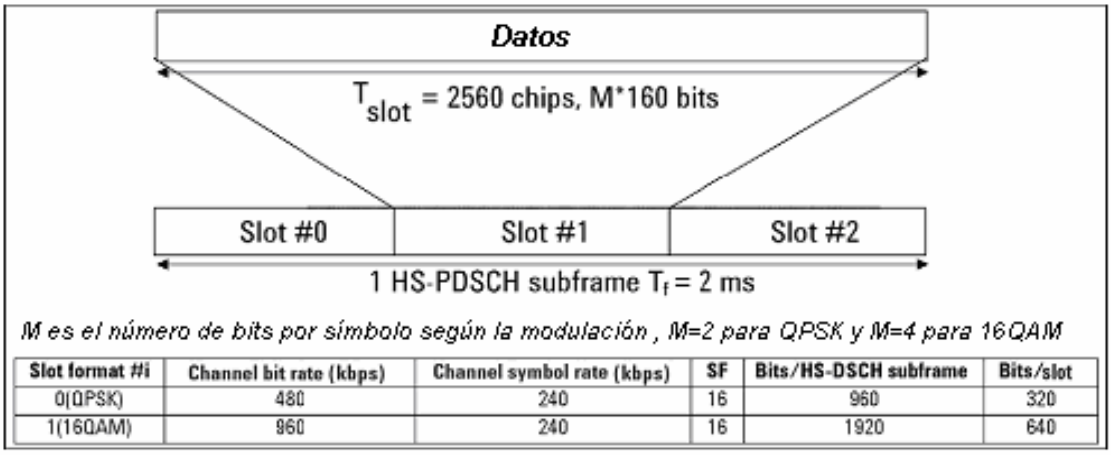
El objetivo es lograr la máxima tasa de datos posible, por lo que se buscaron las mejores capacidades y las mejores opciones. De esta manera definimos trabajar con los equipos de usuario (UE) de categoría 10.

HS-DSCH category	Maximum number of HS-DSCH codes received	Minimum inter-TTI interval	Maximum number of bits of an HS-DSCH transport block received within an HS-DSCH TTI	Total number of soft channel bits
Category 1	5	3	7298	19200
Category 2	5	3	7298	28800
Category 3	5	2	7298	28800
Category 4	5	2	7298	38400
Category 5	5	1	7298	57600
Category 6	5	1	7298	67200
Category 7	10	1	14411	115200
Category 8	10	1	14411	134400
Category 9	15	1	20251	172800
Category 10	**15**	**1**	**27952**	**172800**
Category 11	5	2	3630	14400
Category 12	5	1	3630	28800

Tabla 4.1: Norma 25.396

Como se puede ver en la tabla esto implica que la cantidad máxima de bits por bloque de transporte es de 27.952 y que se pueden utilizar hasta quince canales físicos para transmitir a un usuario con un UE de esta categoría.

Por la misma razón se decidió usar 16QAM, lo que resulta en una cantidad de 1920 bits por trama de 2ms, como se puede ver en la tabla siguiente.



M es el número de bits por símbolo según la modulación , M=2 para QPSK y M=4 para 16QAM

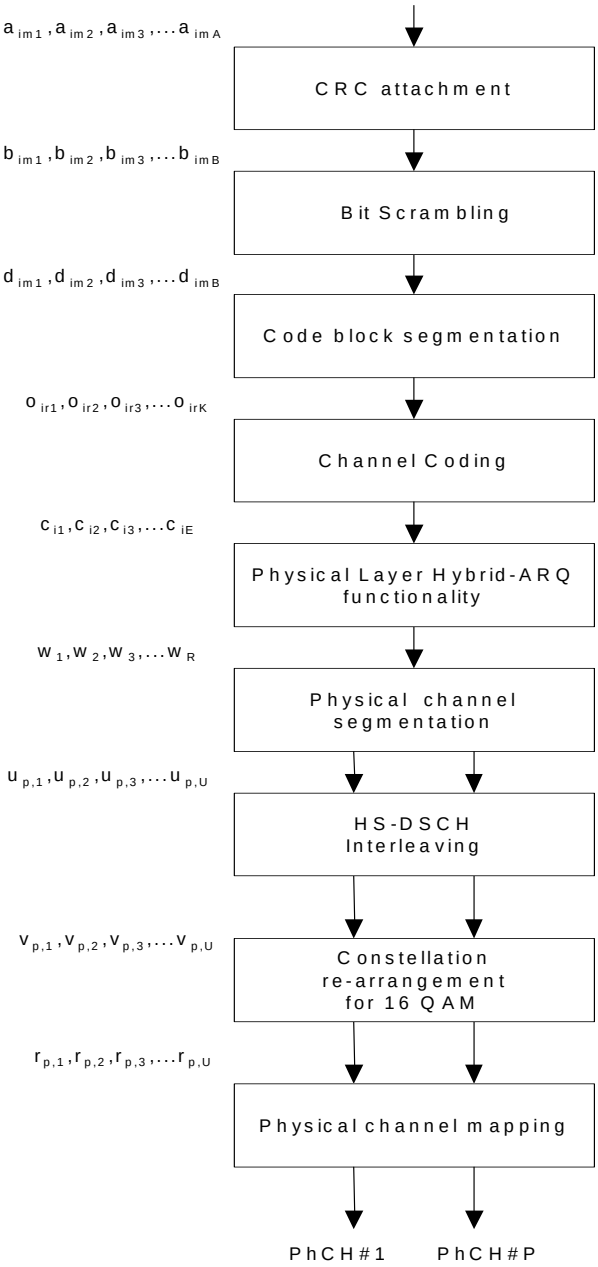
Slot format #i	Channel bit rate (kbps)	Channel symbol rate (kbps)	SF	Bits/HS-DSCH subframe	Bits/slot
0(QPSK)	480	240	16	960	320
1(16QAM)	960	240	16	1920	640

Figura 4.1: Canal de HS-PDSCH

Con esta información se deduce que la cantidad total de bits que se pueden transmitir en una trama de 2 ms, es igual a 1920 bits por la cantidad de canales que el UE puede manejar (15), lo que resulta en 28800 bits. Teniendo en cuenta el spreading factor de 16 y la modulación 16QAM (cuatro bits por símbolo), se calcula la tasa de chips como:

Tasa de chips = 2ms / [(1920b/4)*16] = 2ms / 7680chips = 3.84 Mcps
Y la tasa de bits será igual a: (1920b / 2ms)*15 = 14.4 Mbs

El siguiente es el esquema del conjunto de bloques de multiplexado y codificación de canal que componen la capa física con la cantidad de bits que ingresan en cada bloque para nuestro sistema, detallamos a continuación cómo se modifica la cantidad de bits por trama en cada bloque



A = 27952

B = 27952 +24 = 27976

K = 4663
6 bloques => 4663*6 = 27978

E = 6*(3*K+12) = 84006

R = 28800 (= 1920*15)

U = 1920 (para cada canal físico p)

Cada canal físico p, transmite 1920 bits por trama.

Figura 4.2: Cadena de bloques para HS-DSCH

CRC:
Definido el UE que se va a utilizar, categoría 10, sabemos entonces que al bloque CRC ingresan A = 27.952 bits. El bloque de CRC agrega 24 bits por lo que a su salida tendremos B = 27.976 bits.

Bit Scrambling:
El bit scrambling no altera la cantidad de bits por lo que a su entrada y a su salida tendremos 27.976 bits.

Segmentación:
La segmentación de bloques de código define la cantidad de bloques de código como:

$$C_i = \lceil B \ / Z \rceil$$

Siendo B = 27.976 y Z el tamaño máximo de bloques de código, 5114, Ci vale 6. Como evidentemente B no es múltiplo de 6, se deben agregar bits de relleno en este bloque. Siguiendo el algoritmo de segmentación, resulta en que la cantidad de bits de relleno es igual a 2. De esta manera se conformarán 6 bloques de código de K bits, con K = 4663.

Codificación
Aquí ingresan los seis bloques que forma la segmentación, de longitud K (4663 bits). Cada uno de ellos entrará al codificador Turbo serialmente. Por lo tanto la salida de cada bloque de código será de longitud $Y_i = 3*K + 12$, por tratarse de un codificador Turbo con tasa 1/3 que además agrega los 12 bits de terminación Trellis. Por lo tanto a la salida de la codificación de canal, una trama consistirá de 6 bloques de 14.001 bits, o de 84.006 bits.

H-ARQ
En nuestro caso, como se busca transmitir a la máxima tasa de datos no se implementan retransmisiones en caso de error. Por lo tanto este bloque solamente se encarga del emparejamiento de la tasa de bits entre los bits que salen del decodificador y la cantidad de bits que va a transmitir, 28800.

La primera etapa de rate matching no se ejecuta ya que la cantidad de bits que puede almacenar el buffer, $N_{IR,}$ es mayor que la cantidad de bits que llegan en un TTI, N^{TTI}. N_{IR} para terminales categoría 10 es 172.800 como se ve en la tabla 4.1, y N^{TTI} es la cantidad de bits que ingresan al HARQ en una trama, definido en la codificación como 84.006 bits.

La segunda etapa se ejecuta para los bits de paridad 1 y 2. Esta operación es la que se encarga de descartar los bits necesarios para lograr los 28.800 bits a la salida, que es el máximo que se podrá transmitir.

Segmentación de canales físicos:
La segmentación de canales físicos separa los bits entre los canales, por lo que en cada canal físico tenemos 28.800/15=1.920 bits como corresponde.

Interleaving y Reordenamiento 16 QAM
Las etapas de intercalado y reordenamiento de constelación para 16QAM no alteran la cantidad de bits por lo que tenemos a sus entradas y salidas 1.920 bits por cada canal físico.

La etapa de mapeo de canales físicos, no se realiza, ya que el canal HS-DSCH se mapea directamente en el HS-DPSCH.

4.2 Receptor

4.2.1 Simple

El primer modelo de receptor que se implementó, denominado receptor simple, es directamente la implementación de los bloques inversos del transmisor. Se estudió el comportamiento de cada bloque y se buscó la forma de implementar el bloque con la operación inversa para cada uno de los elementos del transmisor. De esta forma se creó un receptor que reconoce perfectamente la señal transmitida en un ambiente sin ruido. Pero es muy vulnerable a un entorno con múltiples caminos.

4.2.2 Rake

En el sistema rake a implementar, cada rama recibe la señal proveniente de cada camino. Como se trabaja con un modelo de canal SUI 6[8], se utiliza un rake de tres ramas activas para dedicar una de ellas a cada camino.

En la figura se puede ver un modelo similar al utilizado en la implementación. En la parte superior se introducen los delays para cada rama. En nuestro modelo estos retardos son parámetros seteables del bloque, de forma de ajustarlo al canal que se esté utilizando.

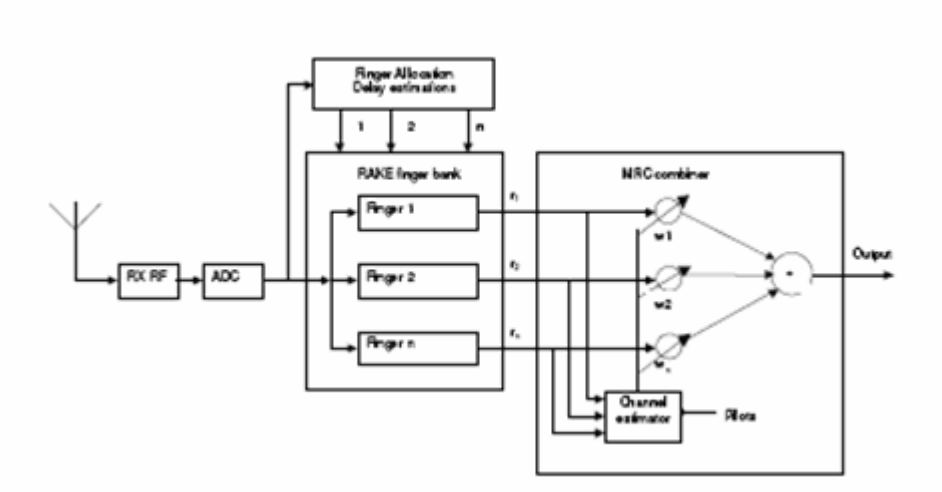


Figura 4.3: Receptor Rake

Cada rama se encarga de obtener la señal que le corresponda, según el delay introducido y luego convertirla a símbolo 16QAM con los códigos de canalización y códigos de scrambling correctos.

El MRCcombiner se ocupa de combinar las señales provenientes de cada rama para formar el símbolo de salida final. En esta operación se incluye la estimación del canal que permite ponderar las señales según sea la atenuación estimada para cada camino. La estimación del canal se lleva a

[8] Modelo de canal multicamino. Se presenta en punto 4.3

cabo con la información de la señal de referencia transmitida (CPICH) más información de la misma señal generada en el receptor.

La estrategia de combinación coherente óptima es MRC (Maximum Ratio Combining).

MRC es la forma óptima de combinación de diversidad porque busca el máximo SNR posible. Los símbolos de salida de ramas diferentes son multiplicados por el complejo conjugado de la estimación de canal, estos resultados son sumados para lograr el símbolo "combinado". Esto es ilustrado en la Figura:

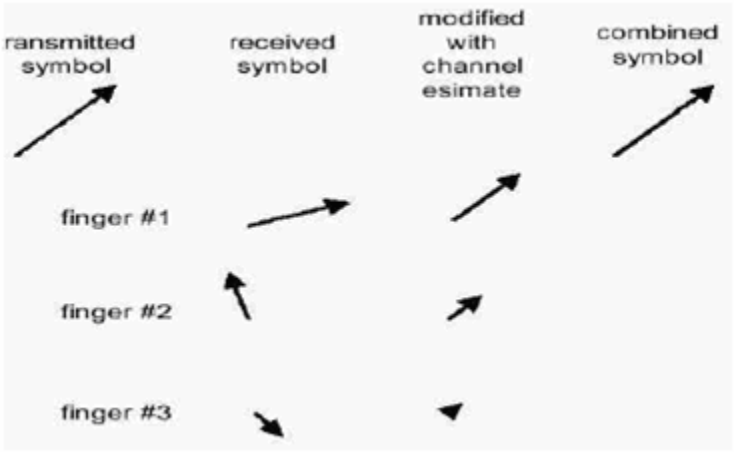


Figura 4.4: Maximal Ratio Combining

4.2.3 Ecualizador

Este receptor implementa un ecualizador iterativo que se puede separar en dos bloques principales: un FIR que filtrará la señal recibida, y un actualizador de los coeficientes del mismo. En este caso también es utilizada, al igual que en el receptor RAKE, la señal de referencia del canal CPICH (Pilots) para obtener información del canal.

La imagen muestra un modelo de un posible ecualizador iterativo.

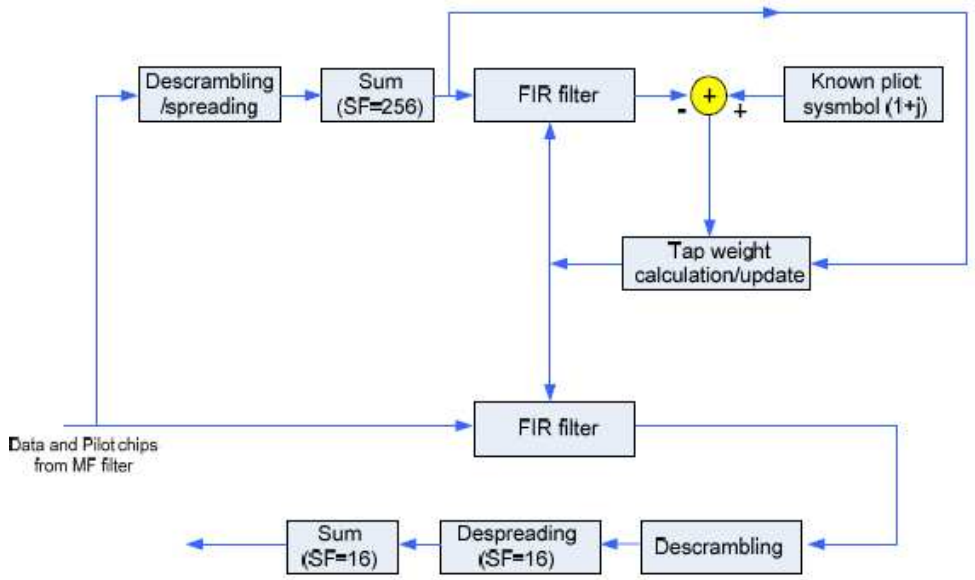


Figura 4.5: Modelo de ecualizador iterativo

El funcionamiento del ecualizador consiste en separar en primer lugar la señal de referencia de la señal de datos. Para eso se le aplica a la señal recibida el código OVSF correspondiente a la señal de referencia y se obtienen los símbolos PILOTS. Esta señal de PILOTS recibida y la señal original que es generada nuevamente en el receptor, se utilizan en el actualizador para calcular los coeficientes necesarios para que un filtro FIR devuelva la señal de datos (o pilots) correcta.

El actualizador genera los coeficientes que son utilizados para recuperar tanto los datos como los PILOTS. Es un bloque realimentado ya que tiene como entrada la diferencia entre la señal de referencia generada en el receptor, y la señal recuperada a través de los coeficientes calculados, como se puede ver en la figura el *bloque Tap Weight Calculation/Update*.

Para calcular estos coeficientes se eligió trabajar con el algoritmo RLS, Recursive Least Square, detallado en Anexo. Esta decisión se basó en algunas ventajas de dicho algoritmo como ser: mayor convergencia y estabilidad, además de ser un caso que se encontró repetidas veces en diferentes materiales de investigación.

Los coeficientes calculados se utilizan en un FIR que se le aplica a la señal de datos.

Este ecualizador puede ser implementado de diferentes formas: a nivel de símbolo o a nivel de chip. La diferencia radica en las señales que se utilizan para actualizar los coeficientes del filtro FIR y en la forma en que éste es aplicado.

En el proyecto se presentarán las dos variantes.

Nivel de símbolo

En este caso, a la señal recibida se le aplica en primer lugar la operación despreading para obtener los símbolos 16QAM de la señal de datos, y los símbolos QPSK de la señal de referencia (pilots). Estos símbolos son los que se usarán en el ecualizador, tanto en el actualizador de coeficientes como en el filtro FIR. Por lo tanto el actualizador calcula los nuevos coeficientes para cada nuevo símbolo PILOT que llega. Por otro lado a la señal de datos, ya a nivel de símbolos se le aplica el filtro FIR, también aplicando el filtro para cada símbolo de datos. Se debe recordar que según la norma, se transmiten 16 símbolos de datos mientras 1 símbolo de PILOT es transmitido, por lo tanto los coeficientes se calculan para cada símbolo PILOT deben ser mantenidos en el FIR durante esos 16 símbolos de datos.

Nivel de Chip

Este caso es similar al anterior, pero basándose en que se deben obtener los mismos resultados al aplicar el FIR y luego el despreading o al hacerlo al revés [7].

En este caso el actualizador y el filtro se aplican a cada chip y no por símbolo. La señal de referencia debe separarse de la señal de datos por medio del despreading primero, para luego inmediatamente volver a aplicarle la operación inversa, el spreading, para volver a su nivel de chip, pero ahora sólo serán chips de pilots que no están mezclados con datos. También la señal de referencia generada en el receptor es convertida a chips, asi el actualizador calcula los nuevos coeficientes para cada chip que llega. La señal de datos simplemente es pasada por el filtro FIR como se recibe y luego sí, se convierte a los símbolos 16QAM esperados.

4.3 Canal multicamino

Para simular los efectos de un canal multicamino se utiliza el modelo SUI-6, con una distribución de Rayleigh.

Existen modelos sencillos que sirven para evaluar la calidad de los enlaces radioeléctricos individuales, pero se necesitan modelos más complejos para evaluar la fiabilidad a nivel de todo el sistema y la conveniencia de las tecnologías específicas. Para las tecnologías de banda estrecha, la dispersión de los retardos temporales puede caracterizarse por un solo valor r.m.s., sin embargo para las tecnologías de banda ancha, el número, intensidad y retardo temporal relativo de las múltiples componentes de la señal puede resultar importante [8].

4.3.1 SUI (Stanford University Interim)

Según [9] el canal inalámbrico se puede caracterizar definiendo los parámetros necesarios para describir los siguientes fenómenos:

- Pérdida por trayecto
- Retardo de multicamino
- Desvanecimiento (Fading)
- Efecto Doppler
- Interferencia de canales adyacentes

Estos parámetros dependen de las condiciones del entorno: terreno, densidad de árboles, altura de antena, velocidad del viento y época del año. Por esta razón resulta evidente que serán parámetros aleatorios y por lo tanto pueden definirse solo estadísticamente. Se especifica por lo general su valor medio y la varianza.

Los modelos SUI fueron desarrollados por la universidad de Stanford. Estos modelos especifican los parámetros necesarios para modelar los distintos fenómenos aleatorios que describen un canal.

Se definen 6 modelos SUI en función de los 3 tipos de terrenos típicos existentes en Estados Unidos, pero que pueden ser extendidos a todo el mundo. La categoría con máxima pérdida por trayecto es la de un terreno montañoso con una densidad de árboles de moderada a alta, llamada categoría A. La categoría con la menor perdida por trayecto es la de un terreno llano con poca densidad de árboles, categoría C. La categoría B corresponde a un escenario intermedio. Estos modelos pueden ser usados para simulaciones, diseño, desarrollo y testeo de tecnologías inalámbricas. Los parámetros fueron seleccionados basándose en modelos estadísticos de los fenómenos nombrados anteriormente.

Tipo de Terreno	**Canal SUI**
A	SUI 1, SUI 2
B	SUI 3, SUI 4
C	SUI 5, SUI6

Tabla 4.2: Tipo de Terreno

La atenuación de multicamino del canal es modelada por tres caminos con retardos no uniformes. La ganancia asociada a cada trayecto es caracterizada con una distribución Ricean, en el caso de K > 0, o Rayleigh con K = 0, y una frecuencia de Doppler máxima, siendo K un factor que depende del entorno.

La tabla 4.2 presenta los parámetros del canal SUI – 6.

SUI – 6 Channel				
	Tap 1	**Tap 2**	**Tap 3**	**Units**
Delay	0	14	20	µs
Power (omni ant.)	0	-10	-14	dB
90% K-fact. (omni)	0	0	0	
75% K-fact. (omni)	0	0	0	
50% K-fact. (omni)	1	0	0	
Power (30° ant.)	0	-16	-26	dB
90% K-fact. (30°)	0	0	0	
75% K-fact. (30°)	2	0	0	
50% K-fact. (30°)	5	0	0	
Doppler	0.4	0.3	0.5	Hz

Antenna Correlation: $\rho_{ENV} = 0.3$
Gain Reduction Factor: GRF = 4 dB
Normalization Factor: $F_{omni} = -0.5683$ dB, $F_{30°} = -0.1184$ dB

Terrain Type: A
Omni antenna: $\tau_{RMS} = 5.240$ µs
overall K: K = 0.1 (90%); K = 0.3 (75%); K = 1.0 (50%)
30° antenna: $\tau_{RMS} = 2.370$ µs
overall K: K = 0.4 (90%); K = 1.3 (75%); K = 4.2 (50%)

Tabla 4.2: Modelo SUI 6

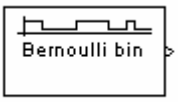
5 Implementación en Simulink

El programa utilizado fue Matlab/Simulink.
A continuación se presenta una breve descripción de cada bloque para el caso del transmisor y para el bloque inverso implementado en el receptor. Información más detallada de algunos bloque puede encontrarse en el Anexo F.

5.1 Generador:

Los bits se introducen al sistema por medio de un generador binario aleatorio. El mismo es el bloque Bernoulli Binary Generator.
Este bloque se encuentra en la librería de Simulink en: Communications blockset → Comm Sources → Random Data Sources

5.1.1 Descripción Generador:



Este bloque genera números binarios aleatorios usando una distribución de Bernoulli. La distribución de Bernoulli con parámetro p produce cero con una probabilidad p y uno con una probabilidad 1-p. Tiene un valor medio 1-p y una varianza p(1-p). El parámetro probabilidad de un cero específica p, y puede ser cualquier número real entre cero y uno.

Parámetros utilizados

Se generan tramas de 27.952 bits cada 2 milisegundos con una probabilidad de 1/2. La semilla inicial se eligió aleatoriamente.

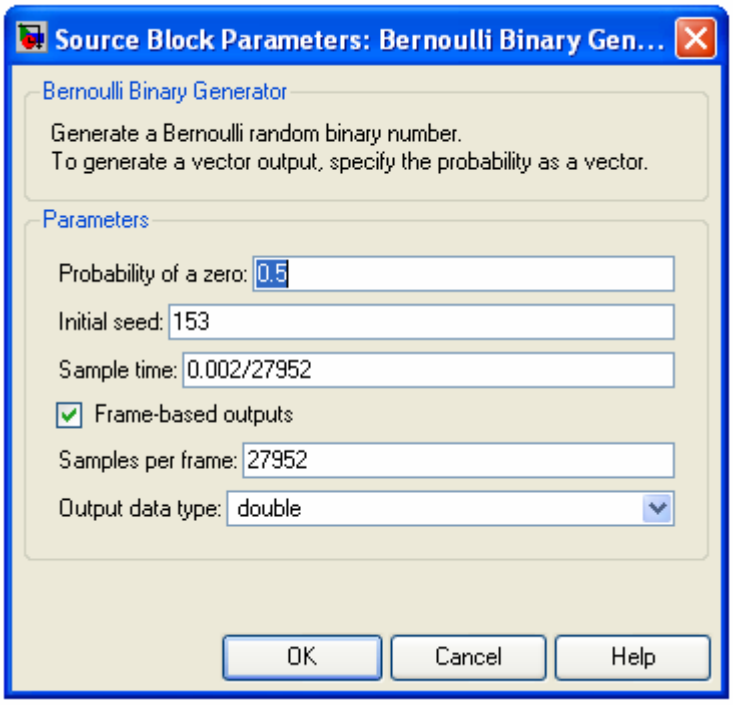


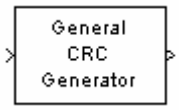
Figura 5.1: Parámetros para el generador

5.2 Bloque CRC

5.2.1 Agregado CRC (Tx)

En Simulink existen dos opciones para el agregado de CRC. El bloque CRC-N Generator y el General CRC Generator. El primero utiliza polinomios generadores predefinidos estandarizados. Se verificó en la norma que el polinomio generador que utiliza HSDPA no es el estandarizado, por lo que es necesario un bloque en el cual sea posible elegir el polinomio generador que está definido en la norma. Por esta razón se utiliza el segundo bloque, General CRC Generador.

5.2.1.1 Descripción CRC



Este bloque genera códigos de redundancia cíclica (CRC) para cada trama de datos de entrada y los agrega a la trama. Se especifica el polinomio generador como parámetro del bloque. Se representa en polinomio en una de estas dos formas:

- Como un vector binario en fila conteniendo los coeficientes de las potencias en orden descendente. Por ejemplo, [1 1 0 1] representa al polinomio $x^3 + x^2 + 1$.

- Como un vector entero en fila conteniendo las potencies de los términos que no son cero, en orden descendiente. Por ejemplo, [3 2 0] representa al polinomio $x^3 + x^2 + 1$.

Se especifica el estado inicial de los shift registers internos con el parámetro initial state

El polinomio generador en la norma es:
$g_{CRC24}(D) = D^{24} + D^{23} + D^6 + D^5 + D + 1$

Se define como un vector binario conteniendo los coeficientes de las potencias en orden descendente. Los estados iniciales deben ser nulos.

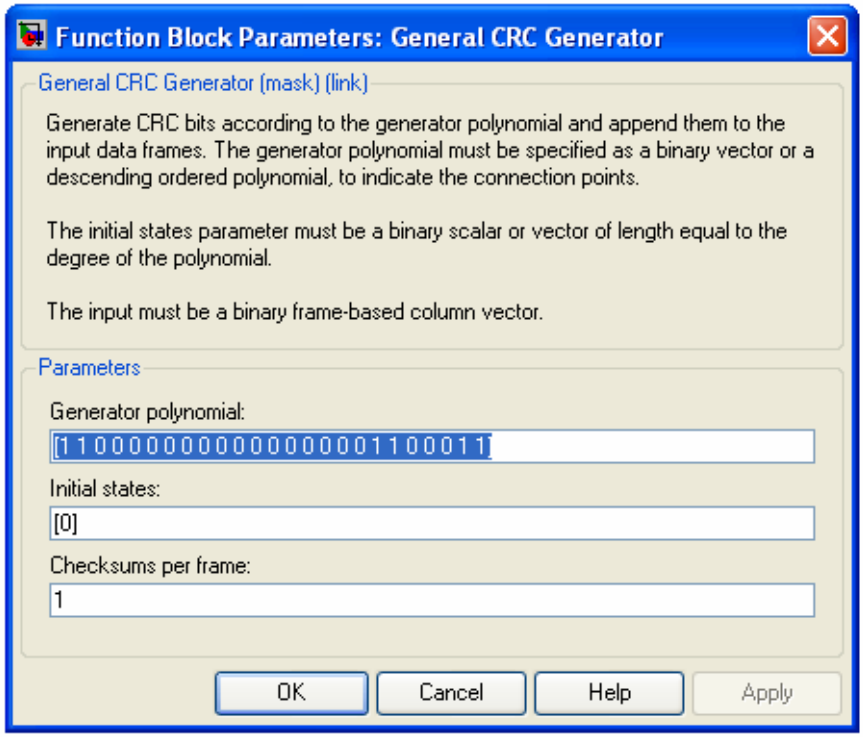
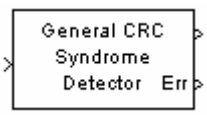


Figura 5.2: Parámetros para el bloque CRC

5.2.2 Detección CRC

Nuevamente existen dos opciones: CRC-N Syndrome Detector y General CRC Syndrome Detector, cada una correspondiente al bloque generador. Por lo tanto, elegimos el General CRC Syndrome Detector.



Este bloque computa checksums para la trama entera de entrada. La segunda salida del bloque es un vector cuyo tamaño es la cantidad de checksums y cuyas entradas son 0 si la computación de checksum da un valor cero y 1 en

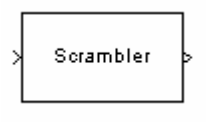
el caso contrario. La primera salida es el conjunto de mensajes con los checksums retirados.

Los parámetros ingresados deben coincidir con los del bloque General CRC Generator.

5.3 Bit Scrambling

5.3.1 Scrambler (Tx)

Para implementar esta operación se utilizó el bloque scrambler de Simulink, luego de verificar que su funcionamiento es el que la norma requiere.



El bloque Scrambler entrevera la señal de entrada, que debe ser un escalar o un vector columna basado en tramas. Si el parámetro base de cálculo es N, entonces los valores de entrada deben ser enteros entre 0 y N-1.

Debajo se encuentra un esquema del scrambler. Todos los sumadores realizan sumas módulo N.

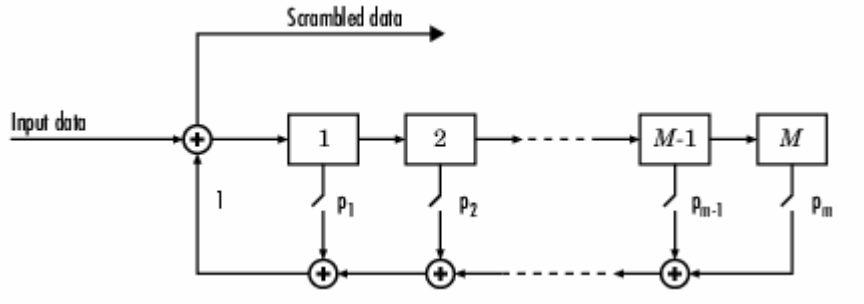


Figura 5.3: Circuito del scrambler

En cada paso temporal, la entrada causa que el contenido de los registros se desplace secuencialmente. Cada interruptor del scambler está encendido o apagado como se define en el polinomio de scramble que se introduce como parámetro.

El parámetro estados iniciales lista los estados de los registros del scrambler cuando la simulación comienza. Los elementos de este vector deben ser enteros entre 0 y N-1

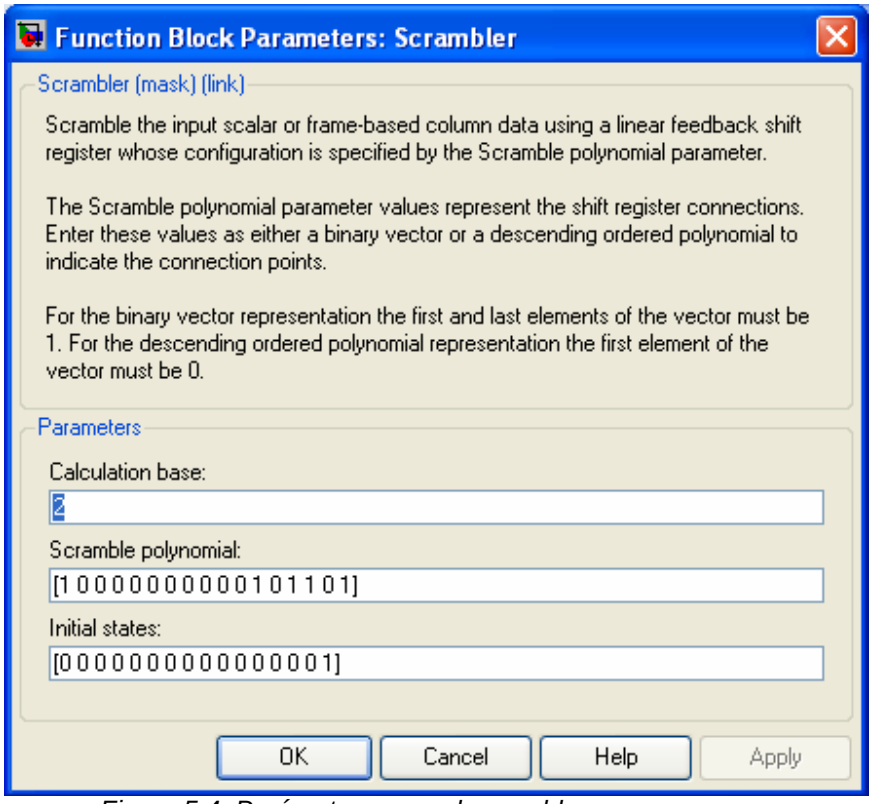


Figura 5.4: Parámetros para el scrambler

La base de cálculo restringe los valores de entrada y salida al rango 0, N-1, por lo que la base de cálculo es N=2. El polinomio de scramble y los estados iniciales surgen de la descripción en la norma.

5.3.2 Descrambler (Rx)

El bloque descrambler hace el proceso inverso al scrambler. Si se usa el bloque Scrambler en el transmisor, entonces se debe usar el bloque Descrambler en el receptor. La señal de entrada debe ser escalar o un vector columna basado en trama.

En la figura se muestra un esquema del descrambler. Todos los sumadores y la sustracción son operaciones módulo N, donde N es el parámetro base de cálculo. Los valores de entrada deben ser enteros entre 0 y N-1.

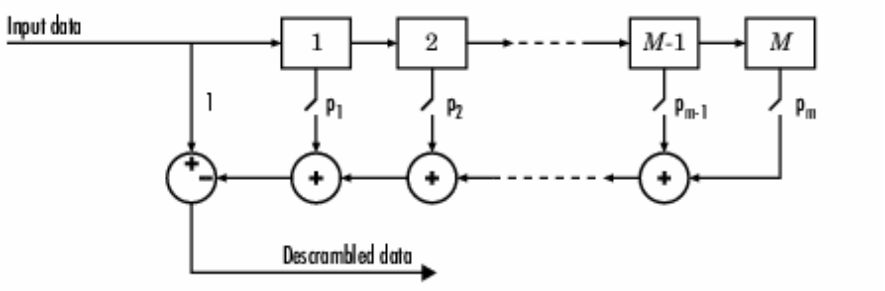


Figura 5.5: Circuito del descrambler

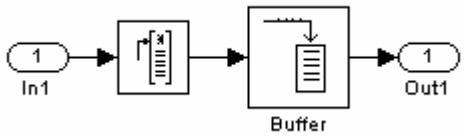
En cada paso temporal, la entrada causa que los contenidos de los registros se desplacen secuencialmente. Cada interruptor en el descrambler está encendido o apagado como lo define el parámetro polinomio de scramble. Para hacer que el bloque Descrambler sea el inverso del Scrambler, se debe usar el mismo polinomio en ambos bloques. Si no hay retraso de señal entre el scrambler y el descrambler, entonces los estados iniciales en los dos bloques deben ser los mismos.

Se usan los mismo parámetros que en su inverso.

5.4 Segmentación

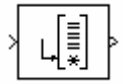
5.4.1 Segmentación (Tx)

Se resolvió utilizando el bloque Pad, que agrega bits de relleno a la entrada y luego un buffer que va agrupando en los bloques de código para así completar los 6 bloques de 4663 bits.



.

Pad



Se especifica que se va a rellenar con ceros al principio de la señal y que saldrán 27.978 bits (entran 27.976), lo que indica el relleno con dos bits de valor cero.

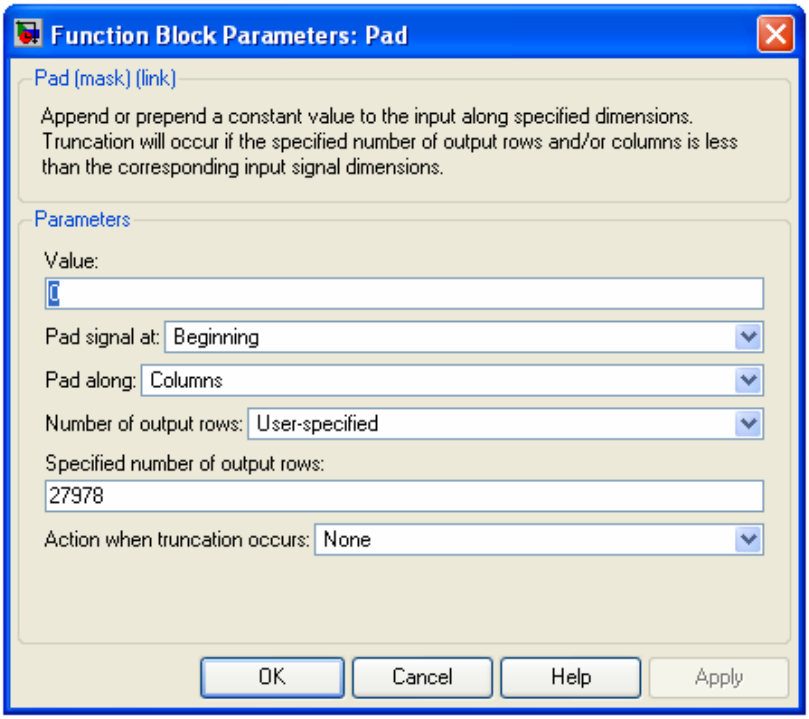


Figura 5.7: Parámetros para el bloque Pad

5.4.2 Concatenación (Rx)

El bloque inverso consistirá en un buffer que agrupa 27.978 bits y luego un bloque Puncture para quitar los dos bits de relleno.

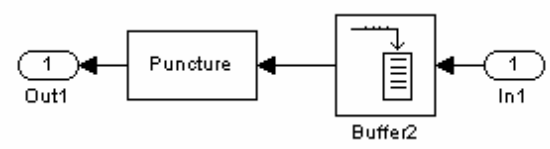


Figura 5.8: Concatenación

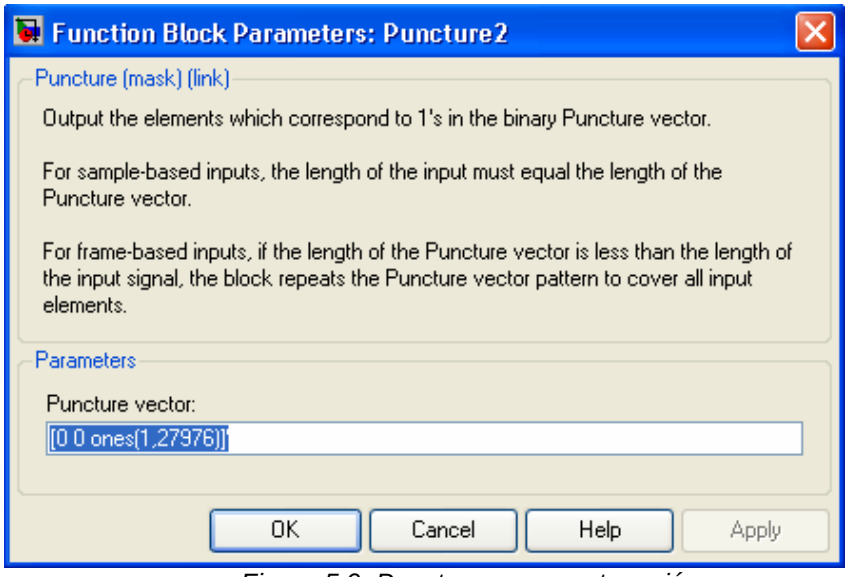


Figura 5.9: Puncture en concatenación

5.5 Codificación

5.5.1 Codificador Turbo (Tx)

Para realizar el codificador turbo, se utilizaron bloques de codificación convolucional y un intercalador (Select Rows).

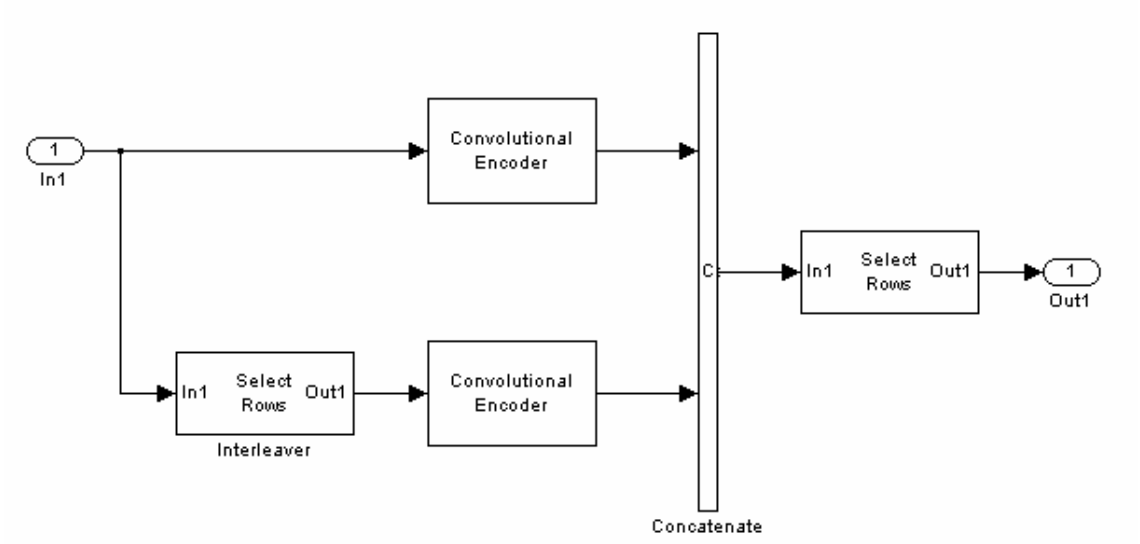
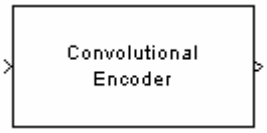


Figura 5.10: Diagrama Codificador Turbo

5.5.1.1 Codificador Convolucional:



Este bloque codifica una secuencia de vector de entrada binario para producir una secuencia de vectores binarios de salida.

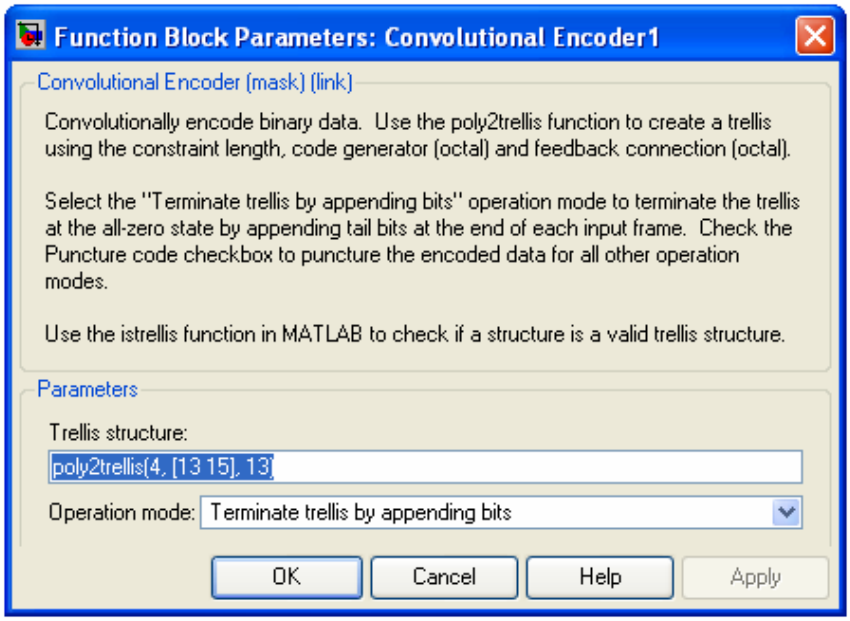


Figura 5.11: Codificador convolucional

Para definir el codificador convolucional, se usa el parámetro *estructura de Trellis.* Se puede especificar el codificador usando su longitud de restricción, polinomios generadores y los polinomios de conexión de retroalimentación ingresándolos como parámetros de la función de matlab *poly2trellis*, detallada en el Anexo D.2.1. Con este parámetro se indica la estructura del codificador convolucional para que sea la especificada por la norma, figura 3.5

El Modo de operación se elige de forma que se agreguen los bits de trellis al final de cada trama de entrada, terminando en el estado todo ceros.

5.5.1.2 Interleaver

Esta operación se implementó con el bloque de Simulink, Select Rows. El mismo reordena las filas o columnas de entrada de acuerdo a un vector que le indica esta el orden de salida.

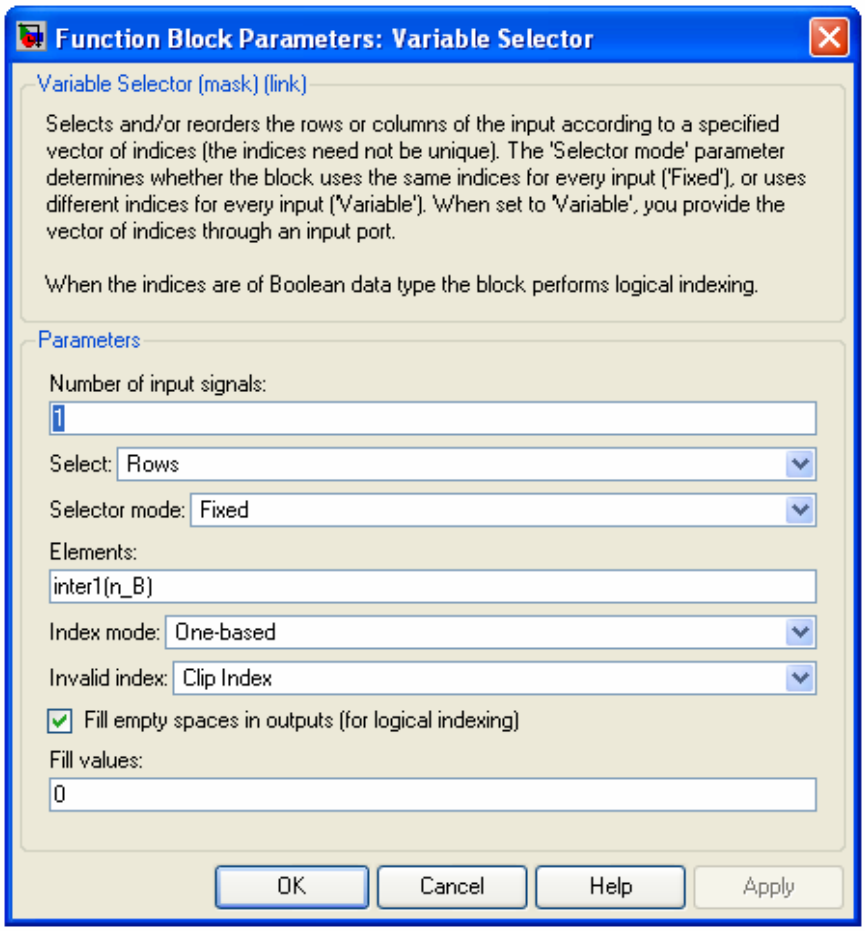


Figura 5.12: Intercalador de codificador Turbo

El vector inter1(n_B) es una función implementada en Matlab que aplica el algoritmo de intercalado de codificación Turbo, Esta función se encuentra en el Anexo, dónde se encuentran todos los códigos implementados.

Los flujos de bits provenientes de cada uno de los codificadores convolucionales, se unen mediante el bloque *vector concatenate*, que simplemente concatena las entradas:

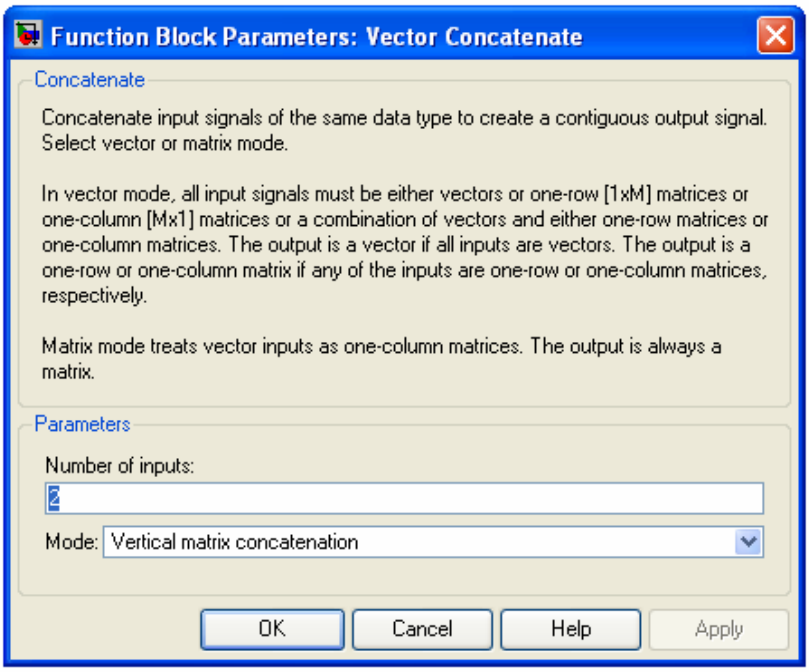


Figura 5.13: Concatenación de los flujos

La salida del codificador Turbo es alcanzada finalmente luego de pasar por otro bloque *Select Rows*. Este especifica la salida correcta del codificador Turbo, seleccionando las filas correspondientes según el algoritmo. Para esto se utiliza la función Matlab *salturbo(n_B)*, que indica los elementos que deben constituir la salida siguiendo la descripción de la norma 25.212.

5.5.2 Decodificador (Rx)

El decodificador Turbo se implementó de esta forma:

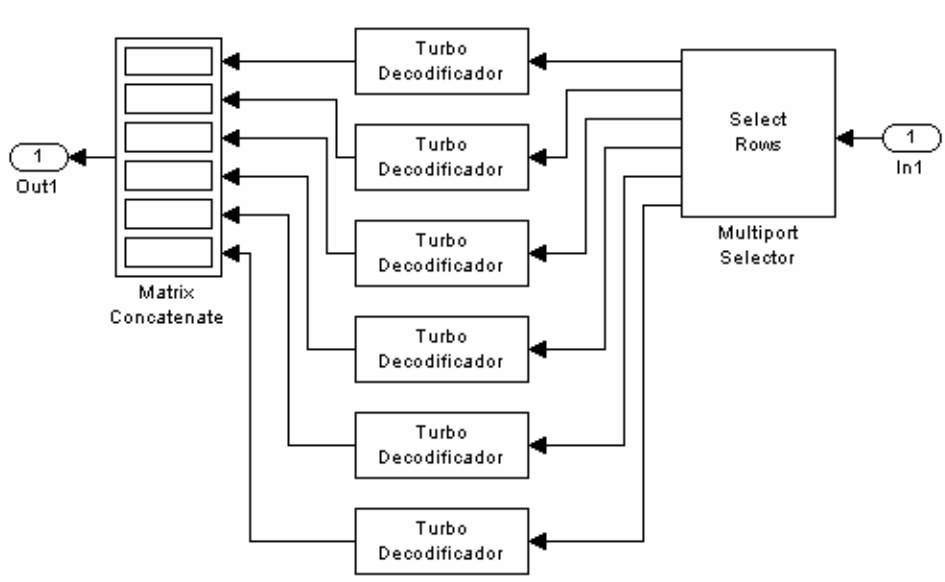


Figura 5.14: Diagrama sistema Decodificador turbo

El bloque *Multiport Selector* a la entrada del bloque se ocupa de separar los seis bloques de código que vienen en la trama de entrada. Estos bloques llegan en serie por lo que para separarlos simplemente se sacan en orden separando en bloques iguales. Recordamos que la codificación se aplica por bloque de código, obtenidos luego de la segmentación que convierte la trama en seis bloques de código, y no sobre la trama entera. De la misma manera la decodificación debe aplicarse sobre estos bloques de código, que son las salidas del *Multiport Selector.*

Cada uno de los seis decodificadores Turbo tiene el siguiente diagrama:

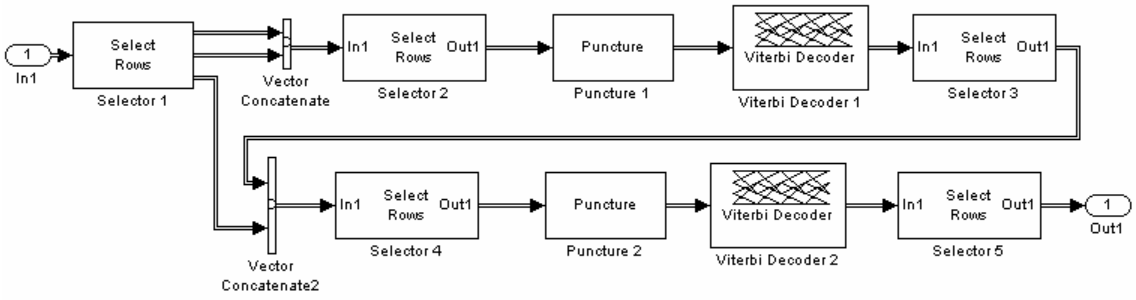
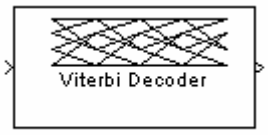


Figura 5.15: Diagrama Decodificador turbo para bloque de código

Como se puede ver se utiliza el bloque Viterbi Decoder de la librería de Simulink.



Este bloque se utiliza para decodificar una señal que fue codificada con un codificador convolucional, utilizando el algoritmo de Viterbi. Permite introducir como parámetro un vector indicando si se produjo borrado de bits en el transmisor, como sucede en nuestro caso en el bloque H-ARQ.

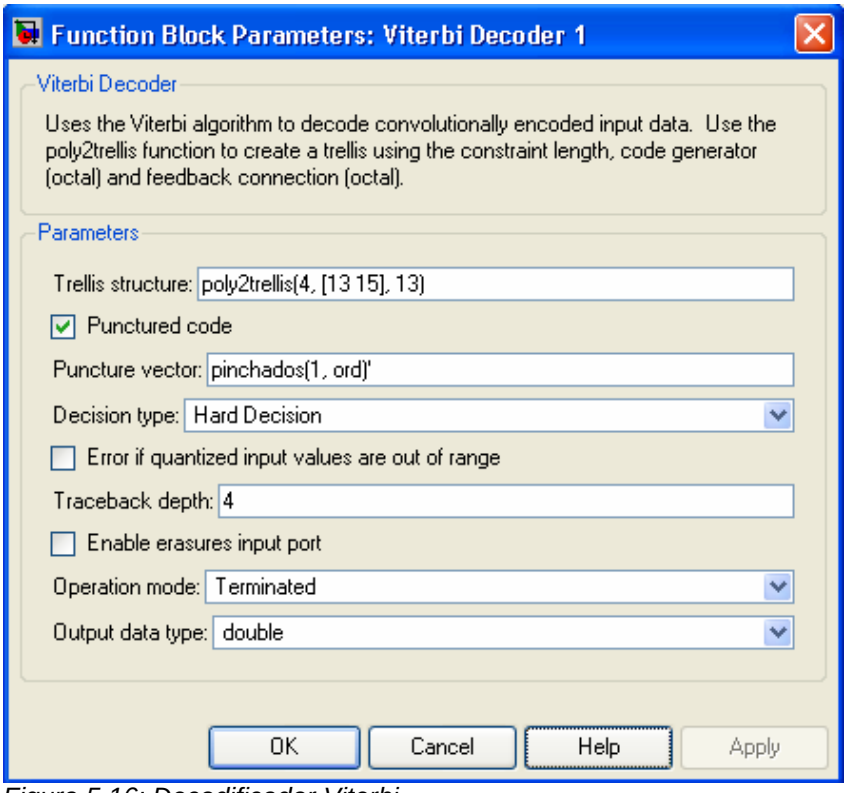


Figura 5.16: Decodificador Viterbi

Trellis structure indica la estructura del codificador convolucional, que debe ser la misma que se utiliza en el transmisor. Se indica que el código fue borrado (Rate Matching), y se especifica el patrón de borrado con el vector *pinchados,* función implementada en Matlab, donde x = 1 para el Puncture 1 y x = 2 para el Puncture 2, y ord es el número de bloque de código (o *Turbo Decodificador*)en el que se está trabajando.

La profundidad del algoritmo de Viterbi, es el número de muestras que tiene que memorizar para luego aplicarle la función de máxima verosimilitud, y se necesitan cierta cantidad de muestras para que esta función llegue a la secuencia generadora más probable. Este valor viene dado por la cantidad de estados del codificador convolucional sobre dos. Los estados del codificador son 2^k, siendo k la cantidad de registros en el codificador.

En nuestro caso, la cantidad de registros del codificador son 3 (figura 3.5), entonces los estados son 8. Por lo tanto, de aquí se deduce que la profundidad del Viterbi, en nuestro caso, es igual a 4.

Operation Mode se detemina a partir del codificador que se utiliza en el transmisor. El modo *Terminated* aplica al codificador convolucional retroalimentado.

Como se ve en la figura, 5.14 , cada decodificador tiene dos ramas similares, cada una de las cuales contiene un *Viterbi Decoder*. A la entrada al sistema, la señal consiste de bits sistemáticos, bits de paridad uno, bits de paridad dos y los bits de Trellis, que se crearon en el codificador Turbo. El funcionamiento de este sistema se basa en el conocimiento de la señal de entrada y en el *Viterbi*

Decoder como inverso del codificador convolucional. En definitiva se trata de decodificar la señal con el Viterbi Decoder en cada rama, utilizando los bits de paridad uno en la primer rama y los de paridad dos en la segunda rama. Adicionalmente, en la segunda rama la entrada de bits sistemáticos será la señal decodificada en la primer rama. De esta manera se utiliza la redundancia de paridad dos sobre la señal obtenida utilizando la redundancia de paridad uno.

Los bloques *Select Rows* se encargan de ordenar el flujo de bits que ingresa en cada elemento del decodificador. Se implementaron funciones de Matlab para crear los vectores que se ingresan como parámetro a cada Selector para indicarle qué bits debe colocar a la salida. En la tabla 5.1 se detalla qué función se utiliza en cada Selector y la acción que realiza, así como también en los bloques Puncture.

Bloque	**Función Matlab**	**Acción**
Selector 1	[demux 2 demux2 demux3]	Separa en tres flujos de bits: sistemáticos, paridad uno, paridad dos
Selector 2	intermedioDC(4663)	Entrelaza los bits sistemáticos y los de paridad uno + trellis, tal como los espera el Viterbi
Selector 3	inter1(4663)	Aplica el algoritmo de intercalado del codificador Turbo
Selector 4	demux(4663)	Selecciona sólo los bits sistemáticos a la salida de del convolucional
Selector 5	intermedioDC(4663)	Entrelaza los bits sistemáticos y los de paridad dos + trellis, tal como los espera el Viterbi
Selector 6	de_inter1(4663)	Aplica el inverso del algoritmo de intercalado del codificador Turbo
Puncture 1	Pinchados (1,x)	Aplica el patrón de borrado del Rate Matching para los bits sistemáticos y de paridad uno del bloque de código x
Puncture 2	Pinchados (2,x)	Aplica el patrón de borrado del Rate Matching para los bits sistemáticos y de paridad dos del bloque de código x

Tabla 5.1:Funciones del bloque decodificador

Por lo tanto, a modo de resumen, cada bloque *Turbo Decodificador* funciona de la siguiente forma:

- El Selector 1 separa en los tres flujos de bits sistemáticos, paridad uno y paridad dos mas los de trellis.
- Se concatenan los bits sistemáticos y los de paridad uno.
- El Selector 2 entrelaza bits sistemáticos y de paridad uno mas trellis para introducirlos en el Viterbi
- El bloque Puncture, borra los bits que indica el patrón de Rate Matching para ese bloque de código
- El Viterbi Decoder decodifica la señal

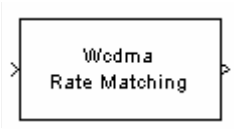
- El Selector 3 aplica el algoritmo de intercalado a la señal tal como se hizo en el codificador Turbo, para obtener los bits sistemáticos que se ingresaron al convolucional 2 en el transmisor.
- La señal se entrelaza con el flujo de bits de paridad dos más trellis
- El bloque Puncture, borra los bits que indica el patrón de Rate Matching para ese bloque de código
- El Viterbi Decoder decodifica la señal
- El Selector 5 ordena la señal aplicando el algoritmo inverso del intercalador Turbo

Por último, la salida de cada uno de los seis bloques *Turbo Decodificador* se concatena para armar la trama de salida del sistema Decodificador.

5.6 H-ARQ

5.6.1 H-ARQ (Tx)

Se implementó utilizando el bloque *Wcdma Rate Matching,* incluido en un demo de Matlab



Para cerciorarnos de que este bloque fuera útil para HSDPA se estudiaron los siguientes archivos:

- swcdma_ratematching.c
- wcdma_ratematchinginit.m

El primer archivo implementa los métodos necesarios para la operación de Rate Matching, dentro de ellos, el algoritmo de pinchado de la norma. El segundo es un archivo de inicialización, y debió ser modificado para adaptarlo a nuestro caso de estudio. En el Anexo adjunta el archivo original y el modificado.

Este bloque se encarga de calcular el patrón de pinchado mediante el algoritmo especificado en la norma**,** y de llevarlo a cabo descartando los bits correspondientes según este resultado.

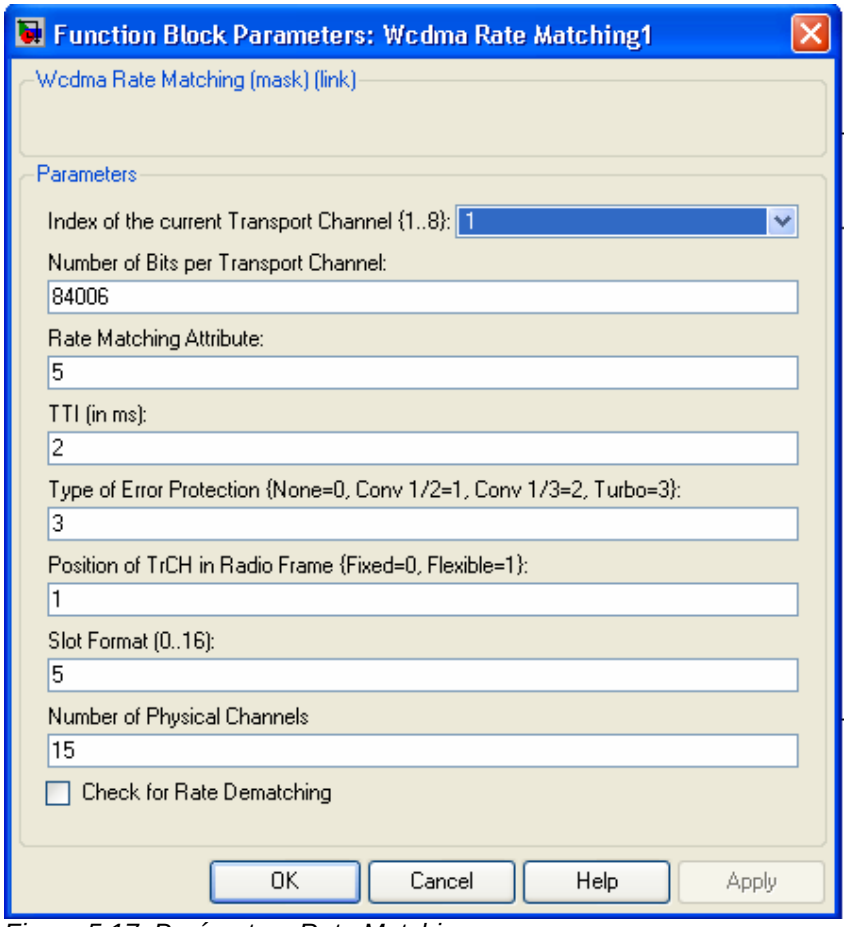


Figura 5.17: Parámetros Rate Matching

En primer lugar se introduce el índice del canal de transporte que elegimos como 1. La cantidad de bits del canal de transporte, en este caso son los que entran al bloque. Se verificó que el atributo de rate matching y el slot format no afectan la operación del bloque para este caso. Luego se especifica el TTI (2ms), la codificación Turbo, la posición flexible y el número de canales físicos igual a 15

5.6.2 Inverso HARQ (RX)

Para el receptor, se utiliza el mismo bloque marcando al check box en la caja de diálogo señalando que se trata del inverso (Rate Dematching). La operación es similar; calcula el patrón de pinchados según el algoritmo en la norma, pero en este caso rellena la señal con los bits que fueron descartados en el transmisor.

5.7 Segmentación de Canales Físicos

5.7.1 Segmentación de canales físicos (TX)

En primer lugar se implementó con un demultiplexor que separaba el flujo de bits a la entrada en los quince canales físicos correspondientes.

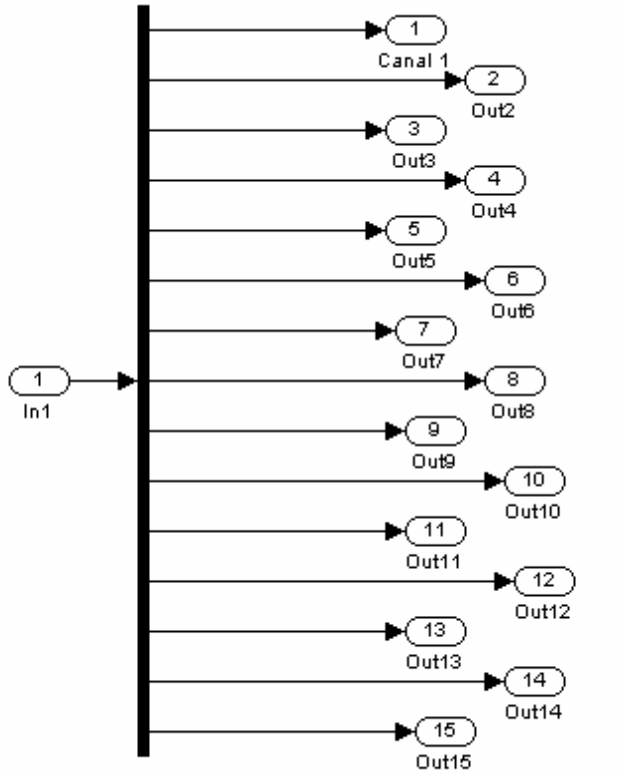


Figura 5.18: Diagrama primera implementación segmentación de canales físicos

En este caso, los bloques siguientes debían implementarse para cada uno de los quince canales, repitiendo cada bloque quince veces. Esta implementación, repitiendo bloques en paralelo llevó a que el tiempo de la simulación fuera bastante elevado.

Posteriormente, en una etapa de optimización del diseño, se mejoró la implementación. Se buscó por un lado ocupar menos espacio en pantalla que permitiera visualizar el sistema, y sobretodo disminuir el tiempo de simulación, que pasó de 7 horas a 2 horas aproximadamente.

Esto se logró descubriendo que Simulink trabaja mucho más rápido si en vez de trabajar en paralelo lo hacemos serialmente. De esta manera, se introduce simplemente un buffer que agrupa 1920 bits, tamaño de trama para cada canal físico, y se envían secuencialmente. De esta forma, los bloques siguientes se implementan sólo una vez y manejan los quince canales en serie. Con esta segunda y definitiva implementación, Simulink sólo necesita

inicializar un bloque en vez de quince, y lo mismo en el procesamiento que necesita, es para un bloque y no para quince.

5.7.2 Concatenación de canales físicos (Rx)

Aquí ocurrió lo mismo que se describió para la segmentación en el transmisor, se utilizó un multiplexor (en la primera implementación) para juntar los 15 canales pero posteriormente se optó por hacerlo de otro modo. Se puso nuevamente un buffer, esta vez es de 1.920*15=28.800 bits.

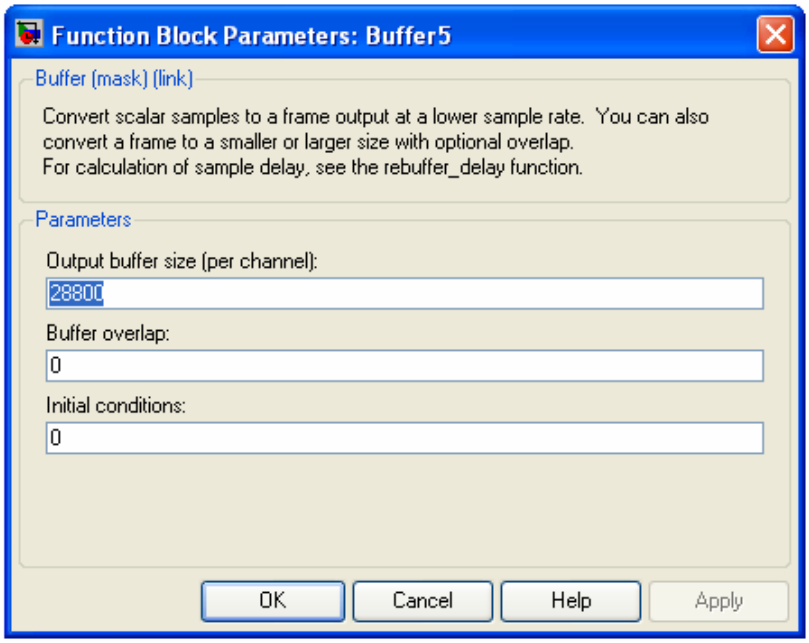


Figura 5.19: Buffer para Concatenación de canales físicos

Esta fue la primera implementación:

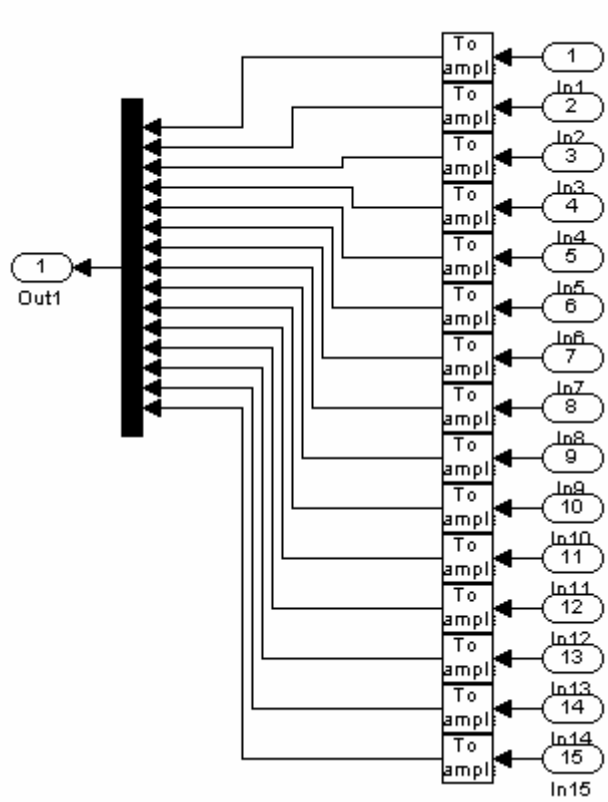


Figura 5.20: Diagrama primera implementación segmentación de canales físicos

Pero al optimizar el sistema se verificó que no era necesario hacer los quince canales en paralelo, sino que llegan las tramas serialmente, por lo que al poner el buffer se agrupan en una sola trama y se logra la concatenación.

5.8 Interleaving

5.8.1 Interleaving (Tx)

El intercalado HS-DSCH consiste en dos intercaladores en paralelo. Esto se realizó de la siguiente manera:

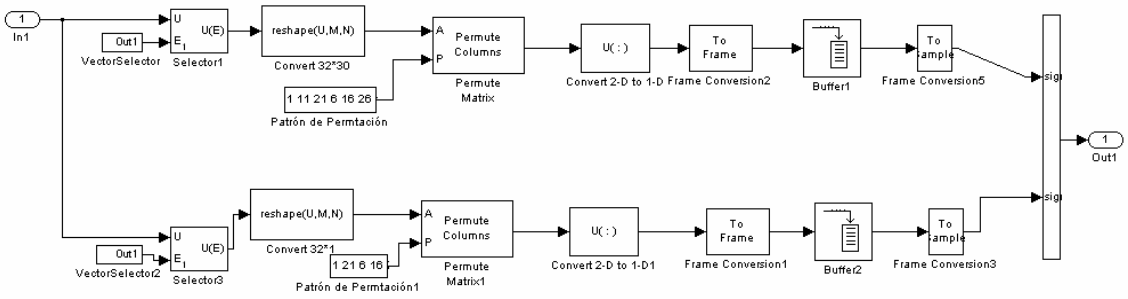


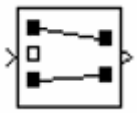
Figura 5.21: Diagrama intercalado

Se colocan en primer lugar dos *selectores*, para tomar dos bits en la rama superior y dos en la inferior sucesivamente como indica la norma. Luego el bloque *reshape (M,N)* forma una matriz de dimensiones 32x30, a la que se le

aplica, con el bloque *Permute Columns*, el patrón de permutación que especifica la norma. La salida de estas matrices se transforma a un vector de longitud 2 con el buffer, para luego transformarlo en una trama que será entrelazada con la otra rama. De esta forma se logra el algoritmo que indica la norma.

A continuación se muestran los bloques más importantes y los parámetros utilizados en los mismos para lograr este sistema. Información mas detallada de los mismos se encuentra en el Anexo F.

Selector



El bloque selector genera una salida de elementos seleccionados de un vector o una matriz de entrada.

Ambos bloques selectores, el superior y el inferior, tienen los mismos parámetros. Reciben 1.920 bits. Lo que cambia es el vector externo que le indica qué elementos elegir para cada una de las ramas.

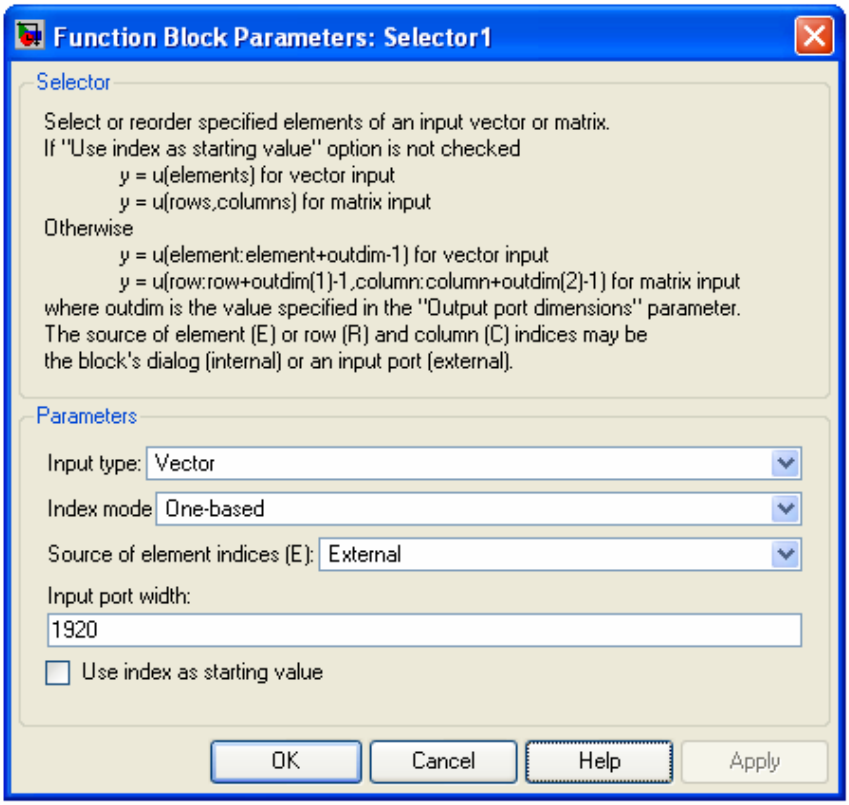
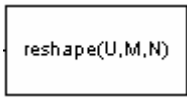


Figura 5.22: Selector 1

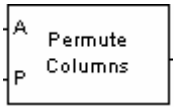
Covert 1D to 2D



Como su nombre lo indica este bloque transforma una señal de tipo vector en una matriz.

La norma especifica que los intercaladores tienen dimensiones 32x30 y es lo que se introduce como parámetro del bloque.

Permutation



Reordena las columnas o filas de una matriz de acuerdo a un vector de entrada que especifica el orden a la salida del bloque.

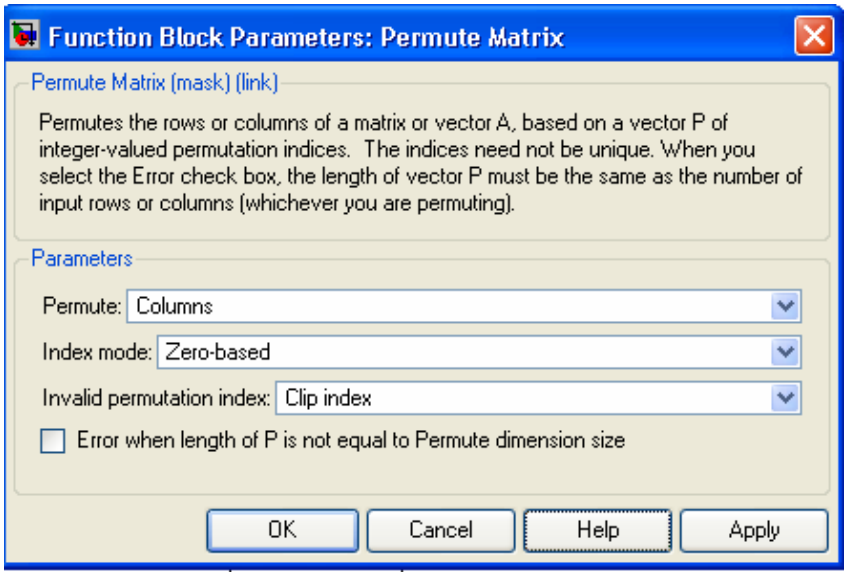


Figura 5.23: Permutación de la matriz

En este caso se permutan las columnas, y se introduce un vector P que tiene el patrón de permutación requerido.

P= [0 20 10 5 15 25 3 13 23 8 18 28 1 11 21 6 16 26 4 14 24 19 9 29 12 2 7 22 27 17]

5.8.2 Deinterleaving - Rx

Se realiza el proceso inverso. A la entrada se toman de a dos elementos con el demultiplexor. Se pone un buffer de 960 para luego transforma a una matriz de dimensiones 32*30 igual que en el transmisor. A esta matriz se le aplica el patrón de permutación inverso al anterior, P.

P = [0 12 25 6 18 3 15 26 9 22 2 13 24 7 19 4 16 29 10 21 1 14 27 8 20 5 17 28 11 23]

Luego solo resta convertirlo nuevamente en un vector, con el bloque Convert 2D to 1D y por último mezclar las dos ramas de dos en dos. Para esto se utiliza el buffer que agrupa de a dos y un multiplexor.

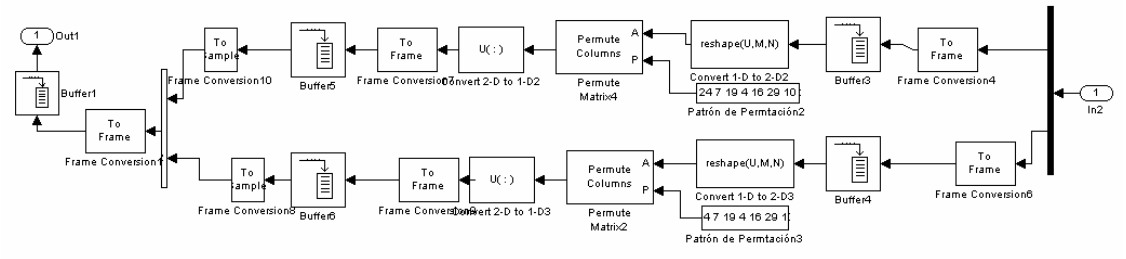


Figura 5.24: Diagrama Deinterleaving

5.9 Reordenamiento 16 QAM

5.9.1 Reordenamiento 16 QAM (TX)

Consiste en tomar los bits en grupos de 4 y dependiendo de un parámetro b, reordenarlos de alguna forma determinada. Esta forma está establecida en la siguiente tabla de la norma 25.212:

Parámetro de constelación *b*	**Secuencia de salida**	**Operación**
0	$v_{p,k}\, v_{p,k+1}\, v_{p,k+2}\, v_{p,k+3}$	Ninguna
1	$v_{p,k+2}\, v_{p,k+3}\, v_{p,k}\, v_{p,k+1}$	Intercambiar MSBs con LSBs
2	$v_{p,k}\, v_{p,k+1}\, \overline{v_{p,k+2}}\, \overline{v_{p,k+3}}$	Inversión de los valores lógicos de LSBs
3	$v_{p,k+2}\, v_{p,k+3}\, \overline{v_{p,k}}\, \overline{v_{p,k+1}}$	Intercambiar MSBs con LSBs e inversión de los valores lógicos de LSBs

Tabla 5.2: Operación Reordenamiento 16 QAM (TX)

Como se puede ver, cuando b es cero no ocurre ningún cambio. Cuando b es uno, se toman los dos primeros elementos y se intercambian con los dos últimos. Si b es igual a dos se invierten los valores lógicos de los dos últimos bits, y cuando b es igual a tres se invierten los valores lógicos de los dos primeros bits y se los intercambia con los dos últimos.

Para lograr esto se utilizó un bloque *case* como se ve en la figura:

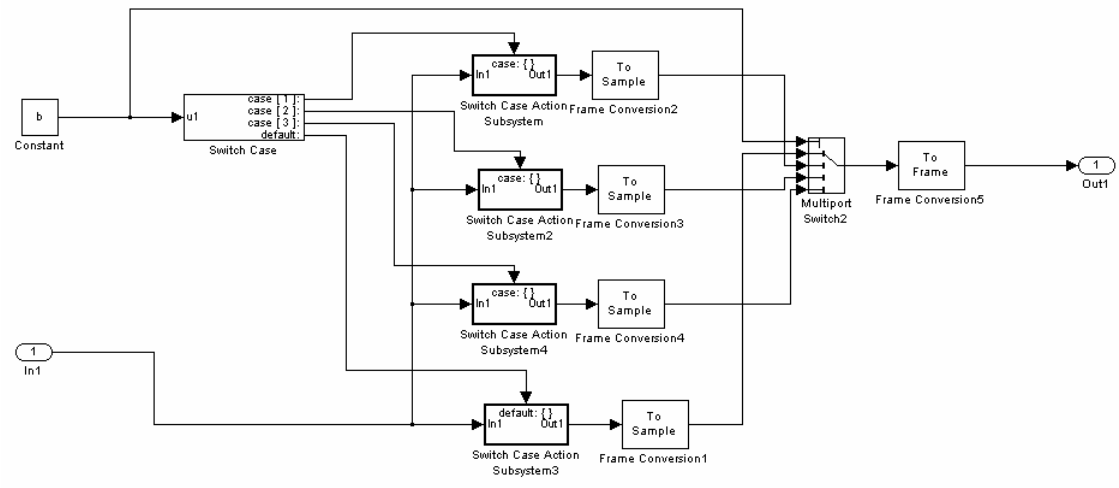


Figura 5.25: Diagrama Reordenamiento constelación 16QAM

Para b=0:

Como no hace nada, es sólo un cable como se ve en la siguiente figura:

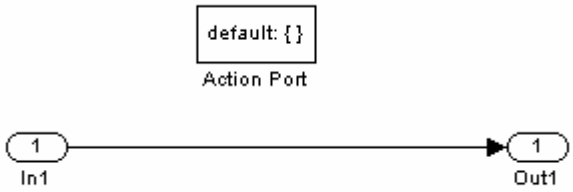


Figura 5.26: Diagrama para b=0

Para b=1:

Se intercambian los primeros dos bits con los dos últimos, para ello se utiliza un selector.

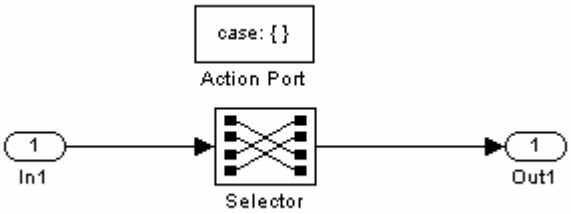


Figura 5.27 Diagrama para b=1

Los parámetros de este selector son:

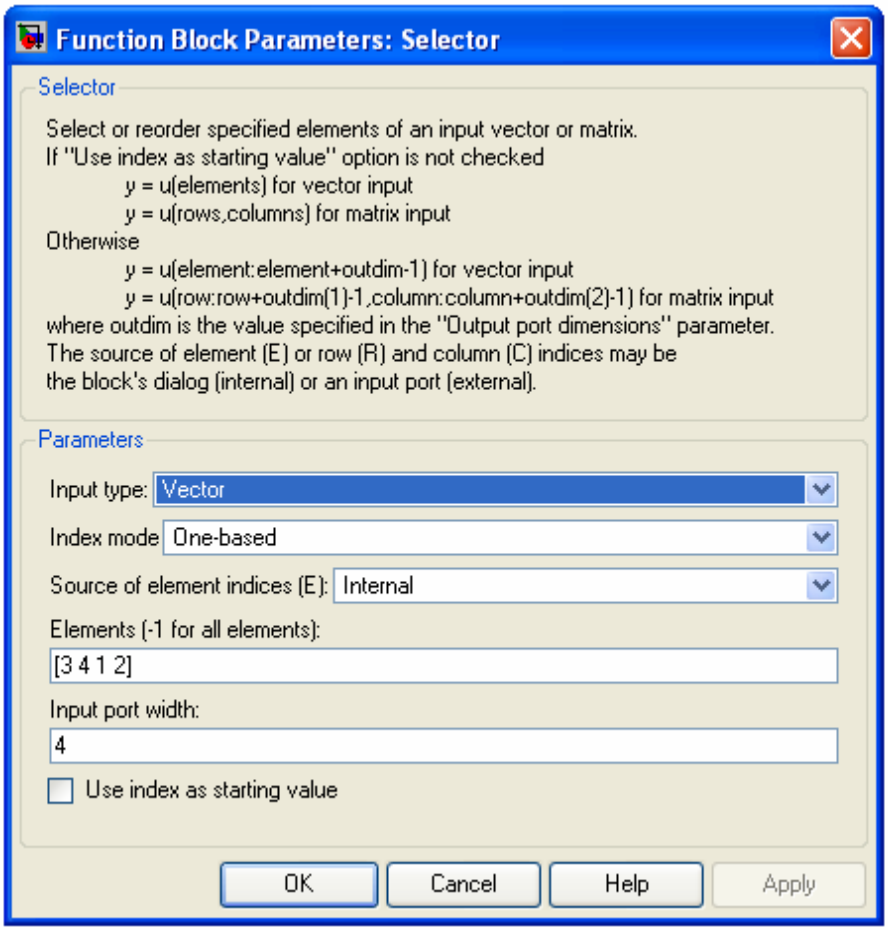


Figura 5.28: Selector para b=1

Para b=2:

Aquí sólo se invierten los valores lógicos de los últimos bits.

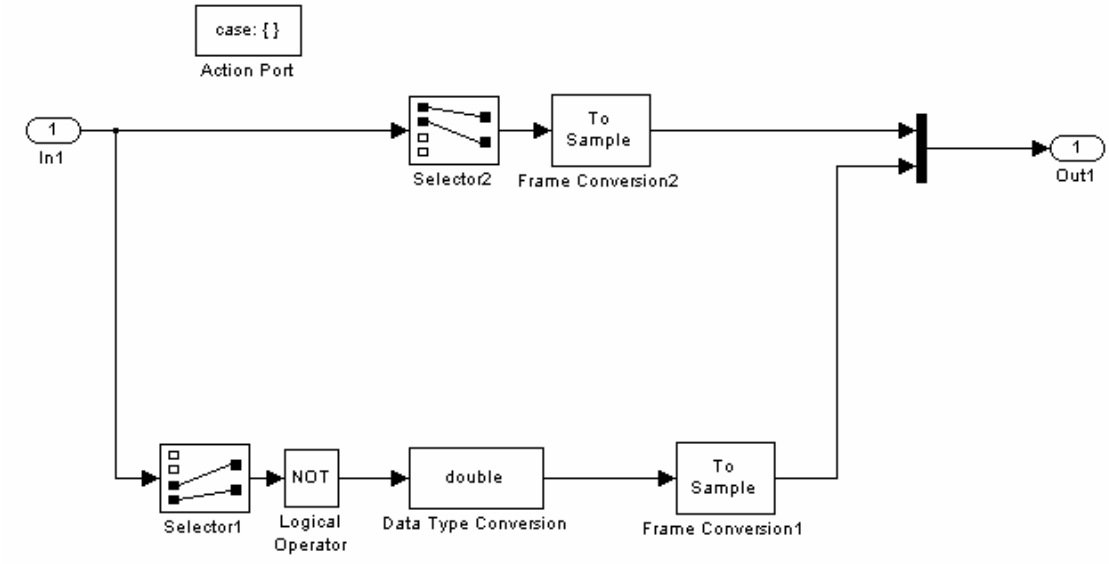


Figura 5.29: Diagrama para b=2

Para b=3:

Se invierte el valor lógico de los dos primeros bits y se los intercambia con los dos últimos.

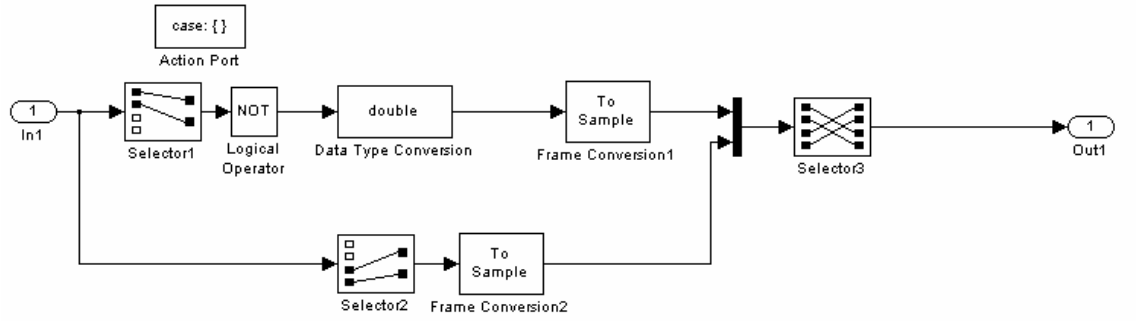


Figura 5.30: Diagrama para b=3

5.9.2 Reordenamiento 16 QAM - Rx

Se puede verificar que aplicando la misma operación que en el transmisor, salvo para b=3, se obtiene la misma secuencia que inicialmente. Por lo que el bloque inverso es igual salvo el caso de b=3.

En el transmisor, para b=0, la secuencia no cambia. Por lo tanto, si no se le hace nada en el receptor, se mantiene la secuencia original (asumiendo que llegan los bits correctos).

En el transmisor, para b=1, se intercambian los dos primeros bits con los dos últimos. Por lo tanto, al aplicar la misma operación, los bits vuelven a su lugar original.

Para b=2, se invierte el valor lógico de los dos últimos bits. Por lo tanto, si se le aplica la misma operación, vuelven a su valor inicial.

En el caso de b=3, se niegan los dos primeros bits y se los intercambia con los dos últimos. Si se vuelve a aplicar la misma operación, quedan todos los bits invertidos. Por lo tanto, aquí viene la variante. Se tiene que invertir el valor lógico de todos los bits.

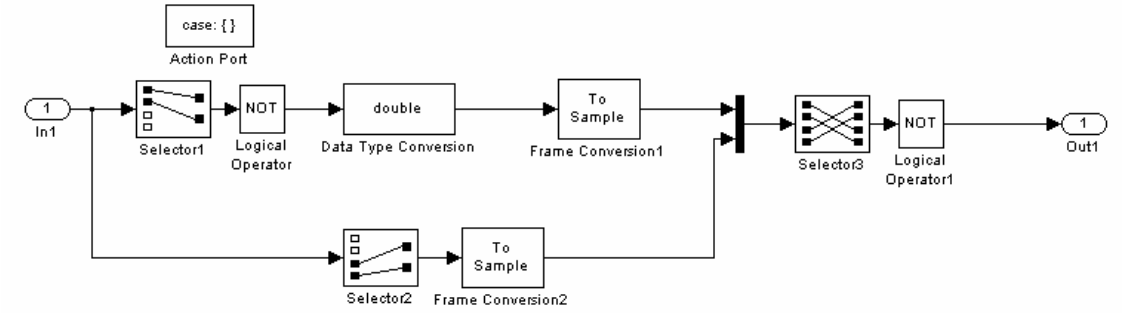


Figura 5.31:Reordenamiento 16 QAM Inverso para b=3

5.10 Spreading y Scrambler

Esta etapa descripta en la norma 25.213 consta de 3 sub-sistemas:

1) Data Mapper
2) Multiplicación por códigos OVSF (Spreading)
3) Scrambling

5.10.1 Data Mapper

El Data Mapper toma los bits de a cuatro y los mapea según la siguiente tabla:

$i_1q_1i_2q_2$	Rama I	Rama Q
0000	0.4472	0.4472
0001	0.4472	1.3416
0010	1.3416	0.4472
0011	1.3416	1.3416
0100	0.4472	-0.4472
0101	0.4472	-1.3416
0110	1.3416	-0.4472
0111	1.3416	-1.3416
1000	-0.4472	0.4472
1001	-0.4472	1.3416
1010	-1.3416	0.4472
1011	-1.3416	1.3416
1100	-0.4472	-0.4472
1101	-0.4472	-1.3416
1110	-1.3416	-0.4472
1111	-1.3416	-1.3416

Tabla 5.3: Símbolos 16QAM

Se puede ver que i1 y q1 dan el signo del símbolo que saldrá en cada rama: si i1 (q1) es cero, el número en la rama I (Q) será positivo y cuando es uno, será negativo. También se puede ver que i2 y q2 dan la magnitud del símbolo de cada rama, cuando i2 (q2) es cero, la magnitud de I (Q) es 0,4472 y cuando es igual a uno, la magnitud es 1,3416. Se verifica también que 1,3416=3*0,4472.

Esto llevó a la siguiente implementación:

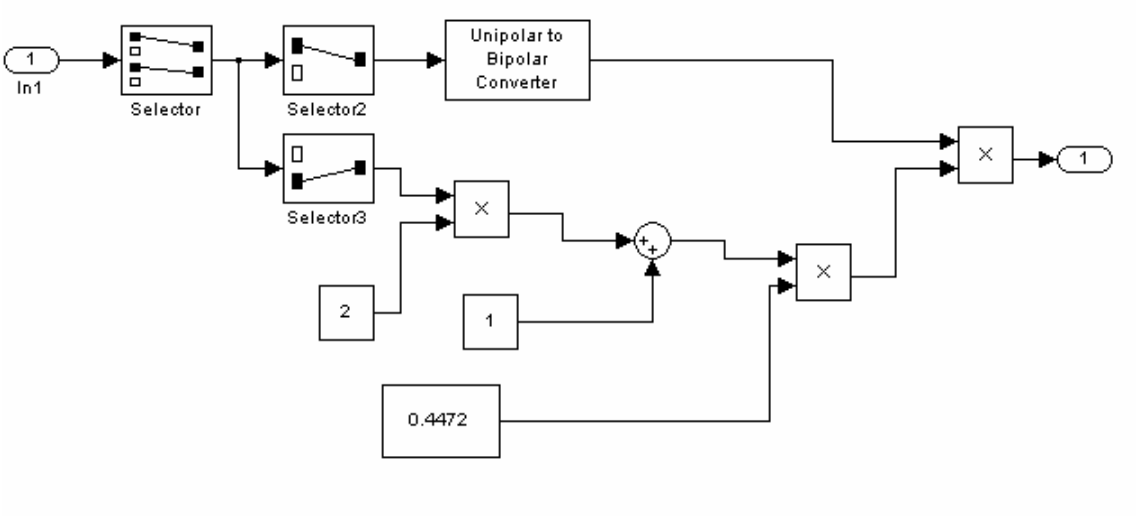


Figura 5.32: Diagrama Data Mapper

La entrada a este sistema deben ser los cuatro bits a convertir. Hay un bloque como el anterior para cada rama, el ejemplo es para la rama I, por lo que el primer Selector toma en este caso los bits i1 y i2. El primer bit, que define el signo, es convertido a notación bipolar por lo que un cero se convierte en uno y si el bit es uno se convierte en uno negativo.

Al segundo bit, que define el valor, se le aplica la operación ((bit*2)+1)*0.4472, para conseguir los valores deseados según el bit sea 0 o 1. Por último se multiplican estas dos subramas para terminar la operación.

5.10.2 Códigos OVSF (Spreading)

Aquí es donde tuvimos que separar en los 15 canales físicos, ya que cada canal tiene un código distinto. Se multiplica, por lo tanto, cada canal por uno de 15 códigos OVSF, códigos ortogonales:

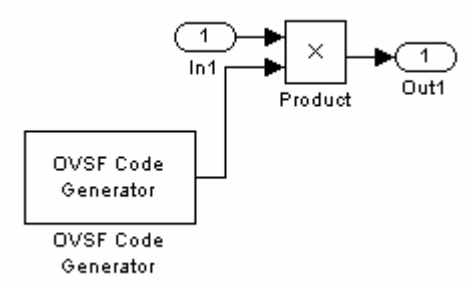


Figura 5.33: Diagrama de multiplicación por OVSF

OVSF Code Generator:

El bloque generador de códigos OVSF genera un código OVSF de un conjunto de códigos ortogonales. Los códigos OVSF fueron presentados por primera vez en sistemas de comunicación 3G. Los códigos OVSF son usados principalmente para preservar la ortogonalidad entre canales distintos en sistemas de comunicación.

El spreading factor es 16 y el índice varía para cada canal físico de 0 a 15 según el canal. Sample time es la tasa de chips en este bloque. Se utilizan tramas de 256 para poder combinar luego los canales de datos, con SF=16, y el CPICH con SF = 256.

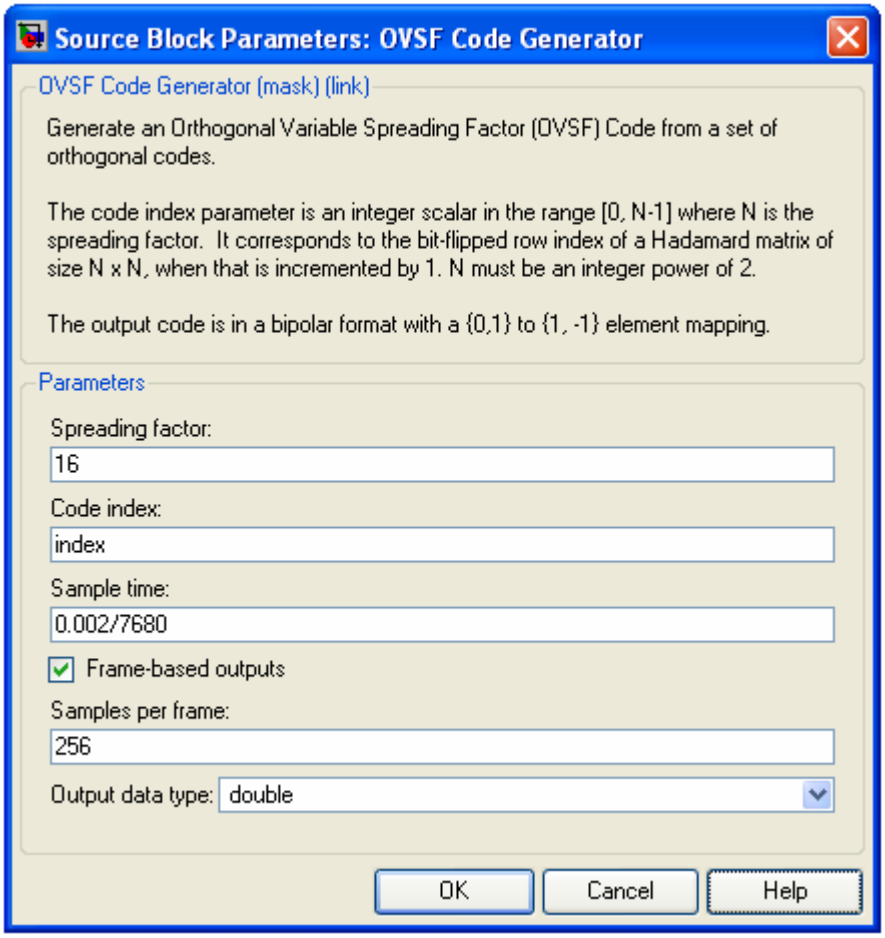


Figura 5.34: Parámetros para el generador de códigos OVSF

5.10.3 Scrambling

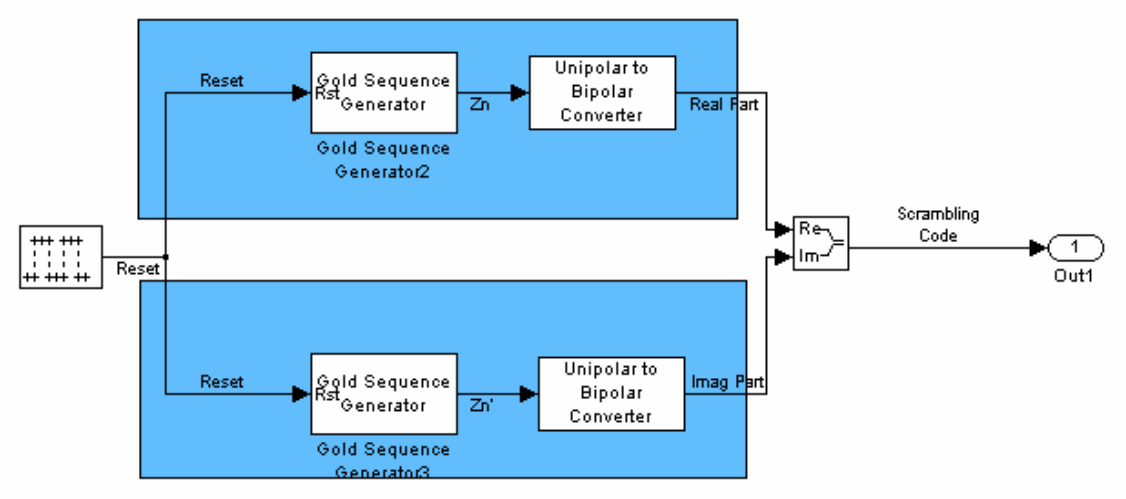


Figura 5.35: Diagrama Scrambling

Cada uno de los Gold Sequence Generator genera la secuencia gold especificada en la norma. Cuentan con un puerto de reset que hace que esta secuencia se repita cada 10 ms como establece la norma.

Parámetros utilizados:

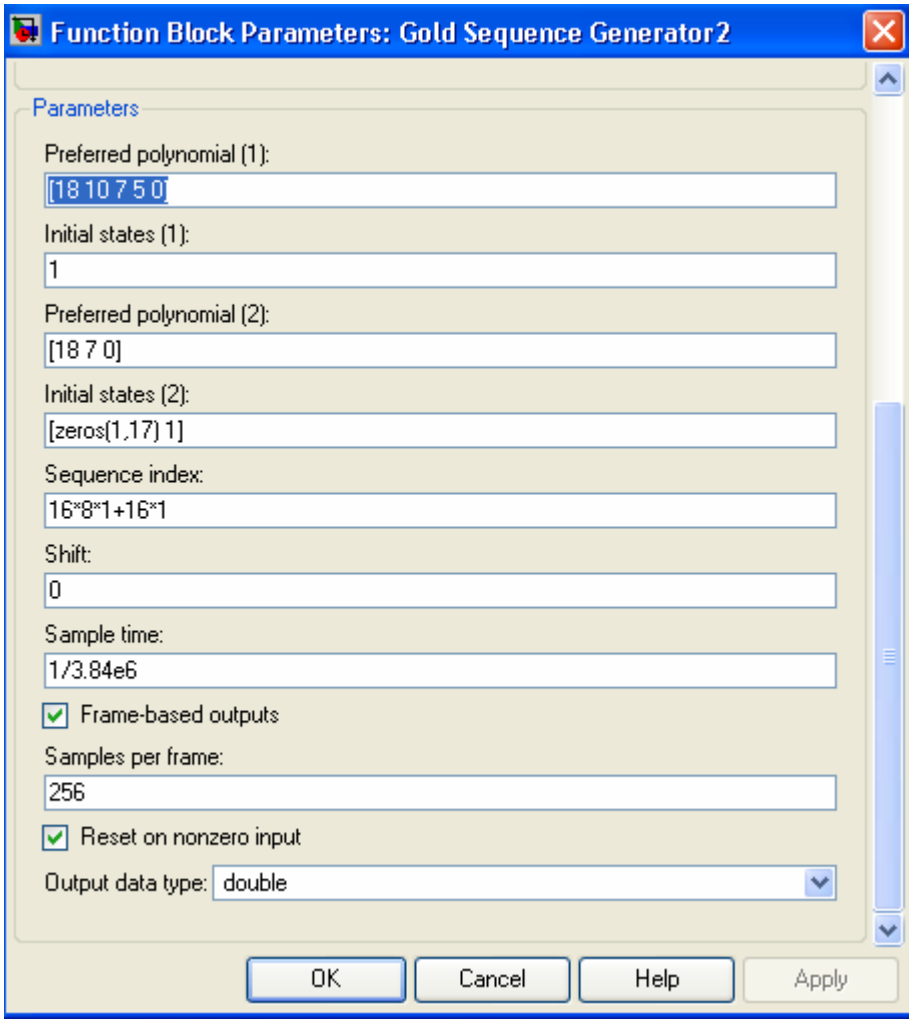


Figura 5.36: Generador de secuencia gold para el código de scrabmbling.

Preferred polynomial son los polinomios generadores a partir de los cuales se genera la secuencia, y se deben agregar los estados iniciales de cada uno.
Sequence index: entero que indica el índice de la secuencia de salida
Shift: Determina el offset de la secuencia desde el tiempo inicial. Se definió la secuencia que luego será la real con shift 0, y la otra con shift 131 según especificación.

El siguiente es el diagrama de todo el sistema, OVSF más scrambling, implementado en el proyecto.

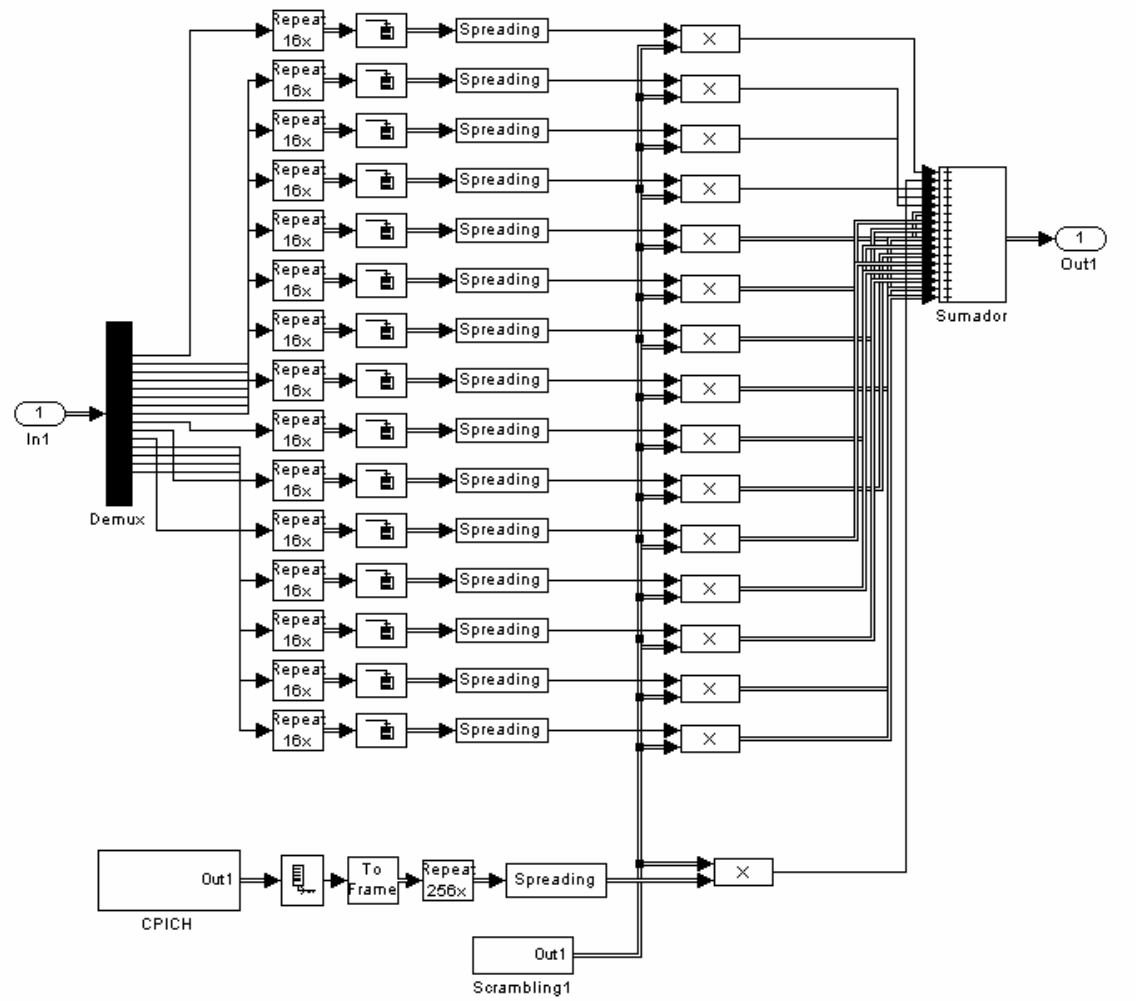


Figura 5.37: Sistema OVSF más scrambling

Entra el flujo de bits de los quince canales de datos serialmente, por esa razón se incluye un demultiplexor para separar estos canales. La trama de entrada se demultiplexa formando las tramas de 480 bits de cada canal. Luego se incluyen un repetidor y un buffer para preparar la señal tal como la espera el bloqe spreading para multiplicarla por su código OVSF.

Asimismo en paralelo se genera la señal de referencia del canal CPICH, la misma también se pasa por un bloque spreading pero en este caso con SF=256. Luego cada canal se multiplica también por el código de Scrambling para finalmente pasar al sumador donde se convierte en un sola señal con la información de los 16 canales mezclada.

5.10.4 Despreading y Descrambling

5.10.4.1 Demapper

Se utiliza una táctica similar al transmisor, el signo del símbolo recibido dará los bits i1 y q1 y la magnitud dará los bits i2 y q2.

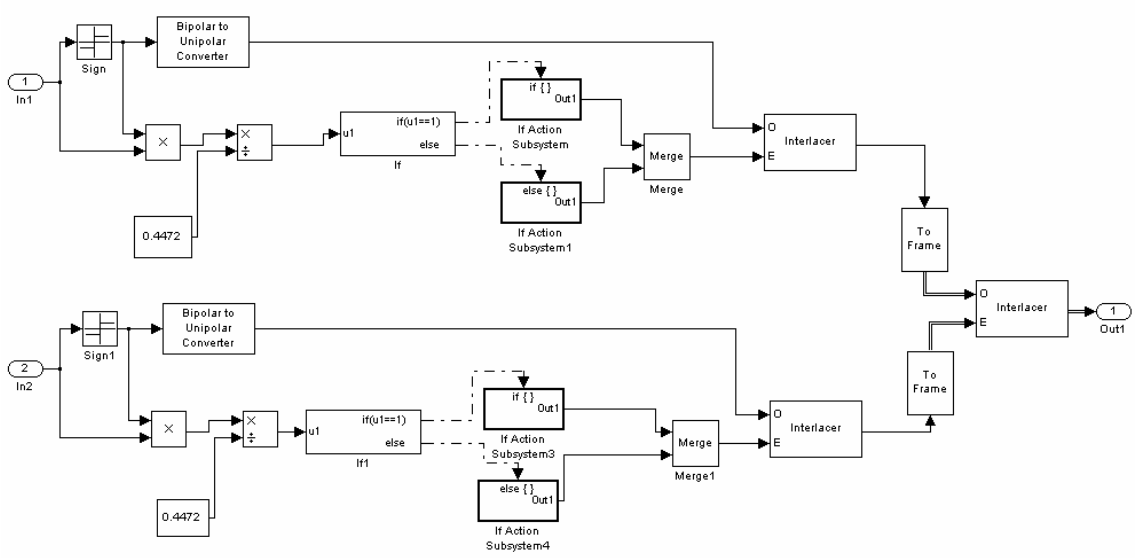


Figura 5.38: Diagrama inverso Data Mapper

En este caso se pone un bloque if para definir la magnitud, si la división del símbolo recibido entre 0,4472 es 1, el bit es un cero, si da 3 es un uno. En caso de que no sea ninguno de estos valores significa que hay un error, pero se eligió que en este caso se asignara un valor igual a 1.

5.10.4.2 Descrambling

Simplemente se genera el inverso del código de scrambling para recuperar la señal original al multiplicarla por este nuevo código.

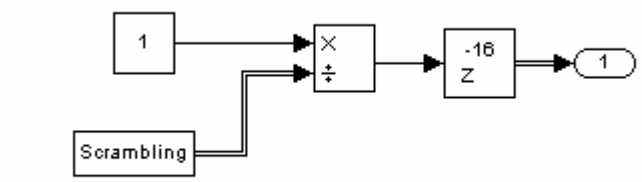


Figura 5.39: Diagrama de código de Descrambling.

Posteriormente este código se multiplica por la señal:

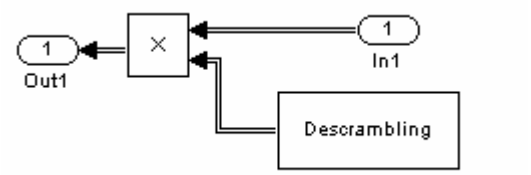


Figura 5.40: Diagrama Descrambling.

Se incluye um delay de -16 para sincronizar el código con la señal recibida, que debido al group delay definido en los filtros de modulación se retraza 16 muestras.

5.10.4.3 Despreading

El bloque Desprading se presenta en la figura x.x

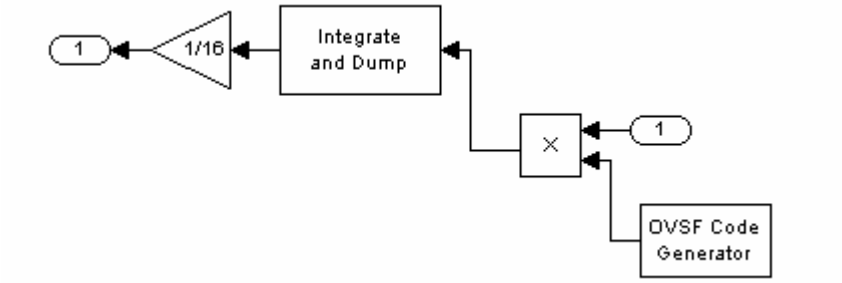


Figura 5.41: Diagrama bloque Desprading para un canal.

En primer lugar la señal de entrada es multiplicada por el código OVSF correspondiente para ese canal. En este punto la señal toma valores de los símbolos 16QAM pero todavía resta pasarla por el bloque *Integrate and Dump*, que integra las muestras de entrada en un período de 16. Finalmete se divide por 16 para obtener el símbolo 16QAM correspondiente a los 16 chips que entraron al bloque.

La figura 5.42 muestra el diagrama completo para todos los canales. Como se puede ver, hay quince bloques *Despreading* como el anterior, cada uno tendrá su código OVSF. Las quince señales obtenidas en cada canal luego del bloque *Despreading* se concatenan en el bloque *Matriz Concatenate.* A la salida se puede ver un *Convertidor 2D a 1D,* un converidor a trama y un unbuffer que simplemente ponen la señal en condiciones para entrar en el bloque siguiente.

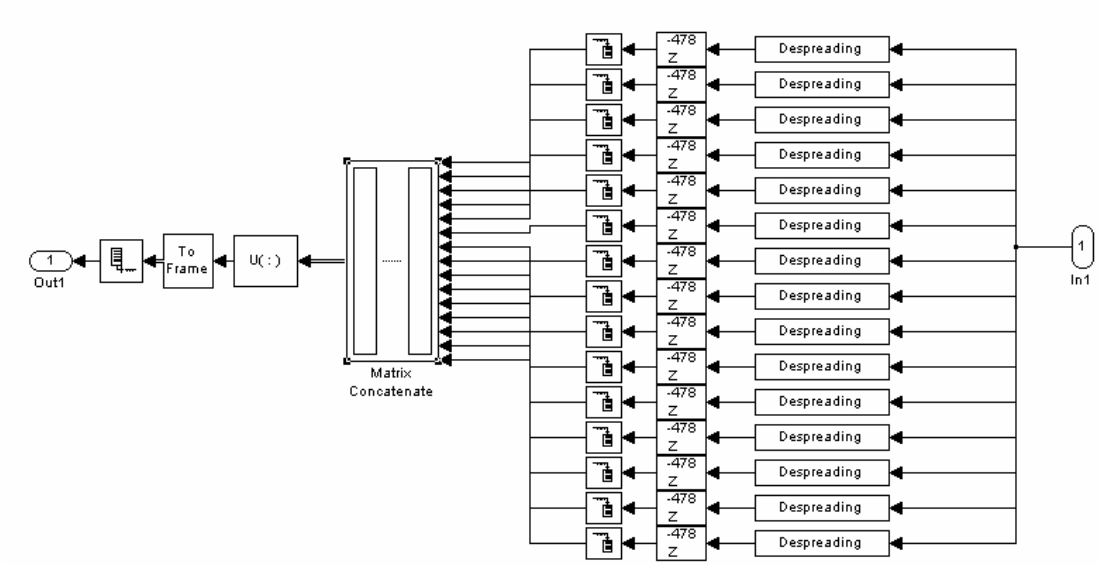


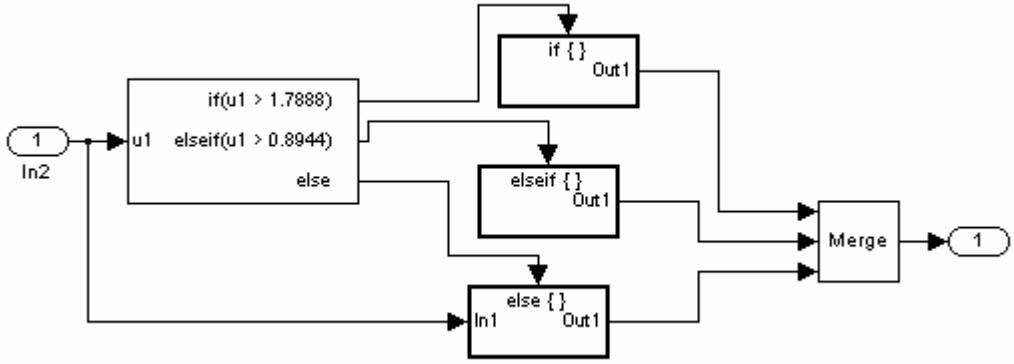
Figura 5.42:: Diagrama sistema Despreading

Los *delays* y *buffers* que se ven en la figura luego de los bloques *Despreading* son fundamentales para el correcto funcionamiento del sistema completo. El buffer agrupa en 480 bits, formando asi la trama de 2 ms de cada canal. Luego el concatenador encadena estas tramas, formando un vector donde las primeras 480 muestras corresponden al canal uno, las segundas 480 al canal dos y asi sucesivamente. El resto del sistema receptor trabaja de esta manera, esperando las tramas de cada canal serialmente, por lo que es imprescindible que éstas no se mezclen incorrectamente.

Los *delay* se incluyen para corregir retrasos que se introducen en el sistema y son inevitables. Como ya se comentó, en los filtros de modulación se retrasa la señal un símbolo, y luego, en el bloque *Depsreading* se introduce otro delay (en el bloque *Integrate and Dump*). Por lo que si se agruparan las 480 muestras en el buffer como vienen, se estarían incluyendo estas dos muestras incorrectas, y a la vez dejando dos muestras correctas para la próxima trama, lo que correría todas las tramas siguientes. Al incluir este delay de 478 muestras en cada canal, se logra que la primer trama que arma el buffer se lleve estas dos muestras incorrectas, para luego sí agrupar las muestras correctamente. Evidentemente esto introduce más delay, sin embargo éste se traduce exactamente en una trama de 2ms por lo que se puede manejar fácilmente y no genera un desorden de los datos

Cuantizador

Al final del sistema se debe agregar este bloque para convertir los valores que obtiene la operación *Despreading* a los símbolos 16QAM exactamente.



Se implementó con bloques if los cuales separan los rangos siguientes para asignar el valor correspondiente según la tabla 5.3 que define los símbolos 16QAM.

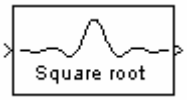
[0.8944 1.7888] => 1.3416
[0 0.8944] => 0.4472
[0 -0.8944] => -0.4472
[-0.8944 -inf] => -1.3416

5.11 Modulacion

Se modula en banda base. Por lo tanto, sólo se aplica el filtro raíz cuadrada de coseno elevado.

5.11.1 Filtro Transmisor

En el transmisor se usa el bloque filtro transmisor de coseno elevado.



El filtro transmisor coseno elevado sobremuestrea y filtra la señal de entrada usando un filtro FIR coseno elevado normal o un filtro FIR raíz cuadrada de coseno elevado.

El parámetro tipo de filtro determina que tipo de filtro usa el bloque; las opciones son normal o raíz cuadrada. El que define la norma es un filtro raiz cuadrada. La respuesta al impulso de un filtro raíz cuadrada de coseno elevado con factor de rolloff es:

$$h(t) = 4R.\frac{\cos\left(\frac{(1+R)\pi t}{T}\right) + \frac{\sin\left(\frac{(1-R)\pi t}{T}\right)}{\frac{4Rt}{T}}}{\pi\sqrt{T\left(1-\left(\frac{4Rt}{T}\right)^2\right)}}$$

La norma fija la siguiente respuesta:

$$RC_0(t) = \frac{\sin\left(\pi\frac{t}{T_C}(1-\alpha)\right) + 4\alpha\frac{t}{T_C}\cos\left(\pi\frac{t}{T_C}(1+\alpha)\right)}{\pi\frac{t}{T_C}\left(1-\left(4\alpha\frac{t}{T_C}\right)^2\right)}$$

Las ecuaciones son similares y desarrollando las mismas, tomando T=Tc y R=α , se puede ver que para que ambas sean iguales, debe multiplicarse la respuesta al impulso del bloque de Simulink por $\sqrt{Tc}$, siendo Tc=1/tasa de chips y tasa de chips=3,84e6

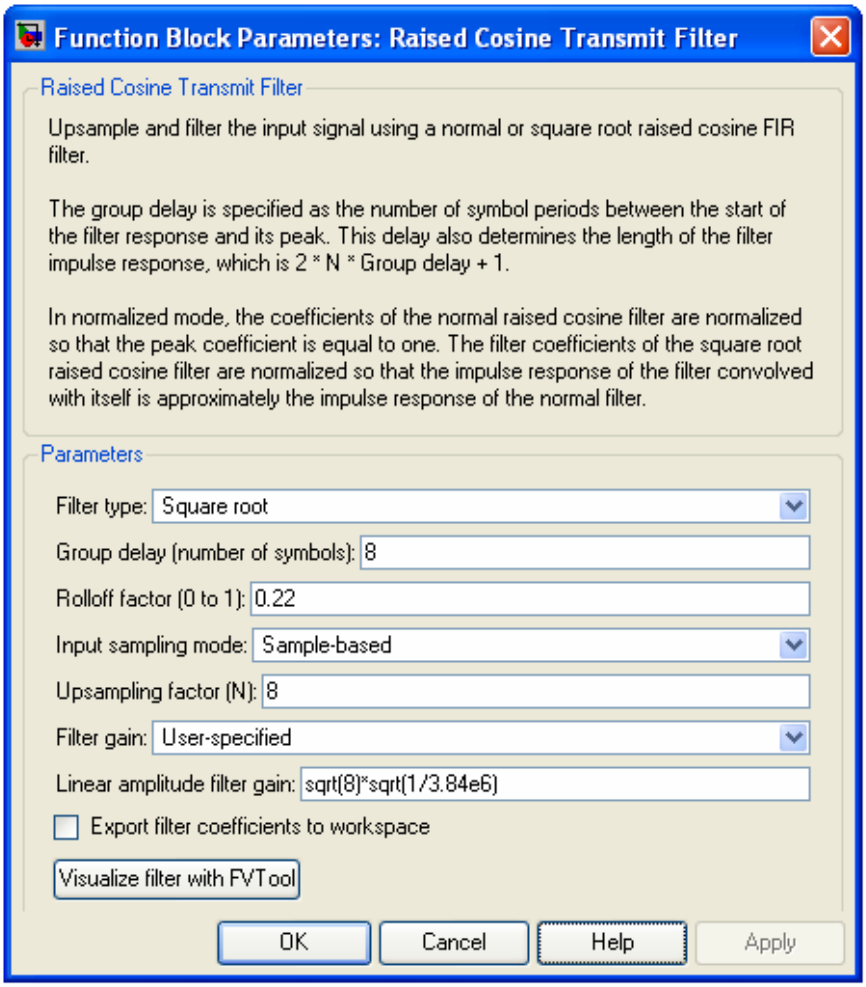


Figura 5.43: Parámetros pulse shaping

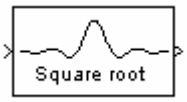
El retardo de grupo está relacionado al sobremuestreo y a la cantidad de coeficientes del filtro. Para elegir este parámetro también se tuvo en cuenta el delay que introduce en la señal, buscando que este delay no fuera más de un símbolo de datos.

El sobremuestreo se fijó en 8. Se eligió esta cantidad ya que figuraba en material de estudio [12] y también en el ejemplo de WCDMA de MATLAB.

El factor de rolloff está definido en la norma como 0,22.

La ganancia tiene en cuenta dos factores: el sobremuestreo y el hecho de que la respuesta del filtro no es exactamente la que se da en la norma como se explicó anteriormente.

5.11.2 Demodulación



El filtro receptor coseno elevado filtra la señal de entrada usando un filtro FIR coseno elevado normal o un filtro FIR raíz cuadrada de coseno elevado. También submuestrea la señal filtrada si se fija el parámetro modo de salida a submuestreo. Las características del filtro se detallan en el anexo.

Los parámetros son los mismos que en el caso del transmisor. Salvo la ganancia, que ahora es la raíz cuadrada del inverso del submuestreo y la raíz cuadrada de la tasa de chips.

5.12 Canal inalámbrico

Para simular el canal inalámbrico se creó un sistema que contiene bloques de Simulink que modelan la suma de ruido blanco más el efecto de la propagación en un entorno con multicaminos:

El sistema simplemente incluye los bloques AWGN y MultiPath Rayleigh Fading de Simulink.

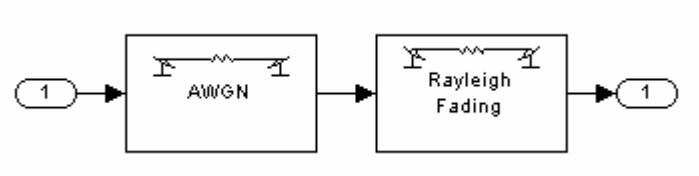


Figura 5.45: Canal inalámbrico

El bloque AWGN agrega ruido blanco gausiano a la señal de entrada. El bloque Rayleigh Fading multiplica la señal de entrada por muestras de un proceso aleatorio complejo de distribución Rayleigh

El siguiente es el cuadro de diálogo que presenta el sistema.

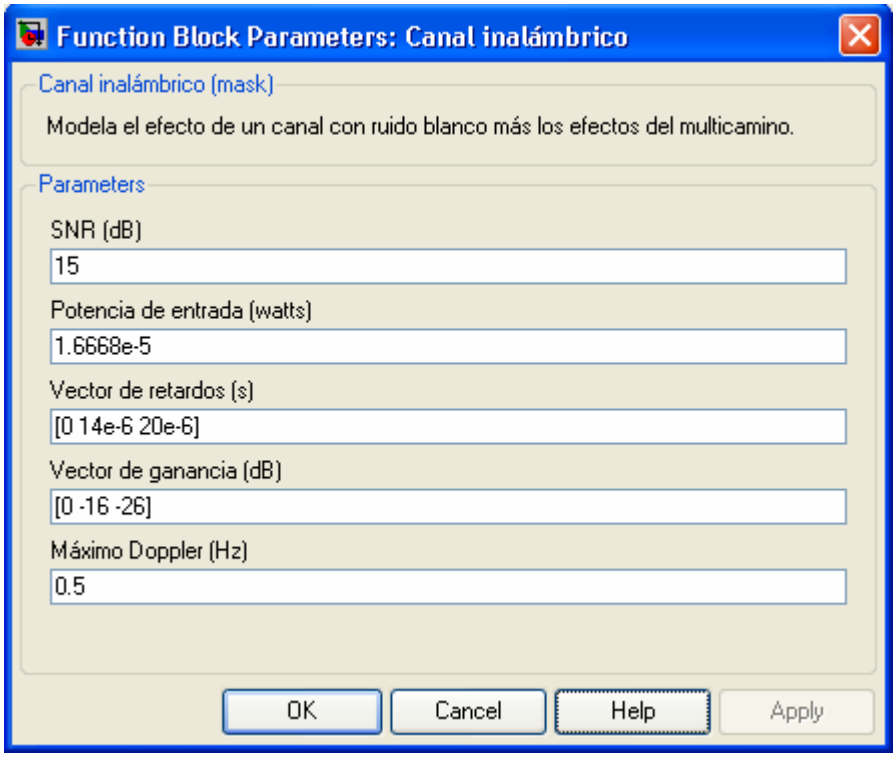


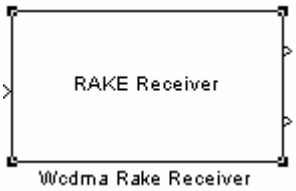
Figura 5.46: Canal inalámbrico

Se ingresan como parámetros la SNR del sistema y la potencia de entrada para modelar el ruido blanco. Para calcular este valor, se corrió en primer lugar el sistema transmisor completo guardando la señal de salida, y luego se calculó la potencia de la misma.

Para modelar los efectos del multicamino se debe ingresar además los vectores de retardo y ganancia de cada camino asi como la frecuencia máxima de doppler para el canal que se quiere representar.

5.13 Rake

Se utilizó el bloque WCDMA RAKE Receiver, incluido en el toolbox de Matlab.
Ubicación: toolbox/commblks/commblksdemos/wcdma_lib



Este bloque recibe tramas de longitud [*oversampling*]*[*SFmayor*], donde SFmayor es igual al spreading factor mayor entre el SF del canal de datos y el SF del CPICH y *oversampling* es el sobremuestreo aplicado en el pulse shapping del transmisor.

A la salida se generan tramas de longitud [ceil(SF_CPICH/SFdatos)], y serán los símbolos recibidos que luego pasarán al demodulador (demapper).

La figura siguiente es una rama del WCDMA RAKE Receiver. En cada rama se genera la secuencia de referencia, CPICH, en *el P-CPICH Generador*, que junto con la estimación de ésta misma señal obtenida en el *WCDMA Rake Finger*, se utilizará para la estimación del canal. También en cada rama se generan los códigos de scrambling y spreading con los que se recupera la señal (*WCDMA Orthogonal Codes Generador*).

El *WCDMA Rake Finger* se encarga del downsampling, despreding y descralmbling, tanto para los datos como para el CPICH. La salida de este bloque es introducida en el *WCDMA Data Derotation*, el cual multiplica por el conjugado de la estimación del canal.

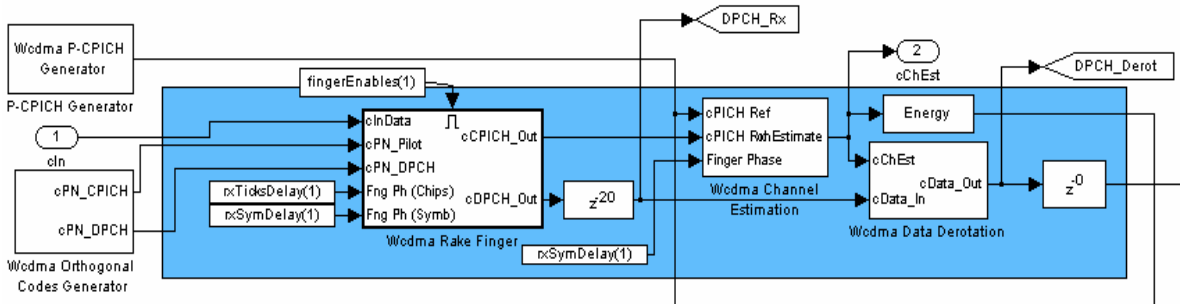


Figura 5.47 – Rama del receptor WCDMA RAKE receiver

Luego, las salidas de las cuatro ramas son combinadas en el *Wcdma Rake Combiner*, que simplemente suma y normaliza.

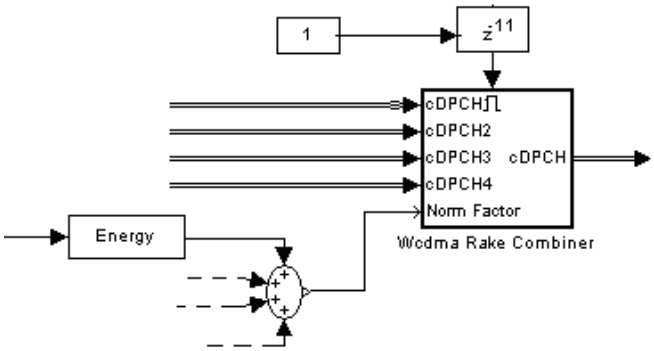


Figura 5.49 – Diagrama Wcdma RAKE Combiner

Parámetros del bloque:

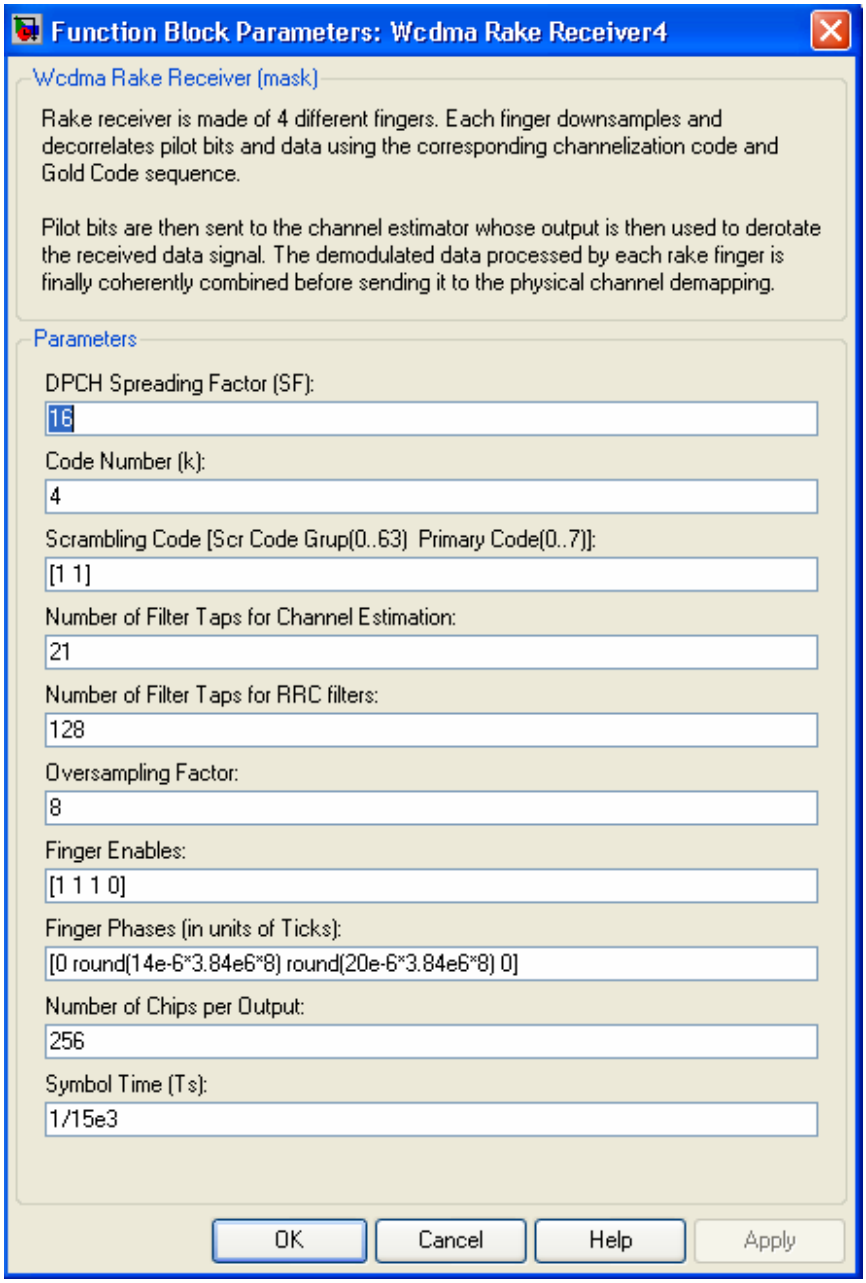


Figura 5.50

DPCH Spreading Factor:
Spreading Factor del canal de datos.

Code Number:
Entero correspondiente al índice del código ortogonal asignado al canal de datos.

Scramlbling Code
Vector de dos elementos correspondiente a los índices del código de scrambling asignado a la estación base.

Number of filter taps for channel estimation
Numero de coeficientes del filtro pasabajos implementado en en Wcdma Channel Estimation.

Number of filter taps for RRC
Número de coeficientes de del filtro root raised cosine.

Oversampling Factor:
Valor entero correspondiente al número de muestras por símbolo.

Finger Enables:
Vector de cuatro elementos que indica que indica qué ramas deben ser activadas. Con un uno se activa, con un cero queda desactivada.

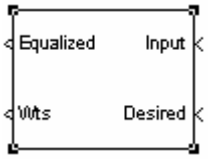
Finger phases:
Vector con los valores en chips de los retrasos correspondientes a cada camino

Number of chips per out:
Número de chips que se va a obtener a la salida del bloque.

5.14 Ecualizador

Como se explicó en el punto 4.2.3, se implementaron ecualizadores en dos niveles diferentes, a nivel de símbolo y a nivel de chip. En ambos casos se utilizan los bloques RLS Linear Equalizer y Digital Fliter, incluidos en Simulink.

RLS Linear Equalizer:



El bloque ecualizador lineal RLS usa un ecualizador lineal y el algoritmo RLS para ecualizar una señal modulada linealmente en banda base a través de un canal dispersivo. Durante la simulación, el bloque usa el algoritmo RLS para actualizar los coeficientes, una vez por símbolo.

El puerto llamado *Input* recibe la señal que se quiere ecualizar. El puerto llamado *Desired* recibe la secuencia de referencia. La salida *Equalized* saca el resultado del proceso de ecualización. Y la salida *Wts*, da los coeficientes del filtro.

Los parámetros que se deben especificar para este bloque son los siguientes:

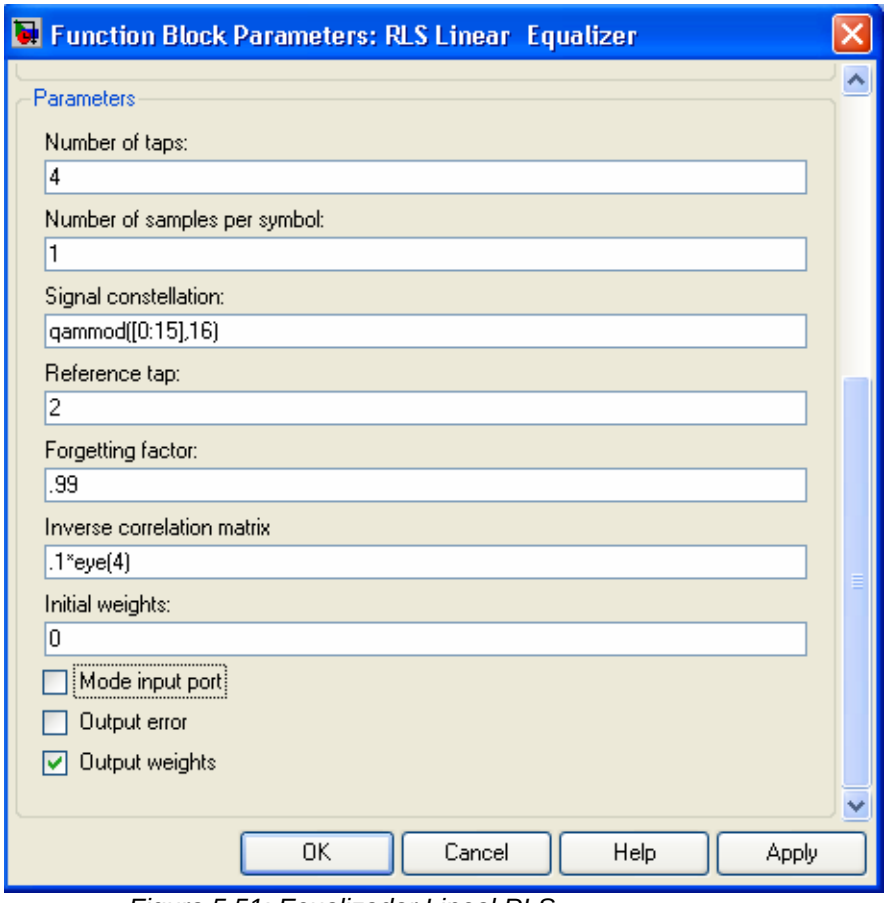


Figura 5.51: Ecualizador Lineal RLS

***Number of taps**:*
La cantidad de coeficientes en el filtro del ecualizador lineal.

***Number of samples per symbol**:*
La cantidad de muestras de entrada para cada símbolo.

***Signal constellation**:*
Un vector de números complejos que especifica la constelación para la modulación.

Reference tap:
Un entero positivo menor o igual que la cantidad de coeficientes en el ecualizador. Se aconseja que se coloque a la mitad del número total de taps establecidos.

***Forgetting factor**:*
El factor de olvido del algoritmo RLS, un número entre 0 y 1.

Inverse correlation matriz:
El valor inicial para la matriz inversa de correlación utilizada en el algoritmo RLS. La matriz debe ser N-por-N, donde N es la cantidad de coeficientes.

***Initial weights**:*
Un vector que lista los pesos iniciales de los coeficientes.

Mode input port:
Si se marca esta casilla, el bloque tiene un puerto de entrada que permite alternar entre los modos entrenamiento y dirigido por decisiones.

Digital Filter:
Este bloque filtra la entrada con un filtro digital IIR o FIR. Puede implementar filtros estáticos con coeficientes fijos o filtros variables con coeficientes cambiando en el tiempo.

Parámeros a definir en este bloque:

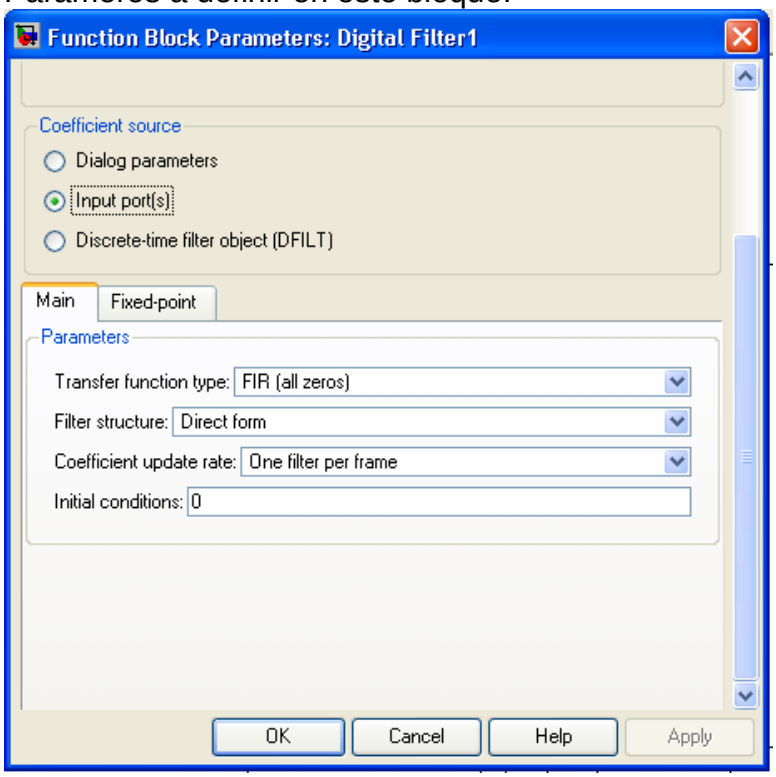


Figura 5.52

***Transfer function type**:*
Selecciona el tipo de función del filtro, IIR o FIR.

Select the filter structure:
Selecciona la estructura del filtro

***Coefficient update rate**:*
Especifica cuándo se actualiza el filtro, puede ser una vez por muestra o una vez por trama. Este parámetro solo afecta en el caso que la entrada sea una trama.

***Initial conditions**:*
Condiciones iniciales de los estados del filtro.

En ambas implementaciones, tanto a nivel de símbolo como de chip, se trabajará con el mismo FIR. La siguiente es la estructura del filtro FIR que se

utilizó, una forma directa, donde los *bi* son los coeficientes del filtro que en nuestro caso surgen del ecualizador RLS.

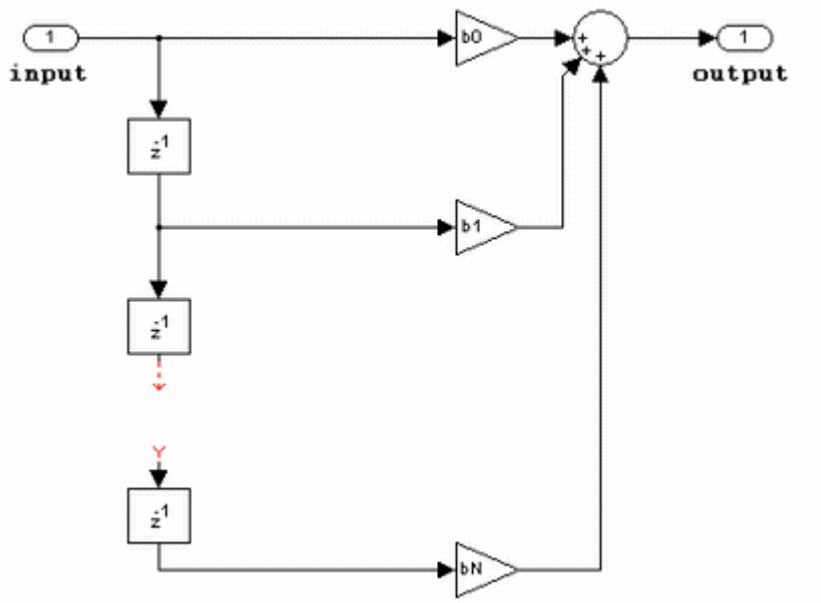


Figura 5.53: Estructura Filtro FIR forma directa

5.14.1 Ecualizador a nivel de Símbolo

En la Figura 5.54 se muestra el diagrama general del sistema con ecualizador a nivel de símbolo

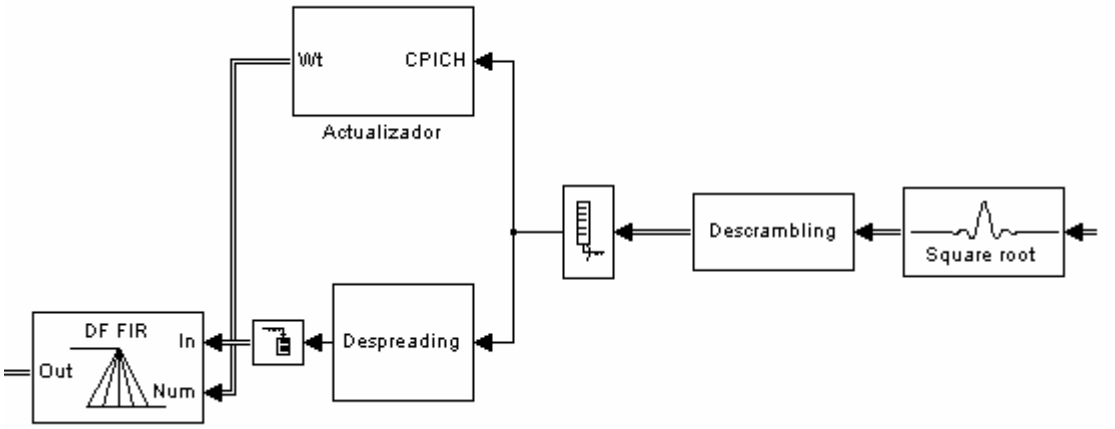


Figura 5-54: Diagrama Ecualizador a nivel de símbolo

La señal recibida es por un lado introducida en el bloque *Despreading,* el cual convierte la señal a los símbolos de datos 16QAM para los quince canales de datos. Por otro lado también va hacia el bloque *Actualizador*, el cual se encarga de actualizar los coeficientes que se utilizarán en el filtro FIR. Dentro de este bloque, como se puede ver en la Figura 5.55, se aplica la operación Despreading pero únicamente para obtener los símbolos del canal CPICH, o

sea con un SF igual a 16 y con el código OVSF correspondiente. Dentro del bloque también se genera la señal CPICH nuevamente, que será utilizada como la señal deseada.

Estas dos señales, la generada en el bloque y la obtenida de la señal recibida son introducidas en el *RLS Linear Equalizer1*, bloque que se encarga finalmente de la actualización de los coeficientes.

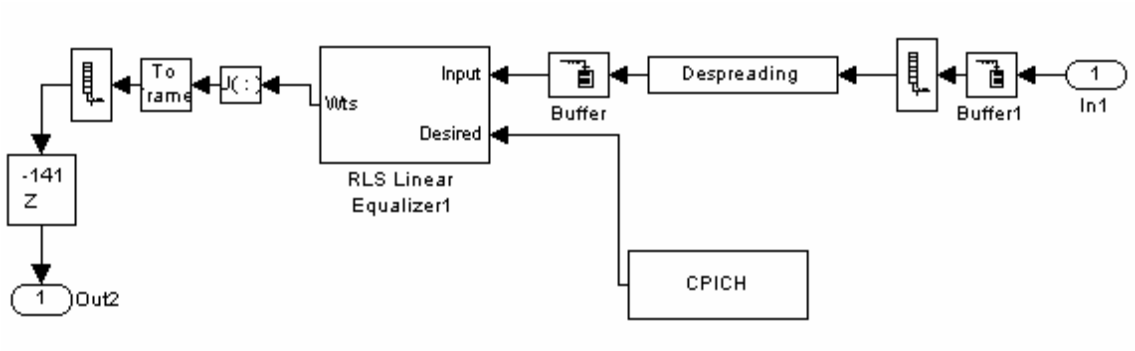


Figura 5.55: Actualizador

El buffer1 que se observa inmediatamente en la entrada del bloque *Actualizador*, se encarga de agregar algunas muestras como delay a los efectos de que en el *Despreading* se puedan formar los símbolos correctamente. Recordemos que la señal recibida tiene un delay igual a un símbolo de datos, o 16 chips, que se arrastra de los filtros de modulación. Si no tenemos en cuenta estas 16 muestras erróneas, el Despreading que integra en períodos de 256 por tratarse del canal CPICH, no agrupará bien los chips y obtendrá símbolos errados. Por lo tanto se agrega aquí un delay de 240 muestras, para que se agrupen con las 16 incorrectas formando un primer símbolo errado, para luego sí formar los símbolos correctos.

El segundo buffer que se ve en la figura luego del Despreading, se agrega solamente porque (luego de varios intentos fallidos) se encontró un reporte de bug de simulink [11], que indica que el bloque RLS Linear Equalizer no funciona bien si su entrada no es una trama. Por esta razón se incluye un buffer que agrupa de a 10 muestras.

A la salida de los coeficientes también se agrega un delay, el mismo es para sincronizar los coeficientes con la señal de datos, de forma que en el filtro FIR se apliquen los coeficientes a los datos que corresponde.

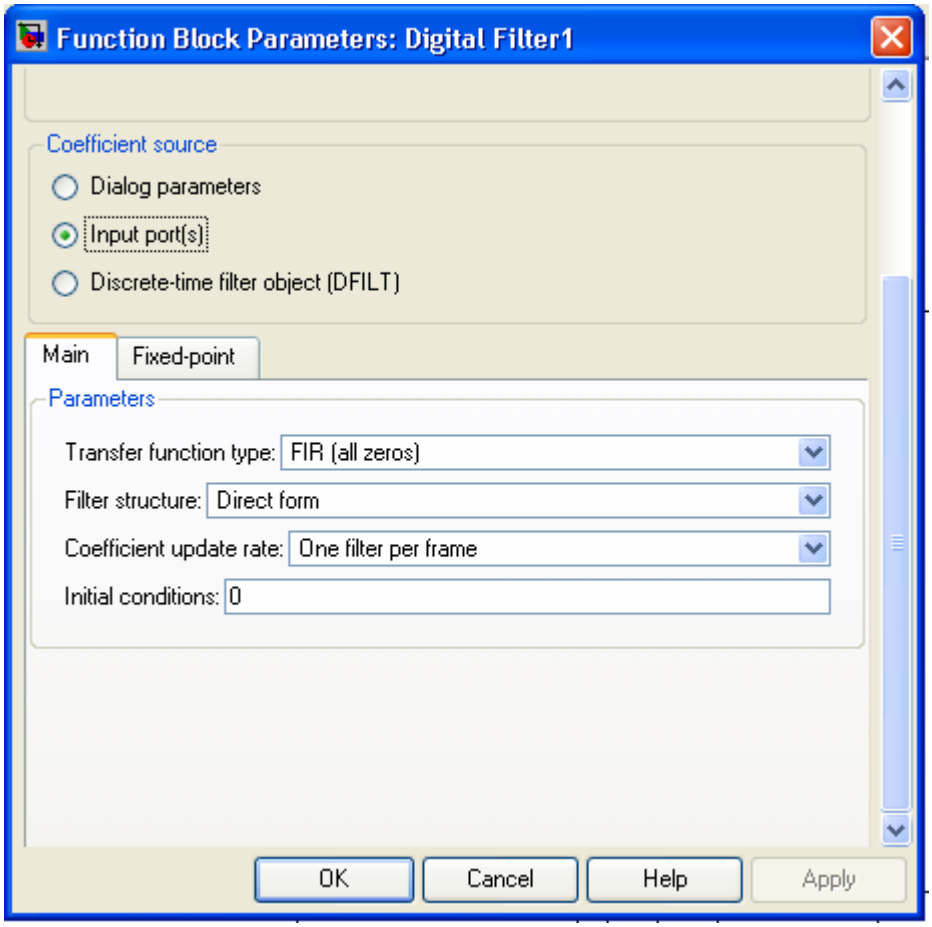


Figura 5.56

En la caja de diálogo se indica que los coeficientes serán una entrada del bloque y la estructura del filtro mostrada antes. También se especifica que se actualizarán los coeficientes para cada trama, o sea que por cada trama de 16 símbolos de datos que ingresan al filtro se utilizan los mismos coeficientes, calculados a esta tasa de 16 símbolos de datos, o un símbolo PILOT.

5.14.2 Ecualizador a nivel de Chip

En la Figura 5.57 se puede observar el diagrama general del sistema con ecualizador a nivel de chip

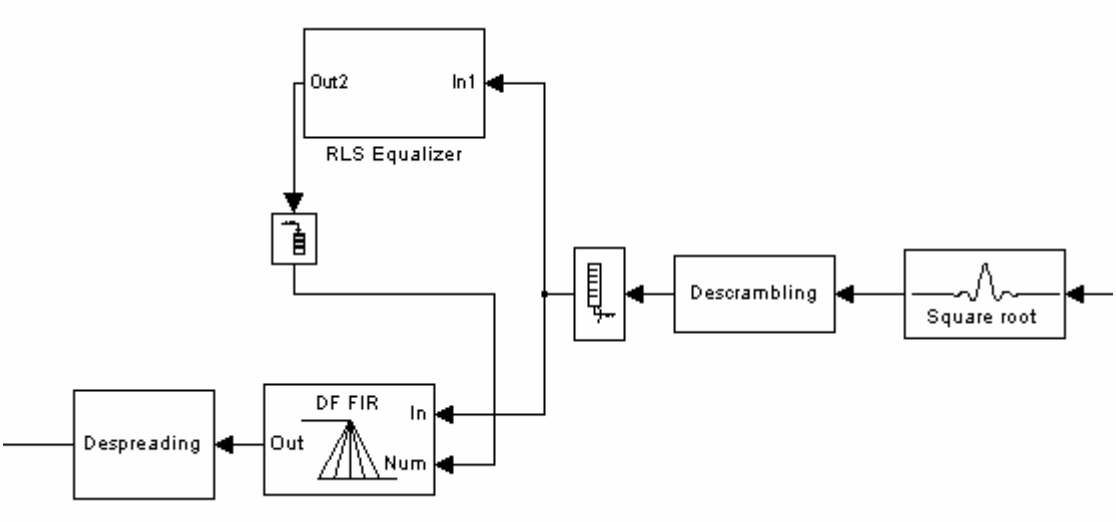


Figura 5-57: Diagrama Ecualizador a nivel de chip

Este diagrama es similar al anterior, pero la diferencia es la ubicación del bloque *Despreading.* En este caso la operación de *Despreading* que devuelve los símbolos de datos 16QAM se efectúa después de aplicar el filtro FIR. Es así que la señal que entra al filtro FIR serán datos a nivel de chip y no símbolos como en el caso anterior.

También se implementa diferente el Actualizador, como se muestra en la imagen siguiente.

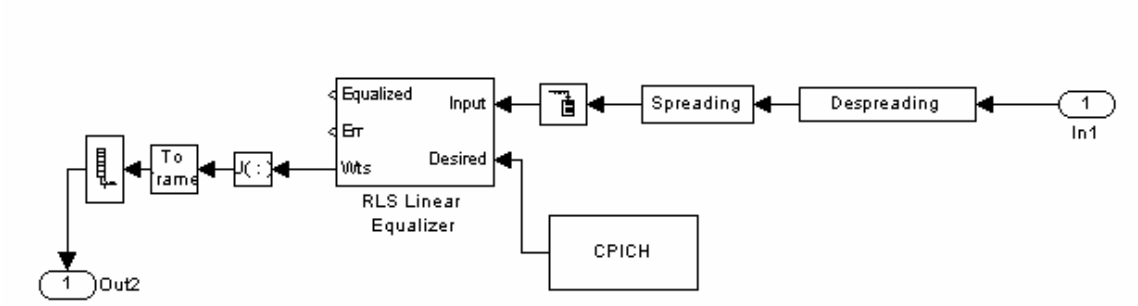


Figura 5.58: Diagrama Actualizador a nivel de chip.

En este caso también se aplica la operación Despreading con SF = 256, para obtener los datos del canal CPICH, pero luego se vuelve a pasar estos datos a chips a través del Spreading. De esta forma se consigue la señal de chips solamente con la información de los bits Pilots, que se utiliza en el RLS Linear Equalizer para conseguir los coeficientes.

Evidentemente este ecualizador debe contar con una mayor capacidad de procesamiento ya que se actualizan los coeficientes por cada chip que se recibe. Esto también ocasiona que el tiempo de simulación sea mayor que en el caso del ecualizador a nivel de símbolo. Sin embargo, este ecualizador será mas ‘adaptivo’ que el anterior, ya que la actualización de coeficientes se hace a cada instante de chip, registrando asi cambios más rápidos del canal que en el otro caso no se llegan a detectar.

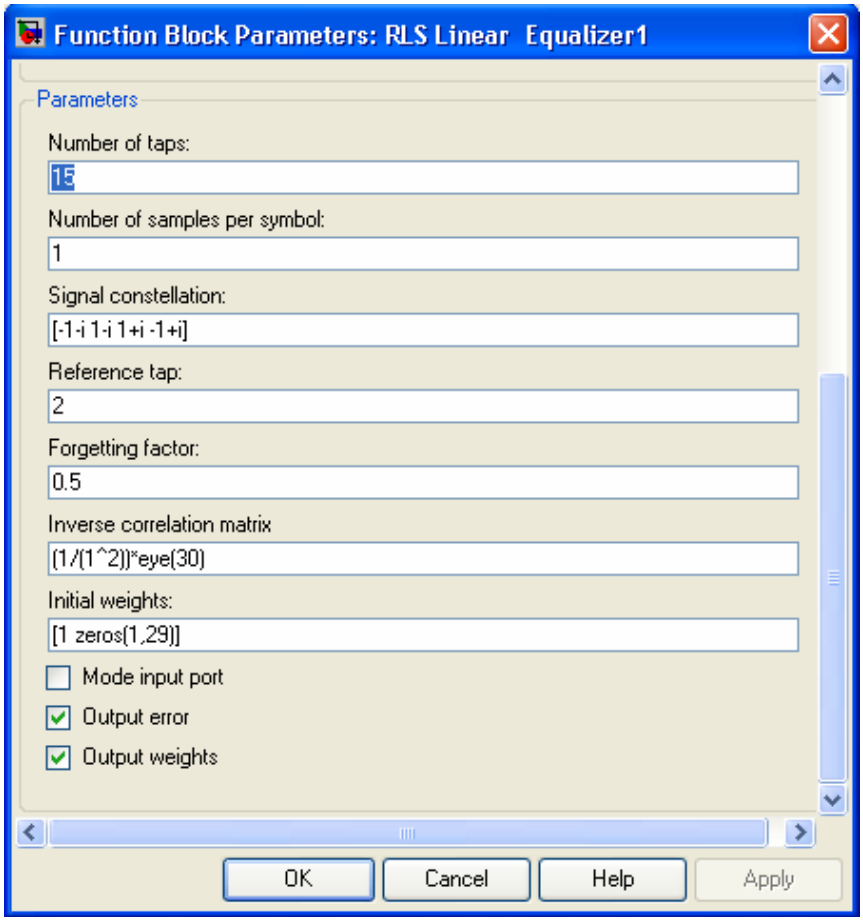


Figura 5.59: Ecualziador a niel de chip

Los parámetros para el ecualizador se encontraron luego de probar diferentes combinaciones de los mismos para llegar a los valores que obtenían mejores resultados. En primer lugar se investigó en material disponible sobre este tema, pero se encontraron diferentes recomendaciones por lo que optamos por hacer un procedimiento de pruebas para lograr obtener los parámetros que en este caso responderían mejor.

6 Simulaciones

6.1 Consideraciones previas a las simulaciones

Mientras se investigaba el sistema simple, se comenzó a realizar algunas simulaciones y se encontró que el BER resultaba muy alto. Mediante una minuciosa investigación, se encontró que el problema se presentaba en el bloque *Bit scrambling*. Para verificar esto, se realizó la siguiente prueba: se colocó una fuente, el bloque de bit scrambling, ruido blanco que generara una pequeña diferencia y el inverso del bit scrambling, como se ve en la siguiente figura.

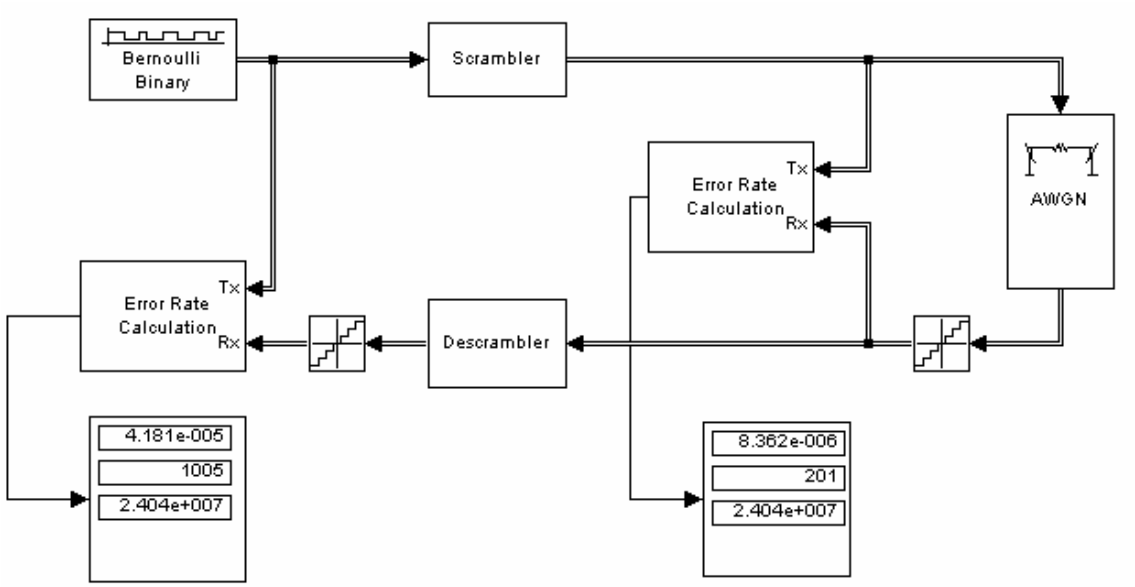


Figura 6.1: Prueba de Bit scrambling

Se verificó que el error se multiplica aproximadamente por 5. Se revisó el funcionamiento de los bloques y no se encontró ninguna falla. Por lo que se decidió en primer lugar seguir adelante con las simulaciones quitando este bloque del sistema. Este bloque simplemente evita secuencias que repitan el mismo símbolo, que podría ocurrir para cierto tipo de contenido, pero en este caso se está usando un generador aleatorio de bits por lo que esto no sucede. Concluimos que su función en este sistema no altera los resultados de BER.

En los siguientes puntos se muestran los resultados obtenidos en las simulaciones para el sistema simple, receptor RAKE y ecualizador. En el capítulo 7, se presenta un análisis de estos resultados a través de gráficas e imágenes que permiten una mejor visualización de los mismos.

Todas las simulaciones excepto para el caso del ecualizador se corrieron durante 1 segundo.

6.2 Sistema simple

Se simuló este sistema en un entorno con ruido blanco solamente y luego agregando los canales multicamino.
Los siguientes son los resultados para estas simulaciones.

6.2.1 Sólo AWGN

SNR (dB)	VER
3	0.04646
6	0.008966
10	6.317e-5
15	0
20	0

Tabla 6.1: Sistema simple con AWGN

6.2.2 AWGN y Multicamino

SUI6

SNR	BER
3	0.365
6	0.3524
10	0.3426
15	0.3374
20	0.3357
22	0.3354
24	0.3352

Tabla 6.2: Sistema simple con AWGN + SUI6

6.3 Receptor Rake

Luego de las simulaciones para el sistema simple se siguió adelante con el receptor RAKE. Para este sistema se corrieron las mismas simulaciones: con ruido blanco en primer lugar, y luego se agregó un canal multicamino. Además se implementaron simulaciones para analizar la incidencia de la interferencia debido a los múltiples canales transmitidos.

Se presentan los resultados en las siguientes tablas.

6.3.1 Solo con AWGN

SNR (dB)	BER
3	0.0748
6	0.03591
10	0.01578
15	0.009573
20	0.008064

22	0.007826
24	0.007697

Tabla 6.3: Receptor RAKE con AWGN

6.3.2 AWGN y multicamino

SUI6:

SNR	BER
3	0.1142
6	0.07803
10	0.05458
15	0.04419
20	0.04104
22	0.04048
24	0.04015

Tabla 6.4: Receptor RAKE con AWGN + SUI6

6.3.3 BER vs cantidad de canales físicos

Canales físicos	Tiempo de simulación (s)	Bits Transmitidos	Errores	BER
1	0.5	4.742e5	0	0
2	0.5	9.49e5	1201	0.001266
3	0.5	1.423 e6	35242	0.003684
4	0.5	1.90e6	1.47e4	0.007723
6	0.5	2.845e6	5.62e4	0.01974
8	0.5	3.79e6	1.35e5	0.03545
10	0.5	4.74e6	2.50e5	0.0527
12	0.5	5.69e6	2.27e5	0.06752
15	0.5	7.11e6	6.95e5	0.09773

Tabla 6.5: Canales Físicos Vs BER

Para obtener estos resultados, no se tuvo en cuenta el sistema completo sino que las medidas de error rate fueron realizadas justo antes de la modulación, o sea antes del mapper (en el transmisor). Se eligió tomar la medida de esta forma porque para hacer esta prueba con el sistema completo habría que cambiar algunos bloques importantes de la parte del sistema que no estamos utilizando, por ejemplo el Rate Matching y la codificación. Consideramos que no es necesario modificar todo el sistema para esta prueba ya que evaluando solo esta parte se puede ver en los resultados lo que se quiere mostrar, la incidencia de la cantidad de canales físicos transmitidos en el BER.

6.4 Ecualizador

El sistema con el ecualizador debería ser el que obtuviera los mejores resultados en toda la simulación. Al tiempo de entregar el proyecto no ha sido posible solucionar un inconveniente que se presentó en esta implementación. El sistema que se pudo lograr, funciona correctamente por un período de tiempo y luego se dispara. Por esta razón se optó por presentar los resultados obtenidos durante este período en que el sistema funciona correctamente. El mismo es de 0,1 segundo y los resultados son los siguientes:

6.4.1 Sólo AWGN

SNR (dB)	BER
3	0.09844
6	0.05034
10	0.01542
15	0.001707
20	2.385e-6
22	0
24	0

Tabla 6.6: Ecualizador AWGN

6.4.2 AWGN y Multicamino

SUI6:

3	6.94e-2
6	2.45e-2
10	3.5e-3
15	4.07e-4
20	6.87e-5

Tabla 6.7: Ecualizador con AWGN + SUI6

7 Análisis de Resultados

7.1 Receptor Simple

Las figuras siguientes muestran la constelación recibida por el sistema simple en un entorno con ruido blanco solamente.

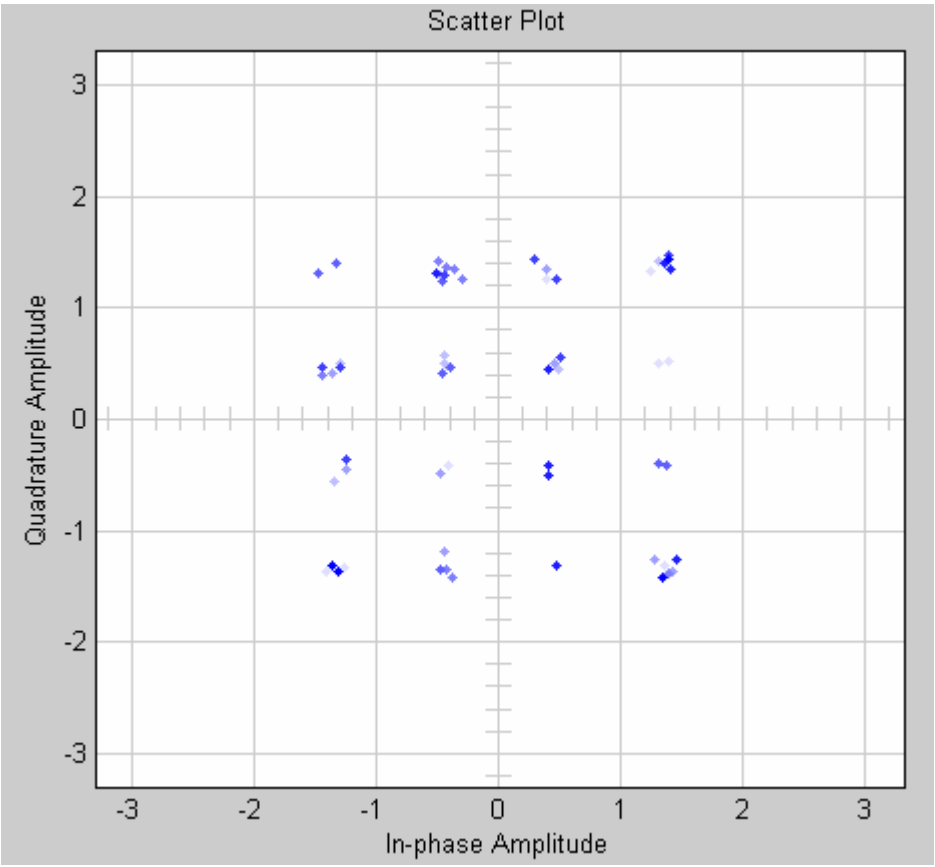


Figura 7.1: Constelación en receptor simple con SNR = 15

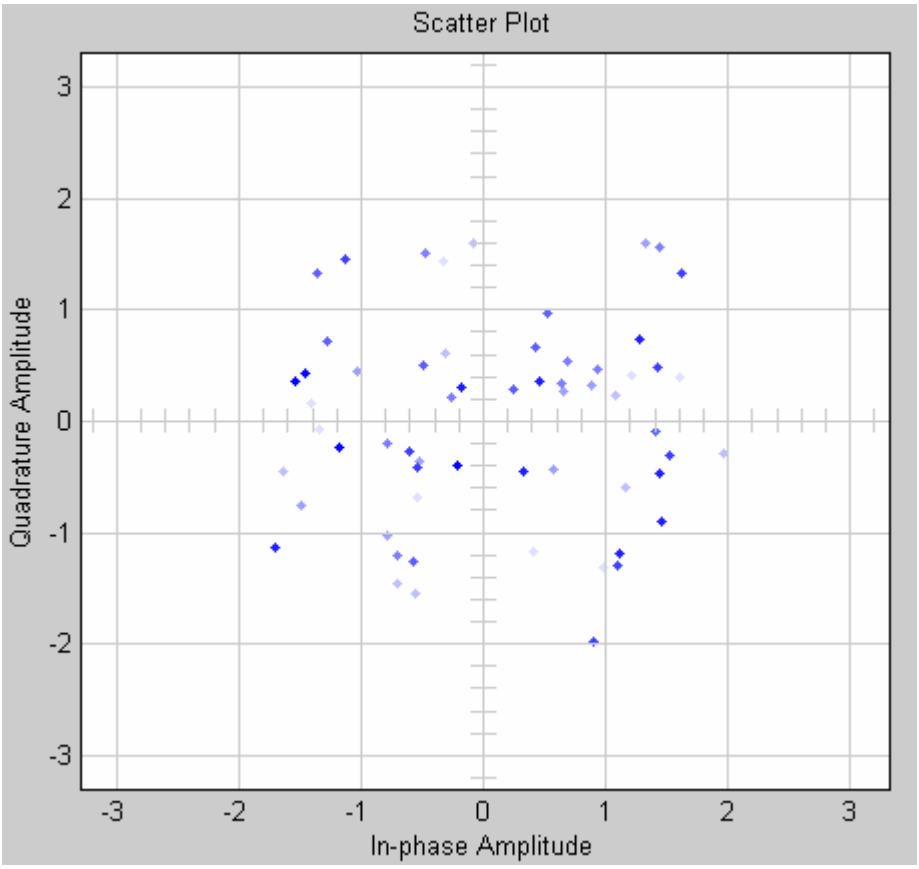


Figura 7.2: Constelación en receptor simple con SNR = 3

De los datos anteriores se puede afirmar que el sistema implementado es capaz de detectar la señal transmitida en un ambiente con ruido blanco. Se puede ver en la Figura 7.3 la relación entre el SNR y el BER para este caso.

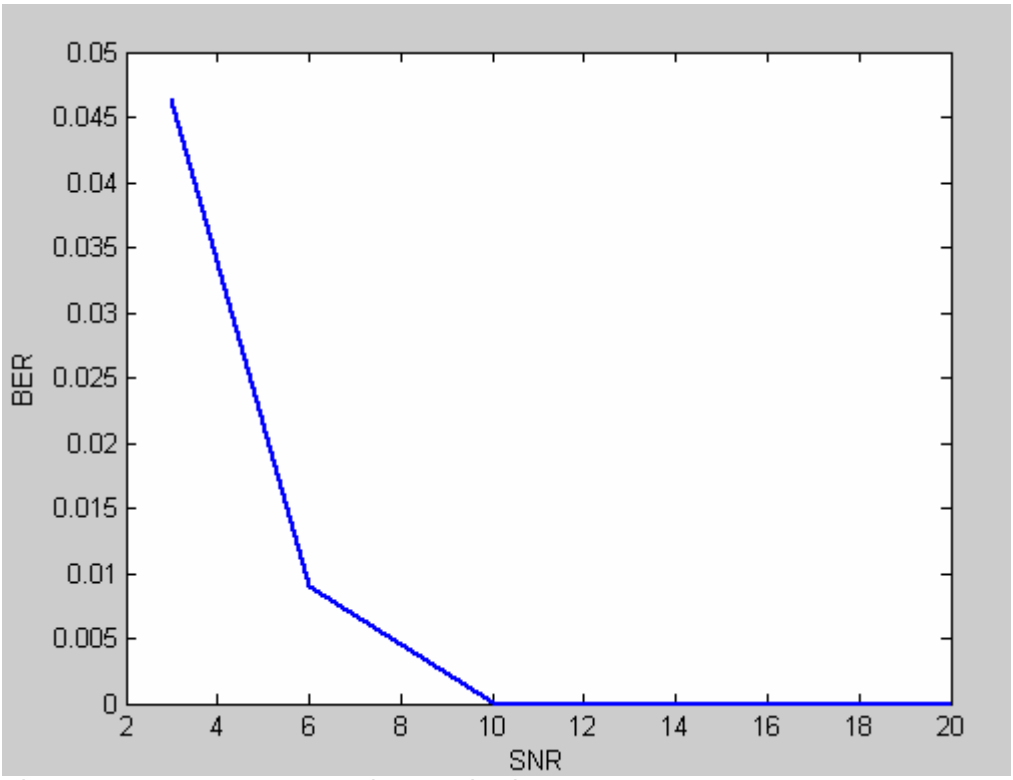


Figura 7.3 SNR Vs BER para sistema simple con AWGN

Las siguientes son las constelaciones detectadas en el caso del sistema simple en un ambiente multicamino.

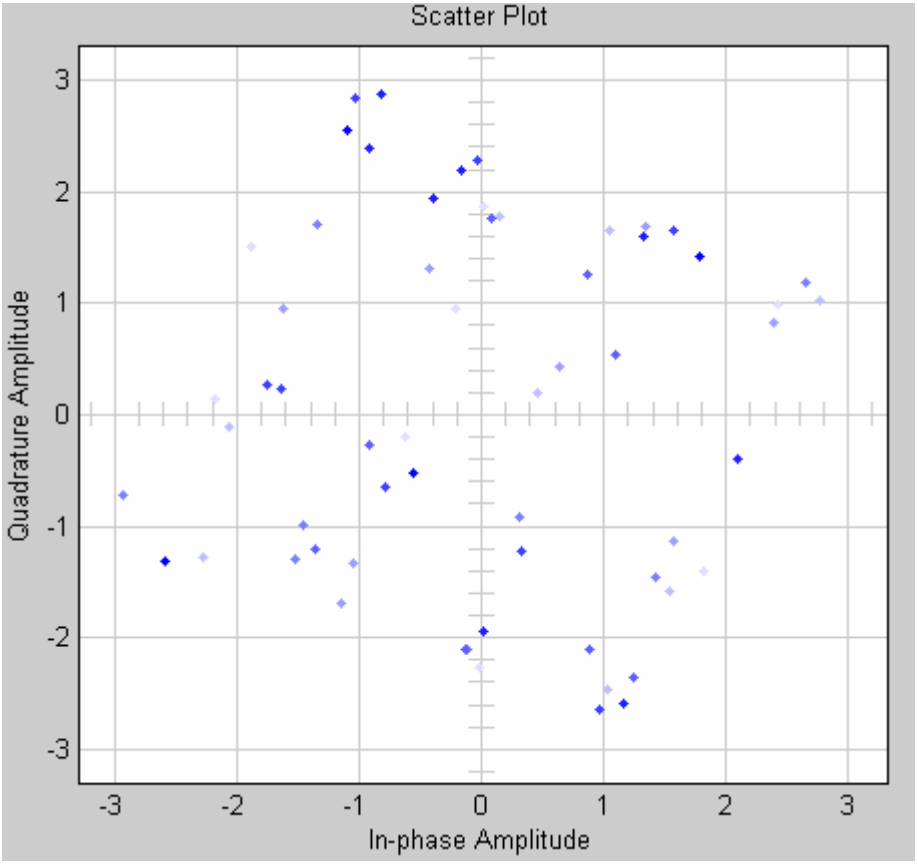


Figura 7.4: Constelación en receptor simple con SNR = 15 + SUI-6

Es evidente tanto de las tablas como de las gráficas de constelación, que el canal multicamino destruye totalmente la señal. Es imposible recuperar la señal transmitida para este sistema cuando la misma atraviesa un entorno multicamino, independientemente del SNR que se pueda obtener.

Efectivamente el sistema no fue diseñado para trabajar en tal entorno, por lo cual era un resultado esperado. Habiendo verificado este punto, se reafirma la necesidad de un rediseño del receptor de forma que sea capaz de detectar la señal incluso en un entorno con rebotes como lo es el canal multicamino.

7.2 Receptor Rake

Se puede verificar que para el caso de ruido blanco los resultados son similares al punto anterior. La figura muestra la constelación recibida en el caso de SNR = 15.

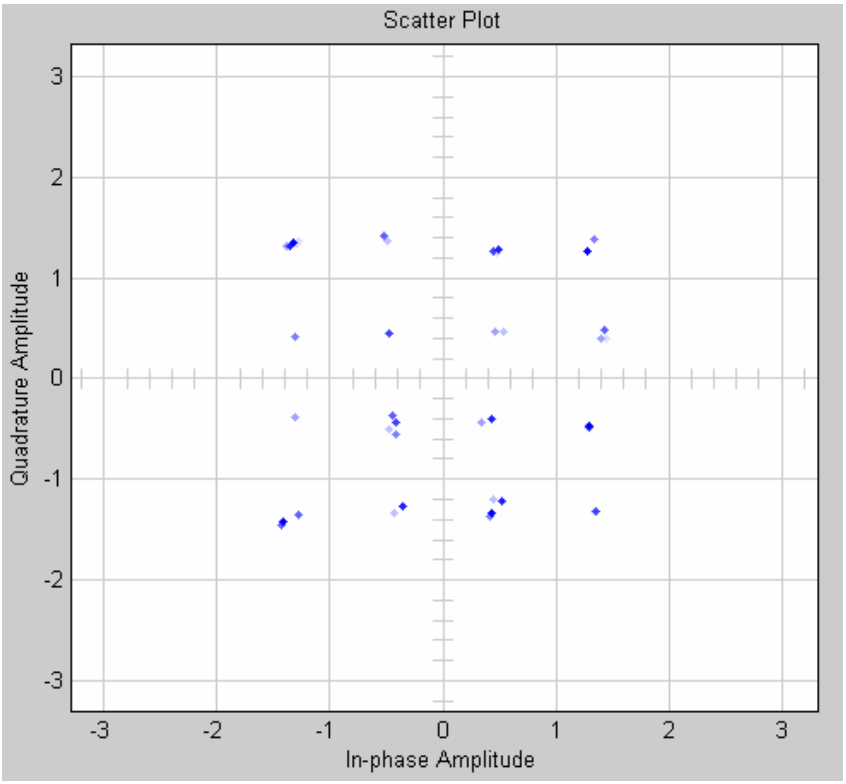


Figura 7.5: Constelación en receptor RAKE con SNR = 15

En el caso del canal multicamino, se puede observar que aunque presenta mejoras respecto al sistema simple, el receptor RAKE no mejora como uno esperaría dado que este receptor está pensado para compensar el multicamino.

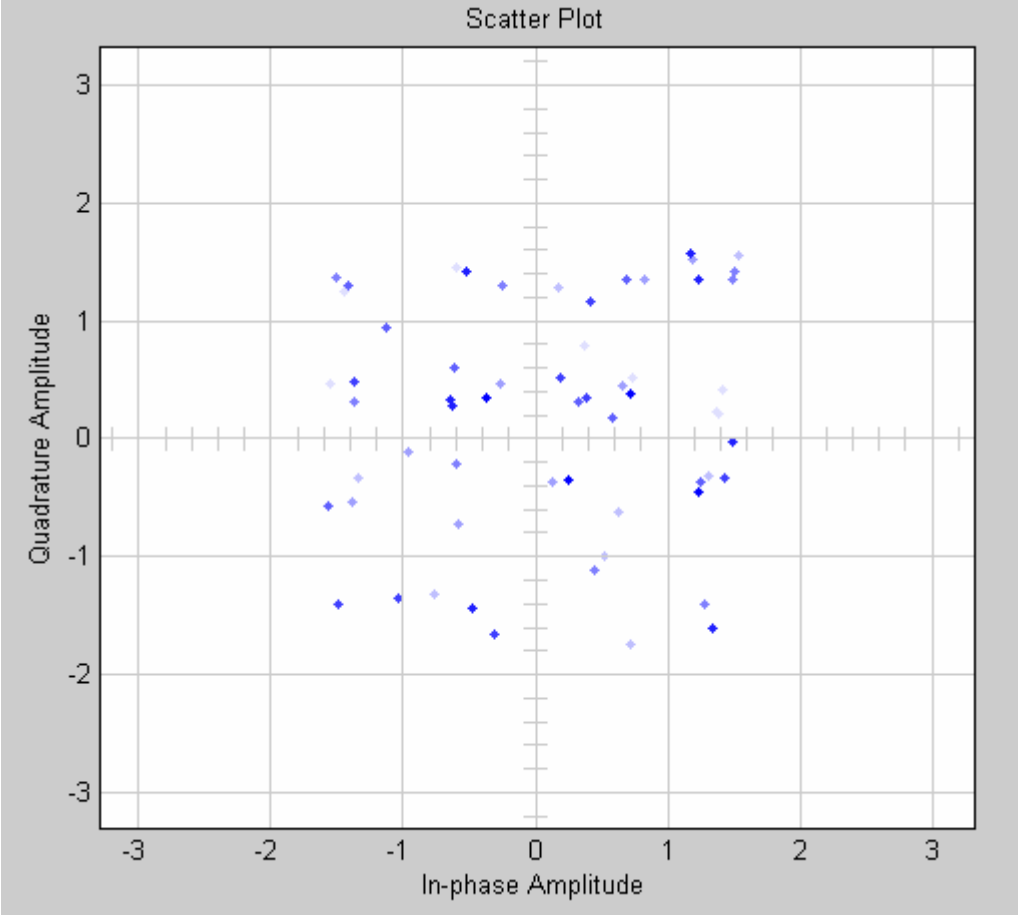


Figura 7.6: Constelación en receptor RAKE con SNR = 15 + SUI-6

Mientras investigamos receptores RAKE y ecualizadores en la literatura disponible, encontramos, por ejemplo [13] “HSDPA es una extensión de UMTS que está hecha a medida para throughputs de datos de hasta 14.4 Mbps. Para llegar a estas tasas tan altas, es indispensable que el receptor cancele la Interferencia de Acceso Múltiple (MAI) para obtener una calidad de canal lo suficientemente alta.

En la práctica esto hace que sea imposible usar el receptor RAKE convencional y hace que la ecualización sea atractiva para sortear estos problemas.” Encontramos varias referencias de este estilo, donde se indicaba que para HSDPA, en particular en las condiciones que estamos investigando que es con la máxima tasa de datos, el receptor RAKE no era apropiado.

En la referencia se plantea que la ortogonalidad de los códigos se destruye por el multicamino y que esto genera MAI (Múltiple Acces interference). Se puede ver en la siguiente gráfica extraida de referencia citada como el BER disminuye con la cantidad de códigos utilizados.

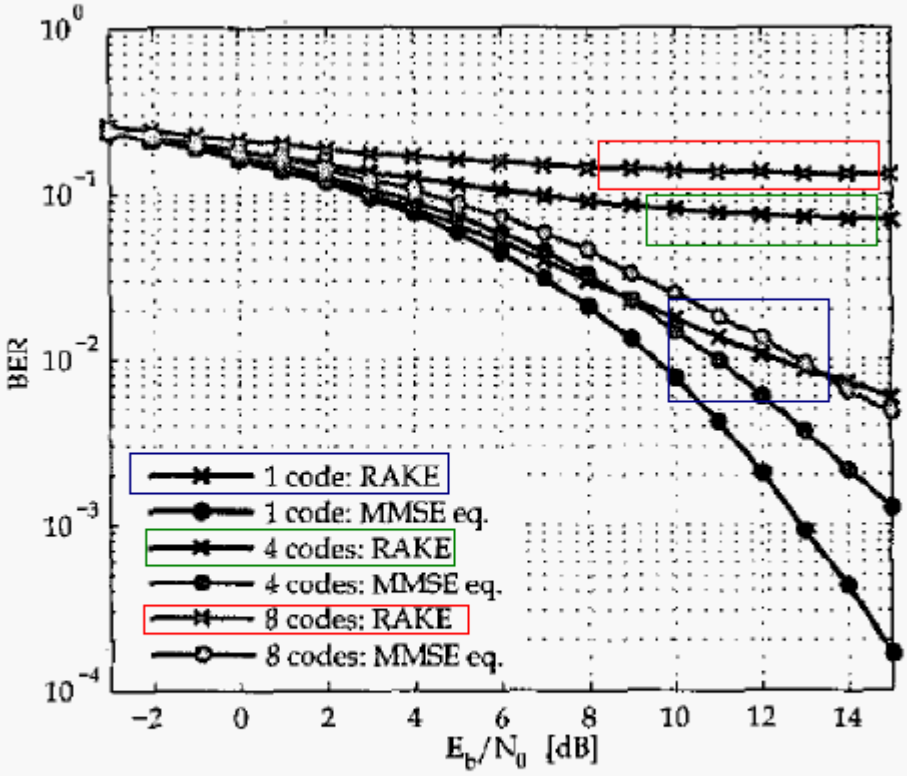


Fig. 5. Comparison of the RAKE receiver and the MMSE equalizer (MMSE: $L_f = 48$, RAKE: $L_a = 16$, static Ped. B).

Figura7.6

Esto a su vez nos llevó a investigar el efecto del incremento de canales en el BER para nuestro sistema. La siguiente gráfica se confeccionó con los datos obtenidos, verificando el resultado anterior.

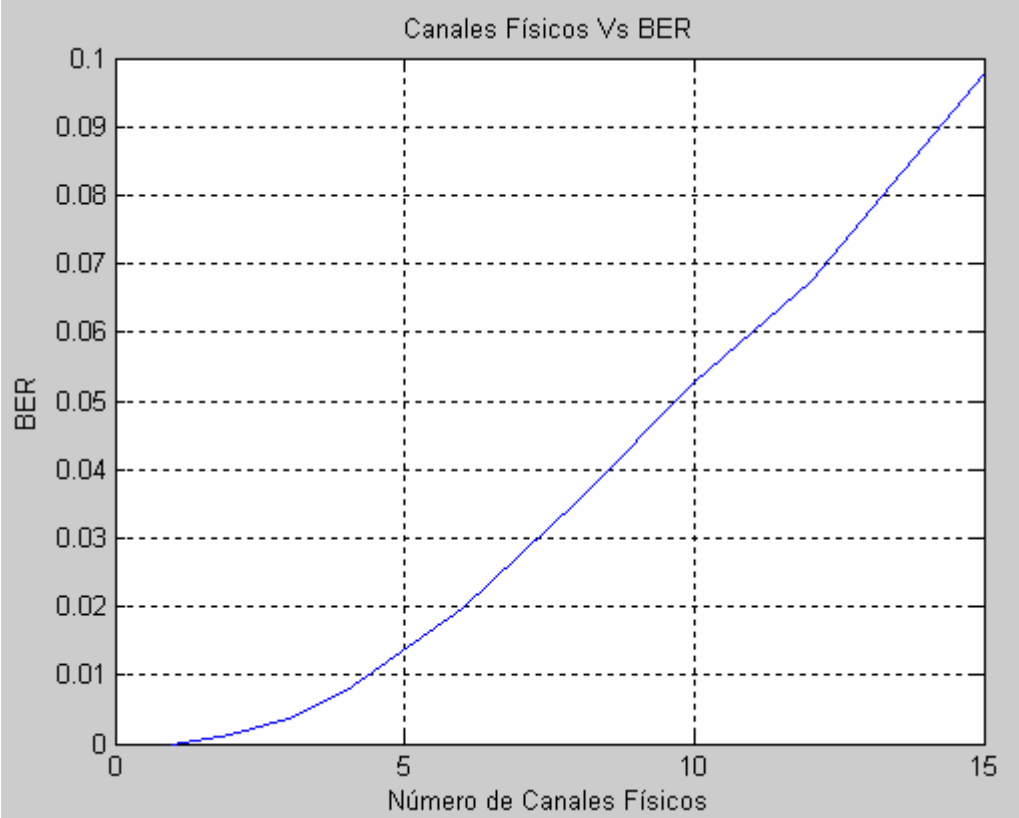


Figura 7.7: Canales Físicos Vs BER

En [14] se planteaba que cuando el spreading factor es pequeño las secuencias de spreading tienen malas propiedades de autocorrelación lo que genera ISI. A su vez, se demuestra en [15] que para spreading factor menores que 32 los efectos del ISI hacen que se degrade rápidamente la performance y nosotros estamos trabajando con spreading factor 16, por lo tanto nos encontramos en dicho caso.

Otras referencias donde se menciona que el receptor RAKE no es apropiado para HSDPA son [16], [17], [18].

7.3 Ecualizador

Al igual que en el caso del receptor rake, verificamos un buen funcionamiento del ecualizador en un ambiente con ruido blanco como único ruido. Los resultados son similares al caso del sistema simple, siendo éste un resultado esperado ya que este sistema se implementa enfocado en mejorar la performance en un entorno de multicaminos.

En las siguientes figuras se puede ver el efecto del ecualizador en el sistema en el caso de un ambiente multicamino. La primera muestra la constelación detectada por el sistema ecualizador a nivel de símbolo exactamente antes del filtro FIR. Se aprecia una rotación en la misma de aproximadamente 30 grados. La segunda gráfica, muestra la señal luego de atravesar el filtro FIR.

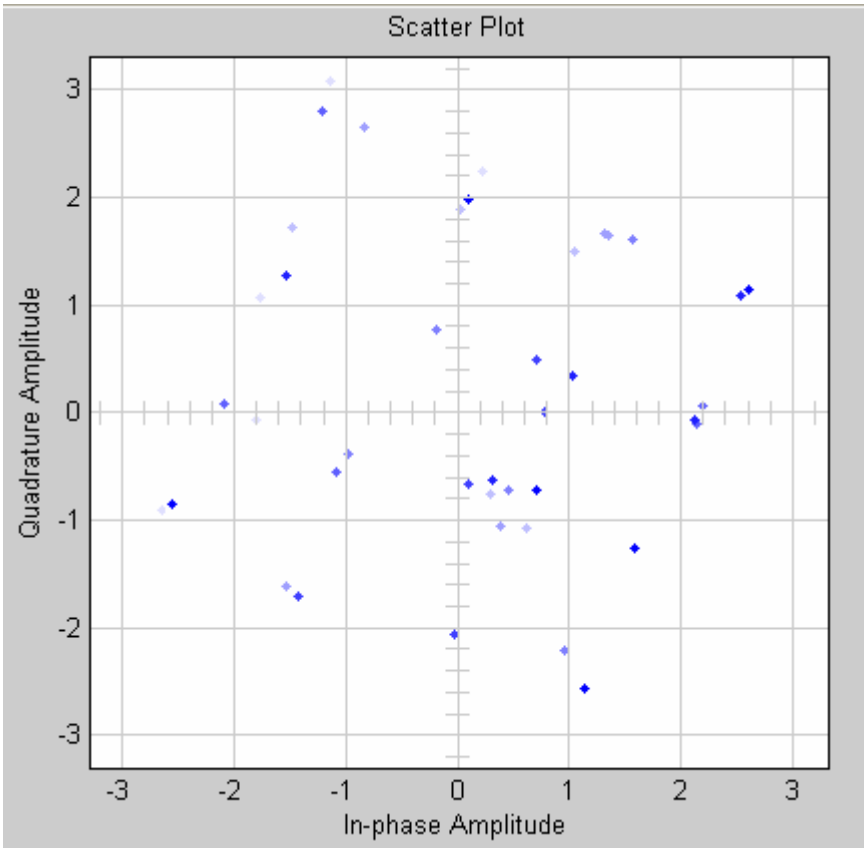


Figura 7.8: Constelación recibida con sui 6

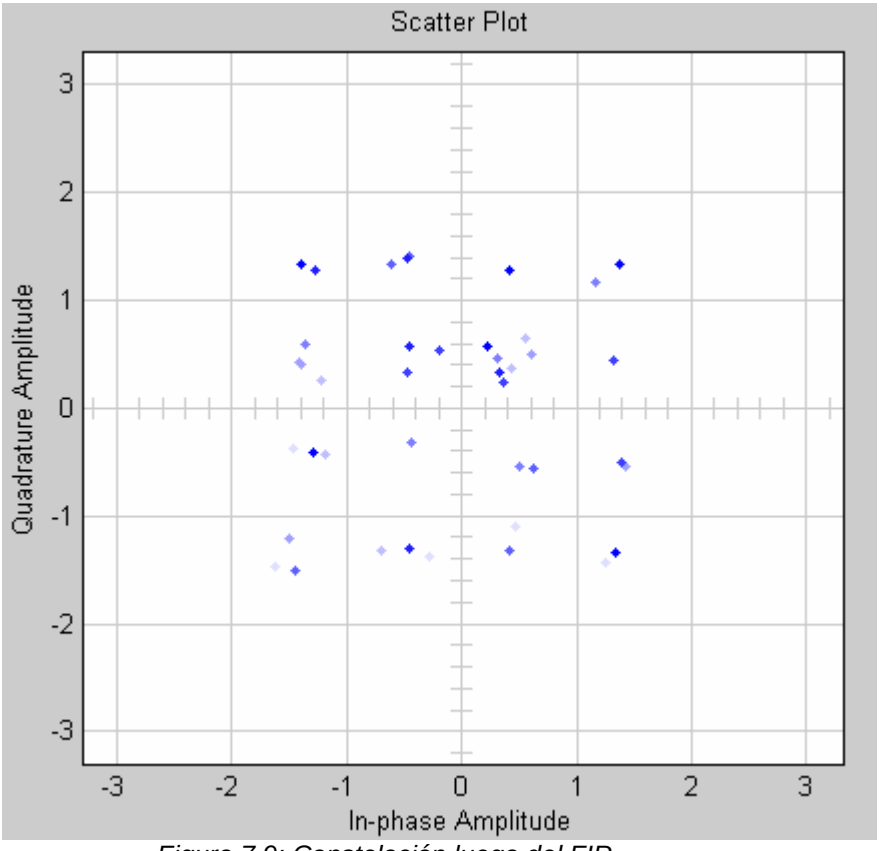


Figura 7.9: Constelación luego del FIR

Como se puede apreciar el sistema corrige la rotación de la constelación, hecho que permite una mejor recepción de la señal. Estos resultados coinciden con lo esperado ya que el ecualizador debía tener un mejor desempeño que el receptor RAKE.

7.4 Comparación

La figura siguiente muestra la relación entre SNR y BER para los tres sistemas implementados en el caso de un canal SUI6. Efectivamente el sistema con ecualizador obtiene mejores resultados frente al receptor Rake y por supuesto frente al sistema simple.

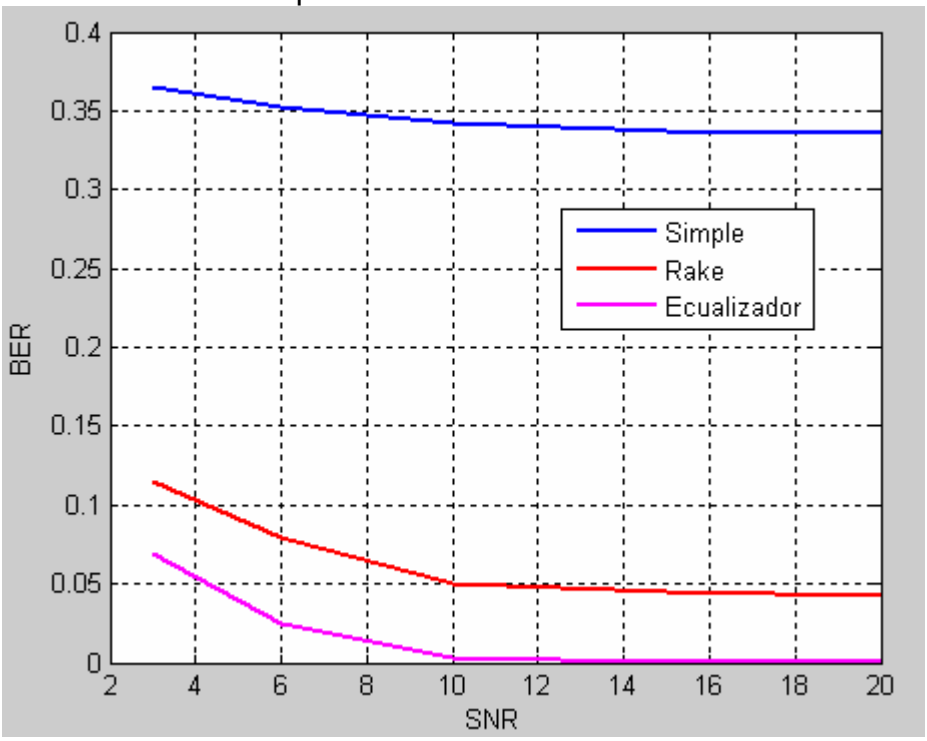


Figura 7.10: Comparación

En la gráfica se aprecia una importante diferencia entre el sistema simple y las nuevas implementaciones, tal como se había visto en los puntos anteriores.

8 Conclusiones

En este trabajo se abordaron las normas del 3GPP sobre la tecnología celular HSDPA (Release 5). Las mismas presentan las especificaciones necesarias para el diseño de un transmisor HSDPA que cumpla con el estándar. Se estudiaron dichas normas y se presenta un resumen sobre el trabajo del 3GPP relativo a las especificaciones para la capa física de esta tecnología.

Se diseñaron e implementaron tres sistemas receptores para el transmisor que especifica la norma, transmitiendo en todos los casos a la máxima tasa de datos teórica de 14,4 Mbps.

En primera instancia, se analizó el sistema simple, o sea, el transmisor especificado en la norma y su inverso bloque a bloque. Se comprueba que el receptor diseñado puede establecer una comunicación en un entorno afectado solamente por ruido blanco. Sin embargo es imposible utilizar este receptor en un entorno más realista, con un modelo de canal multicaminos.

A partir de esta comprobación, se buscaron sistemas más complejos y robustos frente a un canal con múltiples trayectos.

Se analizó el sistema con un receptor RAKE. Se encontró una mejora importante respecto al sistema simple, pero no lo suficientemente satisfactoria como para considerar una comunicación exitosa. Posteriormente pudo verificarse con literatura disponible que el receptor RAKE no es el sistema más adecuado para trabajar en esta tecnología. En el punto 7.3 pueden encontrarse las referencias al respecto.

Finalmente, se analizó el desempeño con un ecualizador. En este caso no fue posible correr las simulaciones para el mismo tiempo que los demás sistemas por presentarse un problema con la implementación que no se pudo llegar a resolver. De todas maneras, se implementó un sistema ecualizador que corre durante un tiempo reducido y presenta resultados mejorados.

Este sistema ecualizador puede detectar la señal en un ambiente multicamino. Por lo que se puede concluir que la tecnología es capaz de establecer una comunicación a la máxima velocidad teórica de 14,4 Mbps, incluso en un entorno de propagación real y no solo de laboratorio.

Los resultados obtenidos concuerdan con la literatura investigada, aunque no se pudo encontrar ningún caso de simulaciones que utilicen las mismas características en la transmisión (velocidad, modulación, cantidad de canales) o el mismo modelo de propagación. De todas maneras se verifica que el ecualizador es el sistema recomendado y con el cual se obtienen los mejores resultados para HSDPA.

Como comentario final, nos permitimos nombrar algunas posibilidades de trabajos futuros para mejorar los sistemas implementados en este proyecto,

teniendo como base el mismo, o incluso nuevos sistemas que obtengan una mejor performance como receptores de la tecnología HSDPA:

- Algoritmo RLS. Existen variedades de este algoritmo, utilizado en el ecualizador para actualizar coeficientes. Puede ser de interés analizar el uso de los diferentes algoritmos.

- MIMO. Mientras se investigaba sobre ecualizadores se encontró mucho material sobre MIMO aplicado a HSDPA. [ ][ ] La tecnología MIMO se puede agregar a la modulación adaptiva de HSDPA para obtener mejores resultados.

- Ecualizadores. Existen muchos diseños de ecualizadores que lamentablemente no se pudieron evaluar por un tema de tiempo, pero en algunos casos son presentados como una mejora al ecualizador RLS.

Es nuestra aspiración que el presente proyecto represente una plataforma adecuada para posteriores investigaciones que permitan profundizar en el análisis de la tecnología que nos ocupa.

Somos conscientes que las previsiones de investigación y obtención de evidencias proyectadas no han podido ser satisfechas integralmente, sin embargo, entendemos que la inversión de esfuerzo, tiempo y dedicación aplicados han permitido alcanzar conclusiones relevantes y esperamos que útiles para la comunidad científica y tecnológica que aspiramos a integrar.

ANEXOS

A. Sobre especificación de capa física

A.1 Algoritmo de Segmentación

Los bits de salida de la segmentación de bloque de código, son denotados por $o_{ir1}, o_{ir2}, o_{ir3}, \ldots, o_{rK}$,
donde *i* es el número de TrCH, *r* es el número de bloque de código y *K* es la cantidad de bits por código de bloque.

Número de bloques de código de TrCH *i:*

$$C_i = \lceil B \ / Z \rceil$$

El siguiente algoritmo explica la segmentación:

if $X_i < 40$ and Turbo coding is used, then

$K = 40$

else

$K = \lceil X_i / C_i \rceil$

end if

Número de bits de relleno: $Y_i = C_i K - X_i$

for $k = 1$ to Y_i -- Inserción de bits de relleno

$o_{i1k} = 0$

end for

for $k = Y_i + 1$ to K

$o_{i1k} = x_{i,(k-Y_i)}$

end for

$r = 2$ -- Segmentación

while $r \leq C_i$

for $k = 1$ to K

$$o_{irk} = x_{i,(k+(r-1)\cdot K - Y_i)}$$

end for

$r = r+1$

end while

A.2 Intercalador interno de código Turbo

Los bits de entrada al intercalador interno de código Turbo son denotados por $x_1, x_2, x_3, \ldots, x_K$, donde *K* es el número entero de bits y toma un valor de 40 ≤ *K* ≤ 5114. La relación entre los bits de entrada al intercalador interno de código Turbo y los bits de entrada a la codificación de canal son definidos por $x_k = o_{irk}$.

Los siguientes símbolos son usados en las sub-cláusulas 3.4.1.4:

K	Número de bits de entrada al intercalador interno de código Turbo
R	Número de filas de la matriz rectangular
C	Número de columnas de la matriz rectangular
p	Número primo
v	Raíz primitiva
$\langle s(j) \rangle_{j \in \{0,1,\cdots,p-2\}}$	Secuencia base para permutación intra-fila
q_i	Mínimos enteros primos
r_i	Enteros primos permutados
$\langle T(i) \rangle_{i \in \{0,1,\cdots,R-1\}}$	Patrón de permutación inter-fila
$\langle U_i(j) \rangle_{j \in \{0,1,\cdots,C-1\}}$	Patrón de permutación intra-fila para la fila *i*-ésima
i	Índice del número de fila de la matriz rectangular
j	Índice del número de columna de la matriz rectangular
k	Índice de la secuencia de bits

Bits de entrada a la matriz rectangular con relleno

La secuencia de bits $x_1, x_2, x_3, \ldots, x_K$ que entra al intercalador interno de código Turbo es escrita en la matriz rectangular como sigue:

(1) Se determina el número de filas de la matriz rectangular, *R*, tal que:

$$R = \begin{cases} 5, \text{si } (40 \le K \le 159) \\ 10, \text{si } ((160 \le K \le 200) \text{ o } (481 \le K \le 530)) \\ 20, \text{si } (K = \text{cualquier otro valor}) \end{cases}$$

.

Las filas de la matriz rectangular son numeradas 0, 1, …, *R* - 1 de arriba hacia abajo.

(2) Se determina el número primo a ser utilizado en la permutación intra, *p*, y el número de columnas de la matriz rectangular, *C*, tal que:

if ($481 \le K \le 530$) then

p = *53* and *C* = *p*.

else

Find minimum prime number *p* from table 2 such that

$$K \le R \times (p+1),$$

and determine *C* such that

$$C = \begin{cases} p-1 & if \quad K \le R \times (p-1) \\ p & if \quad R \times (p-1) < K \le R \times p \\ p+1 & if \quad R \times p < K \end{cases}.$$

end if

Las columnas de la matriz rectangular son numeradas 0, 1, …, *C* - 1 de izquierda a derecha.

Tabla 2: Lista de número primo *p* y raíz primitiva asociada *v*

p	***v***	***p***	***v***	***p***	***v***	***p***	***v***	***p***	***V***
7	3	47	5	101	2	157	5	223	3
11	2	53	2	103	5	163	2	227	2
13	2	59	2	107	2	167	5	229	6
17	3	61	2	109	6	173	2	233	3
19	2	67	2	113	3	179	2	239	7
23	5	71	7	127	3	181	2	241	7
29	2	73	5	131	2	191	19	251	6
31	3	79	3	137	3	193	5	257	3
37	2	83	2	139	2	197	2		
41	6	89	3	149	2	199	3		
43	3	97	5	151	6	211	2		

(3)Se escribe la secuencia de bits de entrada $x_1, x_2, x_3, \ldots, x_K$ en la matriz rectangular $R \times C$ fila por fila comenzando con el bit y_1 en la columna 0 de la fila 0:

$$\begin{bmatrix} y_1 & y_2 & y_3 & \cdots & y_C \\ y_{(C+1)} & y_{(C+2)} & y_{(C+3)} & \cdots & y_{2C} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ y_{((R-1)C+1)} & y_{((R-1)C+2)} & y_{((R-1)C+3)} & \cdots & y_{R \times C} \end{bmatrix}$$

donde $y_k = x_k$ para k = 1, 2, …, K y si $R \times C > K$, los bits postizos son rellenados tal que $y_k = 0 or 1$ para k = K + 1, K + 2, …, $R \times C$. Estos bits postizos son podados de la salida de la matriz rectangular luego de las permutaciones intra e inter fila.

Permutaciones inter e intra fila

Luego que los bits entran a la matriz rectangular $R \times C$, las permutaciones inter e intra fila para la matriz rectangular $R \times C$ se realizan paso a paso siguiendo el algoritmo de los pasos (1) a (6):

(1) Se elige una raíz primitiva v de la tabla 2, la cual se indica a la derecha del número primo p.

(2) Se construye la secuencia base $\langle s(j) \rangle_{j \in \{0,1,\cdots,p-2\}}$ para la permutación inter fila de esta forma:
$s(j) = (v \times s(j-1)) \bmod p$, j = 1, 2,…, (p - 2), y $s(0) = 1$.

(3) Se asigna q_0 = 1 para que sea el primer entero primo en la secuencia $\langle q_i \rangle_{i \in \{0,1,\cdots,R-1\}}$ y se determina el entero primo q_i en la secuencia $\langle q_i \rangle_{i \in \{0,1,\cdots,R-1\}}$ para que sea un entero primo tal que el mayor divisor común entre q_i y p – 1 sea uno, $q_i > 6$, y $q_i > q_{(i-1)}$ para cada i = 1, 2, …, R – 1.

(4) Se permuta la secuencia $\langle q_i \rangle_{i \in \{0,1,\cdots,R-1\}}$ para hacer la secuencia $\langle r_i \rangle_{i \in \{0,1,\cdots,R-1\}}$ tal que $r_{T(i)} = q_i$, i = 0, 1, …, R - 1, donde $\langle T(i) \rangle_{i \in \{0,1,\cdots,R-1\}}$ es el patrón de permutación inter-fila definido como uno de los cuatro tipos de patrones, que son mostrados en la tabla 3, dependiendo de la cantidad de bits de entrada K.

Tabla 3: Patrones de permutación inter-fila para el intercalador de código Turbo

Cantidad de bits de entrada K	Cantidad de filas R	Patrones de permutación inter-fila <$T(0)$, $T(1)$, …, $T(R-1)$>
$(40 \le K \le 159)$	5	<4, 3, 2, 1, 0>
$(160 \le K \le 200)$ o $(481 \le K \le 530)$	10	<9, 8, 7, 6, 5, 4, 3, 2, 1, 0>
$(2281 \le K \le 2480)$ o $(3161 \le K \le 3210)$	20	<19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 16, 13, 17, 15, 3, 1, 6, 11, 8, 10>
K = cualquier otro valor	20	<19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 10, 8, 13, 17, 3, 1, 16, 6, 15, 11>

(5) Se realiza la permutación i-ésima (i = 0, 1, …, R - 1) como sigue:

If ($C = p$) then

$$U_i(j) = s((j \times r_i) \bmod (p-1)), \quad j = 0, 1, \ldots, (p-2), \text{ y } U_i(p-1) = 0,$$

donde $U_i(j)$ es la posición original del bit permutado j de la fila i.

end if

If ($C = p + 1$) then

$$U_i(j) = s((j \times r_i) \bmod (p-1)), \quad j = 0, 1, \ldots, (p-2). \; U_i(p-1) = 0, \text{ y } U_i(p) = p,$$

donde $U_i(j)$ es la posición original del bit permutado j de la fila i

And if ($K = R \times C$) then

Exchange $U_{R-1}(p)$ with $U_{R-1}(0)$.

End if

End if

If ($C = p - 1$) then

$$U_i(j) = s((j \times r_i) \bmod (p-1)) - 1, \quad j = 0, 1, \ldots, (p-2),$$

donde $U_i(j)$ es la posición original del bit permutado j de la fila i.

end if

(6) Se realiza la permutación inter-fila para la matriz rectangular basada en el patrón $\langle T(i) \rangle_{i \in \{0,1,\cdots,R-1\}}$, donde $T(i)$ es la posición original de la fila permutada i.

Salida de los bits de la matriz rectangular con borrado

Luego de las permutaciones inter e intra fila, los bits de la matriz rectangular permutada son denotados por y'_k:

$$\begin{bmatrix} y'_1 & y'_{(R+1)} & y'_{(2R+1)} & \cdots & y'_{((C-1)R+1)} \\ y'_2 & y'_{(R+2)} & y'_{(2R+2)} & \cdots & y'_{((C-1)R+2)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ y'_R & y'_{2R} & y'_{3R} & \cdots & y'_{C \times R} \end{bmatrix}$$

La salida del intercalador de código Turbo es la secuencia de bits leída columna por columna de la matriz rectangular $R \times C$ con permutaciones inter e intra fila comenzando con el bit y'_1 en la fila 0 de la columna 0 y terminando con el bit y'_{CXR} en la fila R-1 de la columna C-1. La salida es cortada borrando los bits de relleno que fueron agregados a la entrada de la matriz rectangular antes de las permutaciones inter e intra fila o sea los bits y'_k que corresponden con los bits y_k con $k > K$ son removidos de la salida. Los bits que salen del intercalador interno de código turbo son denotados por x'_1, x'_2, …, $x'_{K,}$, donde x'_1 corresponde al bit y'_k con el menor índice k luego de la poda, x'_2 al bit y'_k con el segundo menor índice k luego de la poda y así en adelante. La cantidad de bits que salen del intercalador de código interno turbo es K y la cantidad total de bits podados es:

$R \times C - K$.

A la salida del bloque, obtenemos $c_{i1}, c_{i2}, ..., c_{iE}$ tal que E=Yi y

A.3 Algoritmos de Rate Matching

Primera Etapa de Rate Matching

El proceso es el siguiente:

Si se va a realizar pinchaje, los parámetros siguientes deben ser utilizados. El índice b es usado para indicar bits sistemáticos (b=1), primera paridad (b=2) y segunda paridad (b=3).

a=2 cuando b=2
a=1 cuando b=3
Los bits indicados por b=1 no deben ser pinchados.

$$\Delta N_i = \begin{cases} \left\lfloor \Delta N_{il}^{TTI} / 2 \right\rfloor, & b = 2 \\ \left\lceil \Delta N_{il}^{TTI} / 2 \right\rceil, & b = 3 \end{cases}$$

Para cada TTI el patrón de rate-matching se calcula con el algoritmo que se presentará a continuación. Los siguientes parámetros serán utilizados como entradas:

$$X_i = N_{il}^{TTI} / 3,$$
$$e_{ini} = X_i,$$
$$e_{plus} = a \times X_i$$
$$e_{\min us} = a \times \left| \Delta N_i \right|$$

Algoritmo para obtener el patrón de rate matching

Los bits luego del rate matching son $x_{i1}, x_{i2}, x_{i3}, \ldots, x_{iX_i}$, donde *i* es el número de TrCH. Los parámetros X_i, e_{ini}, e_{plus}, y e_{minus} son los anteriores.

La regla de rate matching es la siguiente:

```
if puncturing is to be performed (si debe realizarse pinchaje)
    e = e_ini  -- error inicial entre el cociente de pinchaje deseado y actual
    m = 1                  -- índice del bit actual
    do while m <= X_i
        e = e – e_minus                    -- actualizar error
        if e <= 0 then                     -- chequear si el bit m debe ser pinchado
            set bit x_i,m to δ where δ∉{0, 1}   (fijar el bit x_i,m para luego diferenciarlo y quitarlo)
            e = e + e_plus     -- actualizar error
        end if
        m = m + 1                          --  próximo bit
    end do
else
    e = e_ini                  -- error inicial entre el cociente de pinchaje deseado y actual
    m = 1                      -- índice del bit actual
    do while m <= X_i
        e = e – e_minus        -- actualizar error
        do while e <= 0        -- chequear si el bit m debe ser repetido
            repeat bit x_i,m
            e = e +  e_plus    -- actualizar error
        end do
        m = m + 1              --  next bit
    end do
end if
```

Un bit repetido se ubica directamente detrás del original.

Segunda etapa de Rate matching

Los parámetros X_i, e_{plus} y e_{minus} son calculados según las tablas que siguen.

	X_i	e_{plus}	e_{minus}
Sistemáticos RM S	N_{sys}	N_{sys}	$\lvert N_{sys} - N_{t,sys} \rvert$
Paridad 1 RM P1_2	N_{p1}	$2 \cdot N_{p1}$	$2 \cdot \lvert N_{p1} - N_{t,p1} \rvert$
Paridad 2 RM P2_2	N_{p2}	N_{p2}	$\lvert N_{p2} - N_{t,p2} \rvert$

Se define luego del segundo rate matching:

N_{sys}: cantidad de bits sistemáticos
N_{p1}: cantidad de bits paridad 1
N_{p2}: cantidad de bits de paridad 2
P: cantidad de canales físicos
N_{data} = PX3XN_{data1}, la cantidad de bits disponibles para el HS-DSCH en un TTI con N_{data1} obtenido de la tabla siguiente:

Formato de ranura#i	Tasa de bits del canal (kbps)	Tasa de símbolos del canal (ksps)	SF	Bits/ subtrama HS-DSCH	Bits/ Ranura	Ndata1
0(QPSK)	480	240	16	960	320	320
1(16QAM)	960	240	16	1920	640	640

Norma 25.211

Para $N_{data} \le N_{sys} + N_{p1} + N_{p2}$, se realizará pinchaje en la segunda etapa de rate matching. La cantidad de bits sistemáticos transmitidos en una transmisión es $N_{t,sys} = \min\{N_{sys}, N_{data}\}$ para una transmisión que prioriza los bits sistemáticos y $N_{t,sys} = \max\{N_{data} - (N_{p1} + N_{p2}), 0\}$ para una transmisión que prioriza los bits no sistemáticos.
Para $N_{data} > N_{sys} + N_{p1} + N_{p2}$, se realizará repetición en la segunda etapa de rate matching. Una tasa de repetición similar en todos los flujos de bits es lograda fijando el número de bits sistemáticos transmitidos a $N_{t,sys} = \left\lfloor N_{sys} \cdot \frac{N_{data}}{N_{sys} + 2N_{p1}} \right\rfloor$.

La cantidad de bits de paridad en una transmisión es: $N_{t,p1} = \left\lfloor \frac{N_{data} - N_{t,sys}}{2} \right\rfloor$ y $N_{t,p2} = \left\lceil \frac{N_{data} - N_{t,sys}}{2} \right\rceil$ para los bits de paridad 1 y 2, respectivamente.

El parámetro de rate matching e_{ini} es calculado para cada flujo de bits de acuerdo a los parámetros RV r y s usando

$e_{ini}(r) = \left\{\left(X_i - \left\lfloor r \cdot e_{plus} / r_{\max} \right\rfloor - 1\right) \bmod e_{plus}\right\} + 1$ en el caso de pinchaje, o sea, $N_{data} \leq N_{sys} + N_{p1} + N_{p2}$, y
$e_{ini}(r) = \left\{\left(X_i - \left\lfloor (s + 2 \cdot r) \cdot e_{plus} / (2 \cdot r_{\max}) \right\rfloor - 1\right) \bmod e_{plus}\right\} + 1$ para repetición, o sea, $N_{data} > N_{sys} + N_{p1} + N_{p2}$. Donde $r \in \{0,1,\cdots, r_{\max} - 1\}$ y $r_{\max}$ es la cantidad total de versiones de redundancia permitidas al variar r. r_{max} varia dependiendo del tipo de modulación o sea, para 16QAM r_{max} = 2 y para QPSK r_{max} = 4.

Recolección de bits HARQ:

La recolección de bits HARQ es lograda usando un intercalador rectangular de tamaño N_{fila}xN_{col}.

La cantidad de filas y de columnas es:

N_{fila}=4 para 16QAM y N_{fila}=2 para QPSK

N_{col}=N_{data}/N_{fila} con N_{data} como ya se definió previamente.

Los datos se ingresan en el intercalador columna por columna.

$N_{t,sys}$ es la cantidad de bits sistemáticos transmitidos. Los valores intermedios N_r y N_c se calculan usando:

$$N_r = \left\lfloor \frac{N_{t,sys}}{N_{col}} \right\rfloor \text{ y } N_c = N_{t,sys} - N_r \cdot N_{col}.$$

Si N_c=0 y N_r > 0, los bits sistemáticos se ingresan en las filas 1…N_r.
Sino los bits sistemáticos se escriben en las filas 1…N_r+1 en las primeras N_c columnas y, si N_r > 0, también en las filas 1…N_r en las restantes N_{col}-N_c columnas.

El espacio restante se llena con bits de paridad. Los bits de paridad son escritos columna por columna en las filas restantes de las respectivas columnas. Los bits de paridad 1 y 2 se escriben en orden alternado, comenzando con un bit de paridad 2 en la primer columna disponible con el menor índice.

En el caso de 16QAM para cada columna los bits se leen hacia fuera del intercalador en el orden fila1, fila2, fila3, fila 4. En el caso de QPSK para cada columna los bits se leen hacia fuera del intercalador en el orden fila1, fila2.

A la salida del bloque tenemos w_1, w_2, w_3,…w_R.

B. Códigos

B.1 Códigos OVSF

Los códogos de canalización son códigos OVSF que mantienen la ortogonalidad entre los diferentes canales físicos de los usuarios. Los códigos OVSF pueden definirse usando un árblo de código como muestra la figura B.1

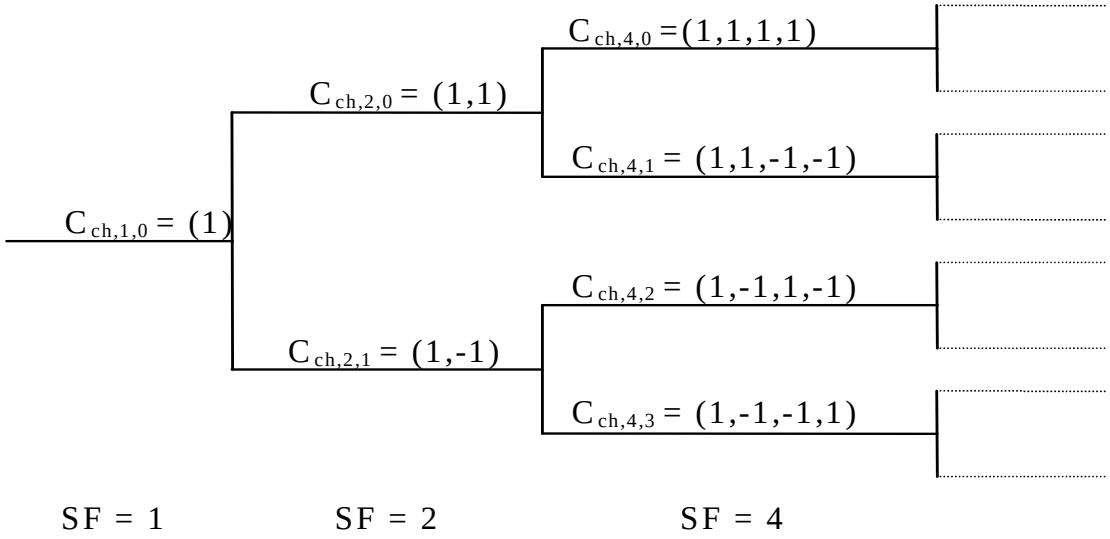


Figure B.1: Árbol de códigos para la generación de códigos OVSF

Los códigos de canalización son denominados uncamente como $C_{ch,SF,k}$, donde SF es el spreading factor del código y k en el número de código, $0 \le k \le$ SF-1. Cada nivel en el árbol de código define códigos de canalización de longitud SF, correspondientes a un spreading factor SF.

El método de generación de los códigos de canalización está definido de la siuiente fomra:

$$C_{ch,1,0} = 1,$$

$$\begin{bmatrix} C_{ch,2,0} \\ C_{ch,2,1} \end{bmatrix} = \begin{bmatrix} C_{ch,1,0} & C_{ch,1,0} \\ C_{ch,1,0} & -C_{ch,1,0} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} C_{ch,2^{(n+1)},0} \\ C_{ch,2^{(n+1)},1} \\ C_{ch,2^{(n+1)},2} \\ C_{ch,2^{(n+1)},3} \\ \vdots \\ C_{ch,2^{(n+1)},2^{(n+1)}-2} \\ C_{ch,2^{(n+1)},2^{(n+1)}-1} \end{bmatrix} = \begin{bmatrix} C_{ch,2^n,0} & C_{ch,2^n,0} \\ C_{ch,2^n,0} & -C_{ch,2^n,0} \\ C_{ch,2^n,1} & C_{ch,2^n,1} \\ C_{ch,2^n,1} & -C_{ch,2^n,1} \\ \vdots & \vdots \\ C_{ch,2^n,2^n-1} & C_{ch,2^n,2^n-1} \\ C_{ch,2^n,2^n-1} & -C_{ch,2^n,2^n-1} \end{bmatrix}$$

B.2 Códigos de scrambling

Pueden generarse un total de 2^{18}-1 = 262143 códigos, numerados 0…262142, aunque no se utilizan todos. Se definen 512 grupos de códigos, donde cada grupo cuenta con un código primario y quince códigos secundarios.

Son definidos de la siguiente forma:

los códigos numerados n=i*16 , con i=0…511 serán los códigos primarios
los códigos numerados n=i*16 + k con k=1..15 serán los códigos secundarios

O sea que el código 0 es un código primario, con sus quince códigos secundarios numerados del uno al quince. El código 16 es otro código primario con sus quince secundarios que van desde el código 17 al 31, etc.

Cada celda tendrá un código primario asignado y solo uno, y además contará con sus quince códigos secundarios.

Para los canales CCPCH primario, CPICH primario, PICH, AICH and S-CCPCH se utiliza siempre el código de scrambling primario. Los demás canales físicos de bajada pueden ser transmitidos con el código primario o con cualquiera de los códigos secundarios del grupo correspondiente.

El código de scrambling se construye a partir de dos secuencias reales que se convierten en una secuencia compleja para formar la secuencia del código de scrambling. Esta secuencia se repite cada 10ms.

Llamamos x e y a las dos secuencias reales, entonces x e y se construyen de la siguiente forma:

Condiciones iniciales:

- x (0)=1, x(1)= x(2)=...= x (16)= x (17)=0.
- y(0)=y(1)= … =y(16)= y(17)=1.

Los siguientes valores se definen recursivamente:

x(i+18) =x(i+7) + x(i) modulo 2, i=0,…,2^{18}-20.

y(i+18) = y(i+10)+y(i+7)+y(i+5)+y(i) modulo 2, i=0,…, 2^{18}-20.

Llamamos z_n a la secuencia correspondiente al código de scrambling *n*, con *n=0...*2^{18}-2

z_n(i) = x((i+n) modulo (2^{18} - 1)) + y(i) modulo 2, i=0,…, 2^{18}-2.

Esta secuencia es una secuencia de bits, que se convierte a una secuencia real mediante la siguiente transformación:

$$Z_n(i) = \begin{cases} +1 & si\ z_n(i) = 0 \\ -1 & si\ z_n(i) = 1 \end{cases} \quad para \quad i = 0,1,\ldots,2^{18} - 2.$$

Finalmente la secuencia correspondiente al código de scrambling n, $S_{dl,n}$ queda definida de esta forma

$$S_{dl,n}(i) = Z_n(i) + j\, Z_n((i+131072) \text{ modulo } (2^{18}-1)),\ i=0,1,\ldots,38399.$$

La figura muestra la configuración del generador de los códigos de scrambling:

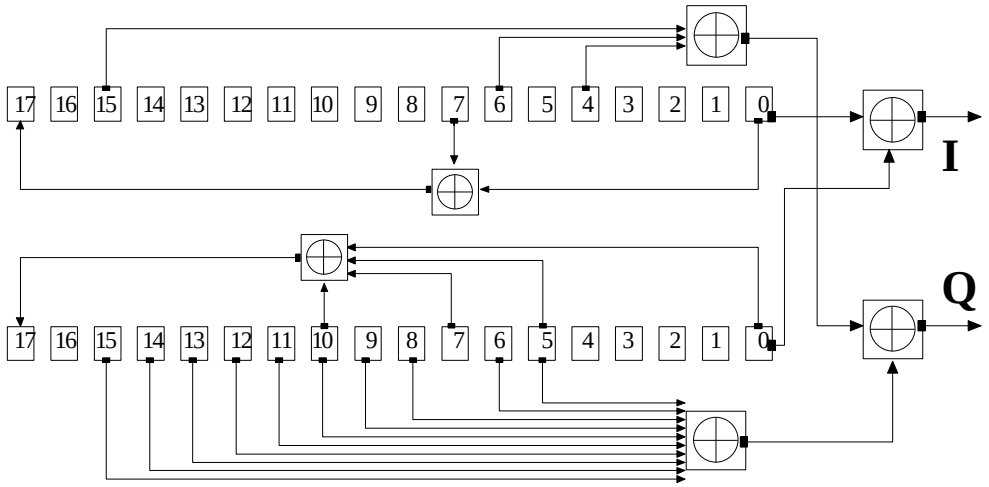


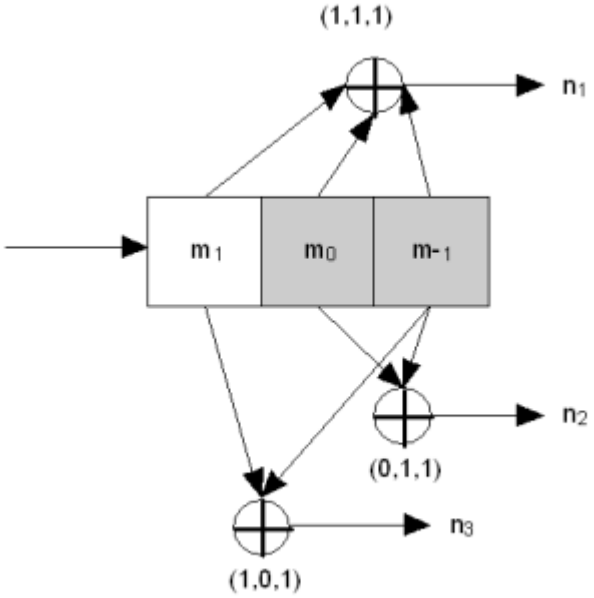
Figura 3.10.2 – Generador de códigos scrambling

C. Codificación Convolucional

Para codificar datos convolucionalmente, se comienza con *k* registros de memoria, cada uno teniendo 1 bit de entrada. A menos que se especifique lo contrario, todos los registros de memoria empiezan con un valor de 0. El codificador tiene *n* sumadores modulo 2 y *n* polinomios generadores – uno por cada sumador (ver figura abajo) Un bit de entrada m_1 es alimentado al registro más a la izquierda. Usando polinomios generadores y los valores existentes en los registros restantes, el codificador da como salida *n* bits. Ahora se corren todos los valores del registro a la derecha y se espera al próximo bit de entrada. Si no quedan más bits, el codificador continúa con la salida hasta que todos los registros hayan regresado a su estado inicial.

La figura inferior es un codificador con tasa 1/3 con longitud de restricción (*k*) igual a 3. Los polinomios generadores son G_1 = (1,1,1), G_2 = (0,1,1) y G_3 = (1,0,1). Por lo tanto, los bits de salida son calculados (módulo 2) como sigue:

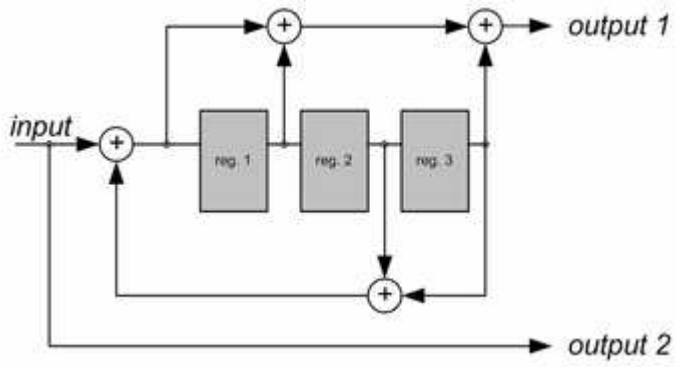
$n_1 = m_1 + m_0 + m_{-1}$
$n_2 = m_0 + m_{-1}$
$n_3 = m_1 + m_{-1}$.



Codificador convolucional con tasa 1/3 no-recursivo, no-sistemático con longitud de restricción 3

C.1 Códigos recursivos y no-recursivos

El codificador de la figura superior es un codificador *no-recursivo*. Aquí hay un ejemplo de uno recursivo:



Codificador convolucional con tasa 1/2 recursivo, sistemático con longitud de restricción 4

Se puede ver que la entrada a ser codificada está incluida también en la secuencia de salida. Tales códigos son referidos como sistemáticos, de otra forma al código se le dice no-sistemático.

Códigos recursivos son casi siempre sistemáticos y a la inversa, códigos no recursivos son no-sistemáticos. No es un requerimiento estricto pero es una práctica común.

C.2 Respuesta al impulso, función de transferencia y longitud de restricción

Un codificador convolucional realiza una *convolución* del flujo de entrada con las *respuestas al impulso* del codificador:

$$y_i^j = \sum_{k=0}^{\infty} h_k^j x_{i-k},$$

donde *x* es una secuencia de entrada, y^j es una secuencia de la salida *j* y h^j es una respuesta al impulso de la salida *j*.

Un codificador convolucional es un sistema discreto, lineal e invariante en el tiempo. Cada salida de un codificador puede ser descrita por su propia función de transferencia, lo que está relacionado muy de cerca con un polinomio generador. Una respuesta al impulso está conectada con una función de transferencia a través de la transformada Z.

Funciones de transferencia para el primer codificador (no-recursivo) son:

- $H_1(z) = 1 + z^{-1} + z^{-2}$,
- $H_2(z) = z^{-1} + z^{-2}$,
- $H_3(z) = 1 + z^{-2}$.

Funciones de transferencia para el segundo codificador (recursivo) son:

- $$H_1(z) = \frac{1 + z^{-1} + z^{-3}}{1 + z^{-2} + z^{-3}},$$
- $H_2(z) = 1$.

Se define *m* por

$$m = \max_i \text{ polydeg } (H_i(1/z))$$

donde, para cualquier función racional $f(z) = P(z)/Q(z)$,

$$\text{polydeg}(f) = \max (\deg(P), \deg(Q)).$$

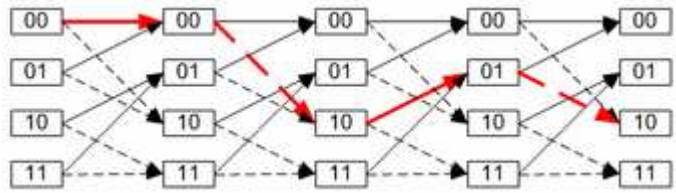
Entonces *m* es el máximo de los grados del polinomio de $H_i(1/z)$, y la *longitud de restricción* es definida como $K = m + 1$. Por ejemplo, en el primer ejemplo la longitud de restricción es 3 y en el segundo la longitud de restricción es 4.

C.3 Diagrama de Trellis

Un codificador convolucional es una máquina de estados finitos. Un codificador con *n* celdas binarias tendrá 2^n estados.

Si el codificador tiene ‘1’ en su celda izquierda de memoria (m_0), y ‘0’ en la derecha (m_{-1}). Designaremos tal estado como "10". Dependiendo del bit de entrada en el próximo turno se puede convertir en el estado "01" o el estado "11". Se puede ver que no todas las transiciones son posibles (por ej., un decodificador no puede convertir de un estado "10" a un estado "00" o incluso permanecer en un estado "10").

Todas las transiciones posibles pueden mostrarse como en la próxima figura:



Img.3. Diagrama de Trellis para el codificador de la Img.1

Una secuencia codificada real puede ser representada como un camino en esta gráfica. Un camino válido es mostrado en rojo como un ejemplo.

Este diagrama nos da una idea sobre *decodificar*: si una secuencia recibida no encaja con esta gráfica, entonces fue recibida con errores y se debe elegir la secuencia *correcta* más cercana (encajando con esta gráfica). Los algoritmos reales de decodificación explotan esta idea.

C.4 Decodificando códigos convolucionales

Existen varios algoritmos para decodificar códigos convolucionales. Para valores de *k* relativamente pequeños, el algoritmo Viterbi es usado universalmente ya que provee performance de máxima probabilidad y es altamente paralelisable. Decodificadores Viterbi son por lo tanto fáciles de implementar.

D. Algoritmo RLS [19]

El algoritmo RLS, Recursive-Least-Squares algorithm, se usa en filtros adaptativos para encontrar los coeficientes del filtro que permiten obtener el mínimo cuadrado de la señal de erroren forma recursiva. Esto en contraste con otros algoritmos que apuntan a reducir el error cuadrático medio. La diferencia es que los filtros RLS dependen de las señales, mientras los filtros MSE son dependientes en sus estadísticas. Si estas estadísticas son conocidas, un filtro MSE con coeficientes fijados (o sea, independientes de los datos de entrada) puede ser armado.

Supongamos que una señal d es transmitida sobre un canal ruidoso y con eco que causa que se reciba como:

$$x(n) = \sum_{k=0}^{q} b_n(k) d(n-k) + v(n)$$

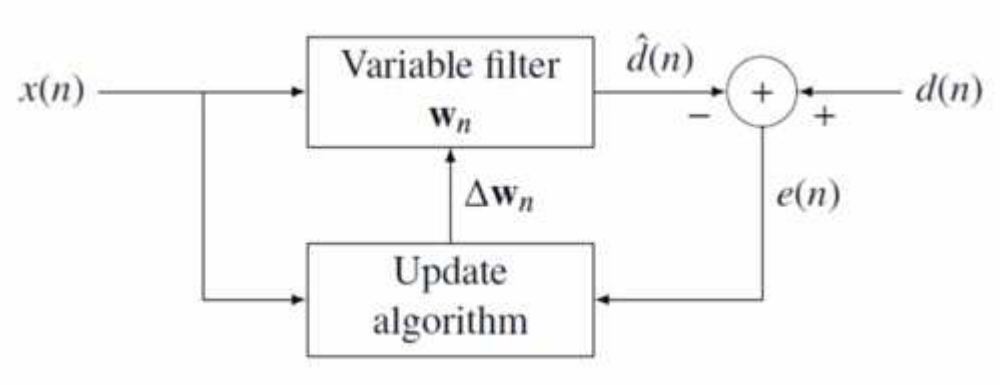
donde *v*(*n*) representa ruido blanco. Intentaremos recuperar la señal deseada d con el uso de un filtro FIR, w:

$$\hat{d}(n) = y(n) = \mathbf{w}_n^T \mathbf{x}(n)$$

Nuestra meta es estimar los parámetros del filtro w, y en cada tiempo n nos referimos al nuevo estimado de mínimos cuadrados w_n. Mientras el tiempo evoluciona, nos gustaría evitar completamente volver a hacer el algoritmo de los mínimos cuadrados para encontrar el nuevo estimado para w_{n+1}, en términos de w_n.

El beneficio del algoritmo RLS es que no hay necesidad de invertir matrices, por lo tanto se ahorra potencia computacional. Otra ventaja es que provee intución detrás de tales resultados como los filtros Kalman.

La idea detrás de los filtros RLS es minimizar una función de costo C al elegir apropiadamente los coeficientes del filtro w_n, actualizando el filtro mientras los nuevos datos llegan. La señal de error *e*(*n*) y la señal deseada *d*(*n*) son definidas como en el diagrama de retroalimentación negativa que se encuentra a continuación:



El error depende implícitamente de los coeficientes del filtro a través del estimado $\hat{d}(n)$:

$$e(n) = d(n) - \hat{d}(n)$$

La función de error de mínimos cuadrados pesados C – la función de costo que deseamos minimizar – es una función de e(n) es por lo tanto dependiente de los coeficientes del filtro:

$$C(\mathbf{w_n}) = \sum_{i=0}^{n} \lambda^{n-i} |e(i)|^2 = \sum_{i=0}^{n} \lambda^{n-i} e(i)\, e^*(i)$$

donde $0 < \lambda \leq 1$ es una factor de peso exponencial que efectivamente limita la cantidad de muestras de entrada basada en cuál es la función de costo minimizada.

La función de costo es minimizada al tomar derivadas parciales para todas las entradas *k* del vector de coeficientes w_n y fijando los resultados en cero

$$\frac{\partial C(\mathbf{w}_n)}{\partial w_n^*(k)} = \sum_{i=0}^{n} \lambda^{n-i} e(i) \frac{\partial e^*(i)}{\partial w_n^*(k)} = \sum_{i=0}^{n} \lambda^{n-i} e(i)\, x^*(i-k) = 0$$

Luego, se reemplaza e(n) con la definición de la señal de error.

$$\sum_{i=0}^{n} \lambda^{n-i} \left[d(i) - \sum_{l=0}^{p} w_n(l) x(i-l) \right] x^*(i-k) = 0$$

Reordenando la ecuación se obtiene

$$\sum_{l=0}^{p} w_n(l) \left[\sum_{i=0}^{n} \lambda^{n-i}\, x(i-l) x^*(i-k) \right] = \sum_{i=0}^{n} \lambda^{n-i} d(i) x^*(i-k)$$

Esta forma puede ser expresada en términos matriciales:

$$\mathbf{R}_x(n)\, \mathbf{w}_n = \mathbf{r}_{dx}(n)$$

donde $\mathbf{R}_x(n)$ es la matriz de autocorrelación ponderada para *x*(*n*) y $\mathbf{r}_{dx}(n)$ es la correlación cruzada entre *d*(*n*) y *x*(*n*). Basado en esta expresión encontramos que los coeficientes que minimizan la función de costo como

$$\mathbf{w}_n = \mathbf{R}_x^{-1}(n)\, \mathbf{r}_{dx}(n)$$

Este es el resultado principal de la discusión.

Eligiendo λ

Cuanto menor es λ, menor es la contribución de las muestras previas. Esto hace que el filtro sea más sensible a las muestras recientes, lo que significa más fluctuaciones en los coeficientes del filtro. El caso de λ=1 es conocido como algoritmo RLS de ventana creciente.

Algoritmo recursivo

La discusión resultó en una ecuación sencilla para determinar el vector de coeficientes que minimiza la función de costo. En esta sección queremos derivar una solución recursiva de la forma

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \Delta \mathbf{w}_{n-1}$$

donde $\Delta \mathbf{w}_{n-1}$ es un factor de corrección en el tiempo *n-1*. Comenzamos la derivación del algoritmo recursivo expresando la correlación cruzada $\mathbf{r}_{dx}(n)$ en términos de $\mathbf{r}_{dx}(n-1)$

$$\begin{aligned}\mathbf{r}_{dx}(n) &= \sum_{i=0}^{n} \lambda^{n-i} d(i)\mathbf{x}^*(i) \\ &= \sum_{i=0}^{n-1} \lambda^{n-i} d(i)\mathbf{x}^*(i) + \lambda^0 d(n)\mathbf{x}^*(n) \\ &= \lambda \mathbf{r}_{dx}(n-1) + d(n)\mathbf{x}^*(n)\end{aligned}$$

donde $\mathbf{x}(i)$ es el vector de datos con dimensión *p+1*

$$\mathbf{x}(i) = [x(i), x(i-1), \ldots, x(i-p)]^T$$

De forma similar expresamos $\mathbf{R}_x(n)$ en términos de $\mathbf{R}_x(n-1)$ de la siguiente manera

$$\begin{aligned}\mathbf{R}_x(n) &= \sum_{i=0}^{n} \lambda^{n-i} \mathbf{x}^*(i)\mathbf{x}^T(i) \\ &= \lambda \mathbf{R}_x(n-1) + \mathbf{x}^*(n)\mathbf{x}^T(n)\end{aligned}$$

Para generar el vector de coeficientes estamos interesados en el inverso de la matriz de autocorrelación determinística. Para esta tarea la identidad matricial Woodbury es útil. Con

A $= \lambda \mathbf{R}_x(n-1)$ es *(p+1)*-por-*(p+1)*

U $= \mathbf{x}^*(n)$ es *(p+1)*-por-1

V $= \mathbf{x}^T(n)$ es 1-por-*(p+1)*

C = I_1 es la matriz identidad 1-por-1

La identidad matricial Woodbury sigue

$$\begin{aligned}\mathbf{R}_x^{-1}(n) = &\left[\lambda \mathbf{R}_x(n-1) + \mathbf{x}^*(n)\mathbf{x}^T(n)\right]^{-1} \\ = &\lambda^{-1}\mathbf{R}_x^{-1}(n-1) \\ &-\lambda^{-1}\mathbf{R}_x^{-1}(n-1)\mathbf{x}^*(n) \\ &\left\{1 + \mathbf{x}^T(n)\lambda^{-1}\mathbf{R}_x^{-1}(n-1)\mathbf{x}^*(n)\right\}^{-1} \mathbf{x}^T(n)\lambda^{-1}\mathbf{R}_x^{-1}(n\end{aligned}$$

Para mantenernos con la literatura estandar, definimos:

$$\mathbf{P}(n) = \mathbf{R}_x^{-1}(n)$$

$$= \lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)$$

donde el vector de ganancia g(n) es

$$\begin{aligned}\mathbf{g}(n) &= \lambda^{-1}\mathbf{P}(n-1)\mathbf{x}^*(n)\left\{1 + \mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)\mathbf{x}^*(n)\right\}^{-1} \\ &= \mathbf{P}(n-1)\mathbf{x}^*(n)\left\{\lambda + \mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}^*(n)\right\}^{-1}\end{aligned}$$

Antes de seguir, es necesario poner g(n) en otra forma

$$\begin{aligned}\mathbf{g}(n)\left\{1 + \mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)\mathbf{x}^*(n)\right\} &= \lambda^{-1}\mathbf{P}(n-1)\mathbf{x}^*(n) \\ \mathbf{g}(n) + \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)\mathbf{x}^*(n) &= \lambda^{-1}\mathbf{P}(n-1)\mathbf{x}^*(n)\end{aligned}$$

Sustrayendo el segundo término en los lados izquierdos se obtiene:

$$\begin{aligned}\mathbf{g}(n) &= \lambda^{-1}\mathbf{P}(n-1)\mathbf{x}^*(n) - \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)\mathbf{x}^*(n) \\ &= \lambda^{-1}\left[\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\mathbf{P}(n-1)\right]\mathbf{x}^*(n)\end{aligned}$$

Con la definición recursiva de $\mathbf{P}(n)$ la forma deseada es

$$\mathbf{g}(n) = \mathbf{P}(n)\mathbf{x}^*(n)$$

Ahora estamos listos para completar la recursión. Como se discutió:

$$\begin{aligned}\mathbf{w}_n &= \mathbf{P}(n)\,\mathbf{r}_{dx}(n) \\ &= \lambda\mathbf{P}(n)\,\mathbf{r}_{dx}(n-1) + d(n)\mathbf{P}(n)\,\mathbf{x}^*(n)\end{aligned}$$

El segundo paso sigue de la definición recursiva de $\mathbf{r}_{dx}(n)$.Luego incorporamos la definición recursiva de $\mathbf{P}(n)$ junto con la forma alternativa de $\mathbf{g}(n)$ y se obtiene

$$\begin{aligned}\mathbf{w}_n &= \lambda\left[\lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)\right]\mathbf{r}_{dx}(n-1) + \\ &= \mathbf{P}(n-1)\mathbf{r}_{dx}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{r}_{dx}(n-1) + \\ &= \mathbf{P}(n-1)\mathbf{r}_{dx}(n-1) + \mathbf{g}(n)\left[d(n) - \mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{r}_{dx}(n\right.\end{aligned}$$

Con $\mathbf{w}_{n-1} = \mathbf{P}(n-1)\mathbf{r}_{dx}(n-1)$ llegaos a la ecuación de actualización

$$\begin{aligned}\mathbf{w}_n &= \mathbf{w}_{n-1} + \mathbf{g}(n)\left[d(n) - \mathbf{x}^T(n)\mathbf{w}_{n-1}\right] \\ &= \mathbf{w}_{n-1} + \mathbf{g}(n)\alpha(n)\end{aligned}$$

donde $\alpha(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}_{n-1}$ es el error a priori. Comparando esto con el error a posteriori; el error que se calcula luego que el filtro es actualizado:

$$e(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}_n$$

Eso significa que encontramos el factor de corrección

$$\Delta\mathbf{w}_{n-1} = \mathbf{g}(n)\alpha(n)$$

Este resultado intuitivamente satisfactorio indica que el factor de corrección es directamente proporcional a ambos el error y el vector de ganancias, que controla cuanta sensibilidad es deseada, a través del factor de peso, λ.

Resumen del algoritmo RLS

El algoritmo RLS para un filtro RLS de orden p puede ser resumido como

Parametros: *p* = orden del filtro
λ = forgetting factor
δ = valor para inicializar $\mathbf{P}(0)$

Initialización: $\mathbf{w}_n = 0$
$\mathbf{P}(0) = \delta^{-1}I$ donde *I es la matriz identidad* (*p* + 1)-por-(*p* + 1)

Computación: Para $n = 0, 1, 2, \ldots$

$$\mathbf{x}(n) = \begin{bmatrix} x(n) \\ x(n-1) \\ \vdots \\ x(n-p) \end{bmatrix}$$

$$\alpha(n) = d(n) - \mathbf{w}(n-1)^T\mathbf{x}(n)$$

$$\mathbf{g}(n) = \mathbf{P}(n-1)\mathbf{x}^*(n)\left\{\lambda + \mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}^*(n)\right\}^{-1}$$

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)$$

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \alpha(n)\mathbf{g}(n).$$

Notese que la recursion para P sigue una ecuación Riccati y por lo tanto tiene un paralelo con el filtro Kalman.

E. Modelo Matemático de canal multicamino [20]

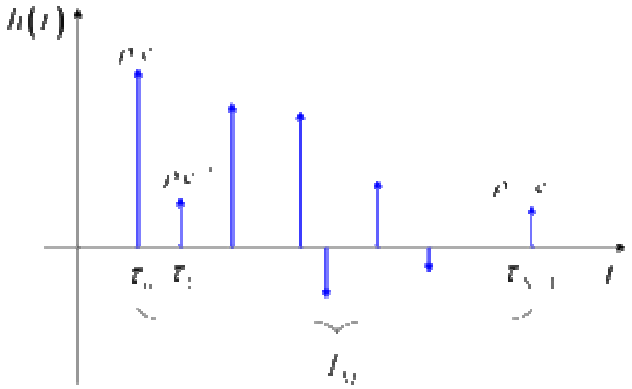


Figura E.1: Modelo matemático de la respuesta al impulso del multicamino

El modelo matemático del multicamino puede ser presentado usando el método de la respuesta al impulso utilizado para estudiar sistemas lineales.

Supongamos que se transmite un pulso de Dirac en el tiempo 0, o sea

$x(t) = \delta(t)$

En el receptor, debido a la presencia de múltiples caminos, más de un pulso será recibido (suponemos aquí que el canal tiene un ancho de banda infinito por lo tanto la forma del pulso no se modifica en lo absoluto), y cada uno de ellos llegará en momentos distintos. De hecho, dado que las señales electromagnéticas viajan a la velocidad de la luz y dado que cada camino tiene un largo geométrico posiblemente diferente del de los demás, hay distintos tiempos de viaje aéreo. Por lo tanto, la señal recibida se expresará como:

$$y(t) = h(t) = \sum_{n=0}^{N-1} \rho_n e^{j\phi_n} \delta(t - \tau_n)$$

donde N es la cantidad de impulsos recibidos (equivalente a la cantidad de pulsos electromagnéticos y posiblemente muy grande), τ_n es el retardo temporal del impulso genérico n, y $\rho_n e^{j\phi_n}$ representa la amplitud compleja (o sea magnitud y fase) del pulso recibido genérico. Como consecuencia, $y(t)$ también representa la función de respuesta al impulso $h(t)$ del modelo equivalente multicamino.

Más en general, en presencia de variación temporal de las condiciones de reflexión geométrica, esta respuesta al impulso es variable en el tiempo y como tal, tenemos:

$\tau_n = \tau_n(t)$

$\rho_n = \rho_n(t)$

$\varphi_n = \varphi_n(t)$

Por lo general solo un parámetro se usa para denotar la severidad de las condiciones del multicamino: se llama tiempo multicamino, T_M, y está definida como el retardo temporal existente entre el primer y el último impulso recibido.

$T_M = \tau_{N-1} - \tau_0$

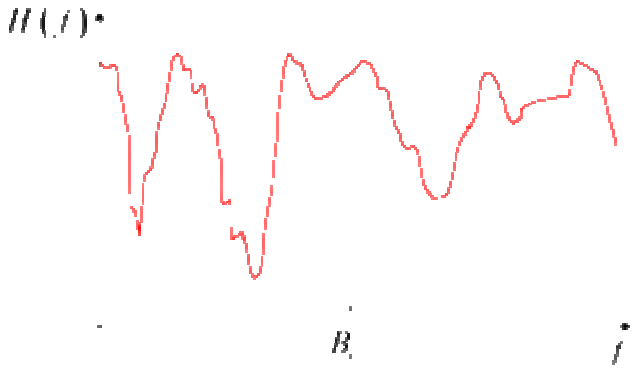


Figura E.2: Modelo matemático de la función de transferencia del canal multicamino

En condiciones prácticas y de medida, el tiempo multicamino es computado considerando como último impulso al primero que permite recibir una determinada cantidad de la potencia total transmitida (dividido por las pérdidas atmosféricas y de propagación), por ejemplo 99%

Manteniendo nuestro objetivo en sistemas lineales, invariantes en el tiempo, podemos caracterizar el fenómeno multicamino con la función de transferencia del canal $H(f)$, la cual está definida como la transformada de Fourier de tiempo continuo de la respuesta al impulso $h(t)$

$$H(f) = \mathfrak{F}(h(t)) = \int_{-\infty}^{+\infty} h(t) e^{-j2\pi ft} dt = \sum_{n=0}^{N-1} \rho_n e^{j\phi_n} e^{-j2\pi f \tau_n}$$

donde el último término a la derecha de la ecuación anterior se obtiene fácilmente al recordar que la transformada de Fourier de Dirac es una función exponencial compleja.

La característica de la transferencia del canal obtenida tiene una apariencia típica de una secuencia de picos y valles; puede mostrarse que, en promedio, la distancia (en Hz) entre dos valles consecutivos (o dos picos consecutivos)

es inversamente proporcional al tiempo multicamino. El asi llamado ancho de banda coherente es entonces definido como

$$B_C \approx \frac{1}{T_M}$$

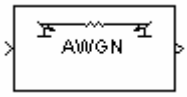
Por ejemplo, con un tiempo multicamino de 3 μs (correspondiendo a 1 km de viaje por aire para el último impulso recibido), hay un ancho de banda coherente de alrededor de 330 kHz.

F. Simulink

Se presentan mas detalles sobre algunos bloques utilizados en la implementación de los sistemas simulados. La información fue obtenida de la documentación de Simulink y se presenta una descripción de los bloques por orden alfabético.

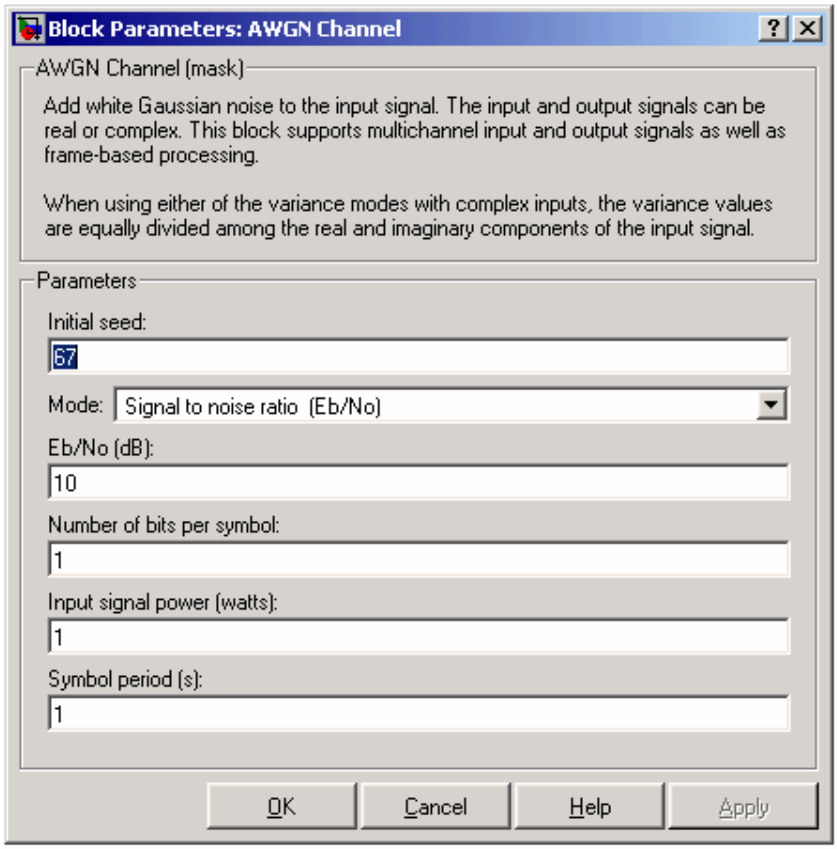
F.1 AWGN

Descripción:



El bloque del canal AWGN agrega ruido blanco gaussiano a una señal de entrada real o compleja. Cuando la señal de entrada es real, este bloque agrega ruido gaussiano real y produce una señal de salida real. Cuando la señal de entrada es compleja, este bloque agrega ruido gaussiano complejo y produce una señal de salida compleja.

Caja de diálogo



Initial Seed (Semilla inicial)
La semilla para el generador de ruido gaussiano

Mode (Modo)
El modo por el cual se especifica la varianza del ruido: Relación señal a ruido (Eb/No), relación señal a ruido (Es/No), relación señal a ruido (SNR), varianza de la máscara o varianza desde puerto.

Eb/No (dB)
La relación de energía de bit por símbolo a ruido, en decibeles. Este campo aparece si el modo se fijó en Eb/No.

Es/No (dB)
La relación de energía de señal por símbolo a ruido, en decibeles. Este campo aparece si el modo se fijó en Es/No.

SNR (dB)
La relación de potencia de señal a potencia de ruido, en decibeles. Este campo aparece si el modo se fijó en SNR.

Number of bits per symbol (Cantidad de bits por símbolo)
La cantidad de bits en cada símbolo de entrada. Este campo aparece si el modo se fijó en Eb/No.

Input signal power (watts) [Potencia de la señal de entrada (watts)]

La raíz cuadrada de la potencia de los símbolos de entrada (si el modo es Eb/No o Es/No) o muestras de entrada (si el modo es SNR), en watts. Este campo aparece si el modo se fijó en Eb/No, Es/No o SNR.

Symbol period (s) [Período de símbolo (s)]
La duración del símbolo de canal, en segundos. Este campo aparece si el modo se fijó en Eb/No o Es/No.

Variance (Varianza)
La varianza del ruido blanco gaussiano. Este campo aparece si el modo se fijó en varianza de máscara

F.2 Codificador Convolucional

Este bloque codifica una secuencia de vector de entrada binario para producir una secuencia de vectores binarios de salida.

Tamaños de entrada y salida

Si el codificador toma k flujos de bits de entrada (esto es, puede recibir 2^k símbolos posibles de entrada), el largo del vector de entrada de este bloque es L*k para algún entero positivo L.
Similarmente, si el codificador produce n flujos de bits de salida (esto es, puede producir 2^n símbolos posibles de salida), el largo del vector de salida es L*n.

La entrada puede ser un vector basado en muestras con L = 1, o un vector columna basado en tramas con cualquier entero positive L.

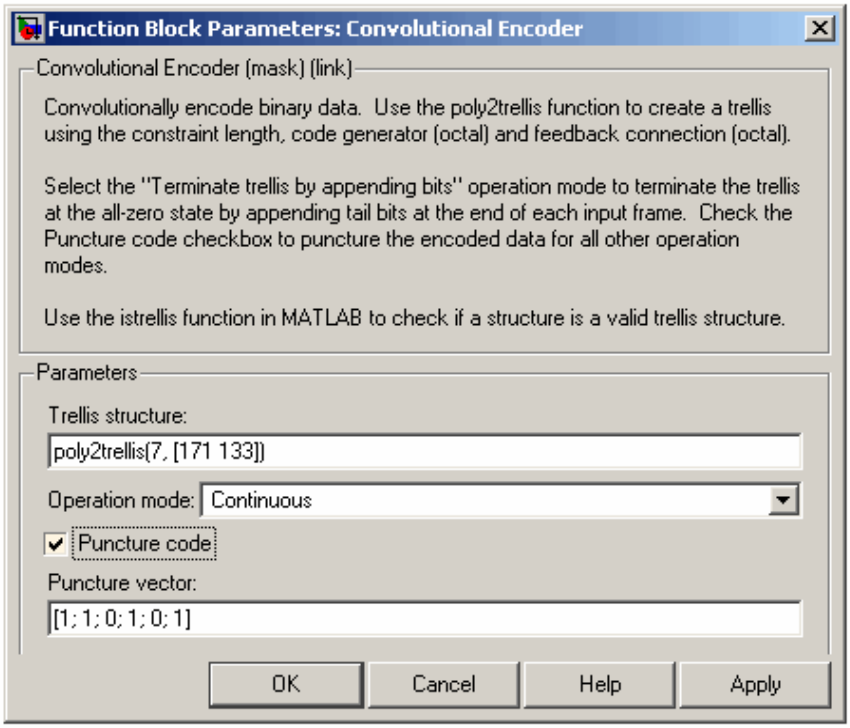
Especificando el codificador

Para definir el codificador convolucional, se utiliza el parámetro estructura de Trellis. Puede usarse este campo en dos formas:

Con una variable en el espacio de trabajo de MATLAB que contiene la estructura Trellis. Esta manera causa que Simulink emplee menos tiempo actualizando el diagrama al principio de cada simulación, comparado con el uso descripto a continuación.

O se puede especificar el codificador usando su longitud de restricción, polinomios generadores y polinomios de conexión de retroalimentación posibles, usando el comando poly2trellis dentro del campo estructura Trellis.

Caja de Diálogo:



Trellis structure (Estructura Trellis)
Estructura MATLAB que contiene la descripción trellis del codificador convolucional.

Operation mode (Modo de operación)
En modo continuo, el bloque guarda su métrica de estados internos al final de cada trama, para usarlo con la próxima trama. Cada camino rastreado hacia atrás es tratado en forma independiente.

En modo truncado (resetea cada trama), el bloque trata cada trama en forma independiente. El camino rastreado hacia atrás comienza en el estado con la mejor métrica y siempre termina en el estado todo ceros.

En el modo terminar trellis agregando bits, el trellis es terminado en el estado todo ceros, agregando bits de cola al final de cada trama de entrada.

En el modo resetear entrada no nula a través de puerto, el bloque tiene un puerto adicional de entrada etiquetado Rst. Cuando la entrada Rst es no nula, el codificador resetea al estado todo ceros.

Puncture code (Código de Pinchaje)
Elegir esta opción abre el campo vector de pinchado.

Puncture vector (Vector de pinchado)
Vector usado para pinchar los datos codificados. El vector de pinchaje es un patrón de 1s y 0s donde los 0s indican los bits pinchados.

F.2.1 Función Poly2Trellis

Trellis = poly2trellis (ConstrainLength, CodeGenerator, FeerbackConnection) devuelve una estructura de trellis donde:

Constraint Lengths
Indica el número de bits almacenados en cada registro, incluyendo la entrada actual.
CodeGenerator
Es una matriz de dimensión k*n, donde k es la cantidad de entradas al diagrama (una en este caso) y n son las salidads, (dos). El elemento en la fila i y columna j indica cómo la entrada i contribuye a la salida j.

Para los bits sistemáticos este parámetro es el mismo que el que se deduce para el vector de FeedbackConnection que se explica más adelante.

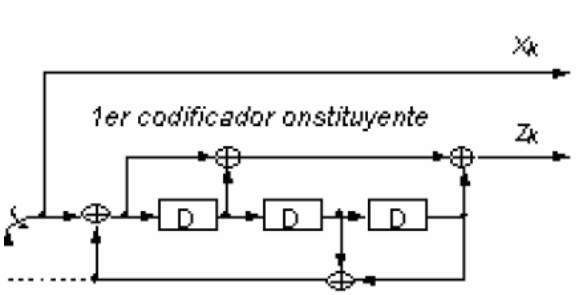
Para las demás salidas se determinan los elementos de la matriz de la siguiente manera:
- Se construye un número binario colocando un 1 o un 0 en cada conexión desde un registro al sumador anterior a la salida, según si existe la conexión o no (1 si hay línea 0 si no). En el número binario, el primero desde la izquierda corresponde a la entrada, y el más a la derecha representa la entrada mas vieja que todavía permanece en el registro.
- Se convierte esta representación binaria a una representación octal, considerando tripletas de bits empezando de la derecha. Por ejemplo: 1010 se interpreta como 001 010 y se convierte a 1 2

FeerbackConnection
La longitud de este vector es el número de entradas al codificador. Los elementos de este vector indican la conexión de realimentación para cada entrada. También se especifica en notación octal.

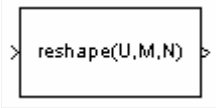
Si el codificador tiene realimentación y además es sistemático, entonces los parámetros CodeGenerator y FeedbackConnection correspondientes a los bits sistemáticos son los mismos.
La figura x.x presenta el modelo del codificador convolucional utilizado en el codificador turbo. De la misma se pueden desprender los valores para los parámetros de la función polly2Trellis. Tiene tres registros, más la entrada actual definen el parámetro Constraint Lengths = 4. La salida Zk puede representarse con el polinomio 1101, según las conexiones que contribuyen al sumador. Y la salida Xk, sistemática se representa al igual que el parámetro *FeerbackConnection*, como 1011.

Para pasarlo a la notación octal: 001 011 = 13
001 101 = 15

Lo que define los parámetros CodeGenerator y FeedbackConnection.
De esta mander la función que define correctamente este codifcador para el caso que se quiere implementar es: polly2trellis(4,[13 15],13).

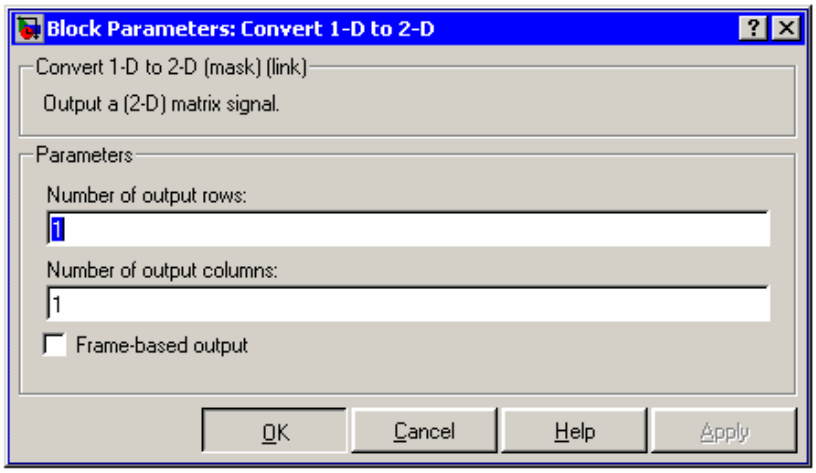


F.3 Convert 1-D to 2-D



El bloque Convertir 1-D a 2-D transforma un vector de largo Mi de una dimensión o una matriz Mi-por-Ni a una matriz Mo por No, donde Mo es especificado por el parámetro cantidad de filas de salida y No es especificado por el parámetro cantidad de columnas de salida.

Caja de diálogo:



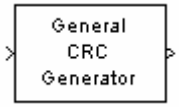
Number of output rows (La cantidad de filas de salida)
La cantidad de filas, Mo, en la matriz de salida.

Number of output columns (La cantidad de columnas de salida)
La cantidad de columnas, No, en la matriz de salida.

Frame-based output (Salida basada en tramas)
Crea una salida basada en tramas cuando es seleccionada.

F.4 CRC

Ubicación: Communications Ubicación blockset → Error Detection and Correction → CRC



Este bloque genera códigos de redundancia cíclica (CRC) para cada trama de datos de entrada y los agrega a la trama. Se especifica el polinomio generador a utilizar por el algoritmo de CRC usando el parámetro incluido en la máscara. Este bloque es general en el sentido que el grado del polinomio no necesita ser potencia de dos. Se representa en polinomio en una de estas dos formas:

- Como un vector binario en fila conteniendo los coeficientes de las potencias en orden descendente. Por ejemplo, [1 1 0 1] representa al polinomio $x^3 + x^2 + 1$.

- Como un vector entero en fila conteniendo las potencies de los términos que no son cero, en orden descendiente. Por ejemplo, [3 2 0] representa al polinomio $x^3 + x^2 + 1$.

Se especifica el estado inicial de los shift registers internos con el parámetro initial state. Este parámetro es vector de escalares o binarios de longitud igual al grado del polinomio generador. Un valor escalar es expandido a un vector de longitud igual al grado del polinomio generador. Por ejemplo, el valor inicial por defecto de [0] se expande a un vector donde todos los elementos son cero.

Se especifica la cantidad de checksums que el bloque calcula para cada trama de entrada en el parámetro Checksums per frame. Este valor debe dividir igualmente el tamaño de la trama de entrada. Si el valor de este parámetro es k, el bloque hace lo siguiente:

Divide cada trama de entrada en k subtramas de igual tamaño.

Prefija el vector de estados iniciales para cada una de las k subtramas.

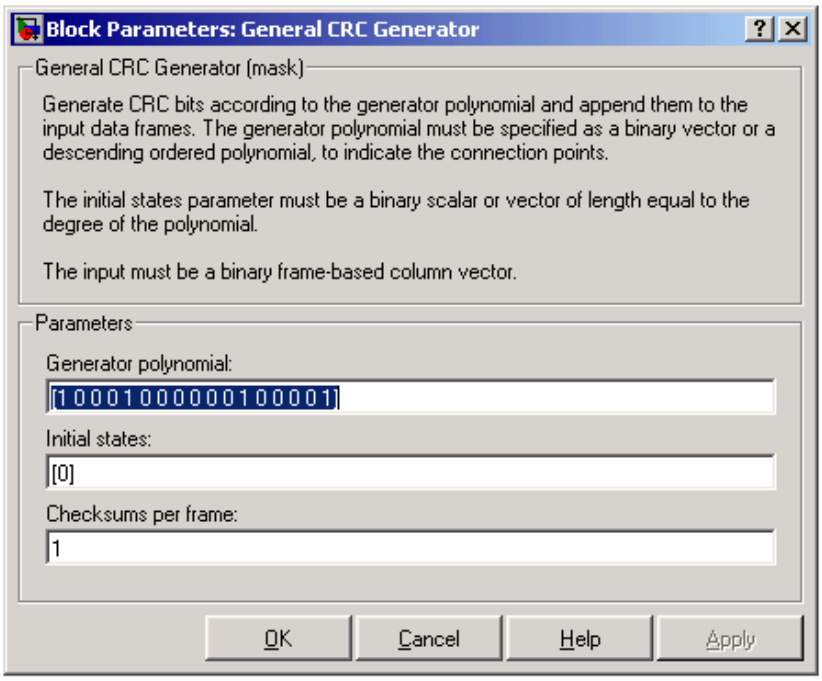
Aplica el algoritmo CRC a cada subtrama aumentada.

Agrega los checksums resultantes al final de cada subtrama.

Saca las subtramas concatenadas.

Si el tamaño de la trama de entrada es m y el grado del polinomio generador es r, la trama de salida tiene tamaño m + k * r.

Caja de diálogo:

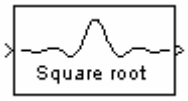


Generator polynomial (polinomio generador)
Un vector fila binario o entero que especifica el polinomio generador, en orden descendente de las potencias.

Initial states (estados iniciales)
Vector fila escalar o binario de largo igual al grado del polinomio generador, especificando el estado inicial de los shift registers internos.

Checksums per frame (Checksums por trama)
Entero positivo que especifica la cantidad de checksums que el bloque calcula para cada trama de entrada.

F.5 Filtro Receptor



Características del filtro

Las características del filtro coseno elevado son las mismas que las del bloque filtro transmisor coseno elevado, excepto que la longitud de la respuesta de entrada del filtro tiene una expresión ligeramente distinta: 2 * N * retardo de grupo + 1, donde N es el valor del parámetro muestras de entrada por símbolo (no el factor sobremuestreo, como en el caso del bloque transmisor coseno elevado).

Si el parámetro ganancia del filtro es seleccionado para ser especificado por el usuario, entonces la ganancia pasabanda del filtro es:

- $20\log_{10}[(Factor\ de\ sobremuestreo\ (N)]x\ ganancia\ lineal\ del\ filtro)$ para un filtro nomal.
- $20\log_{10}[(\sqrt{Factor\ de\ sobremuestreo\ (N)}]\ x\ ganancia\ lineal\ del\ filtro)$ para un filtro raíz cuadrada.

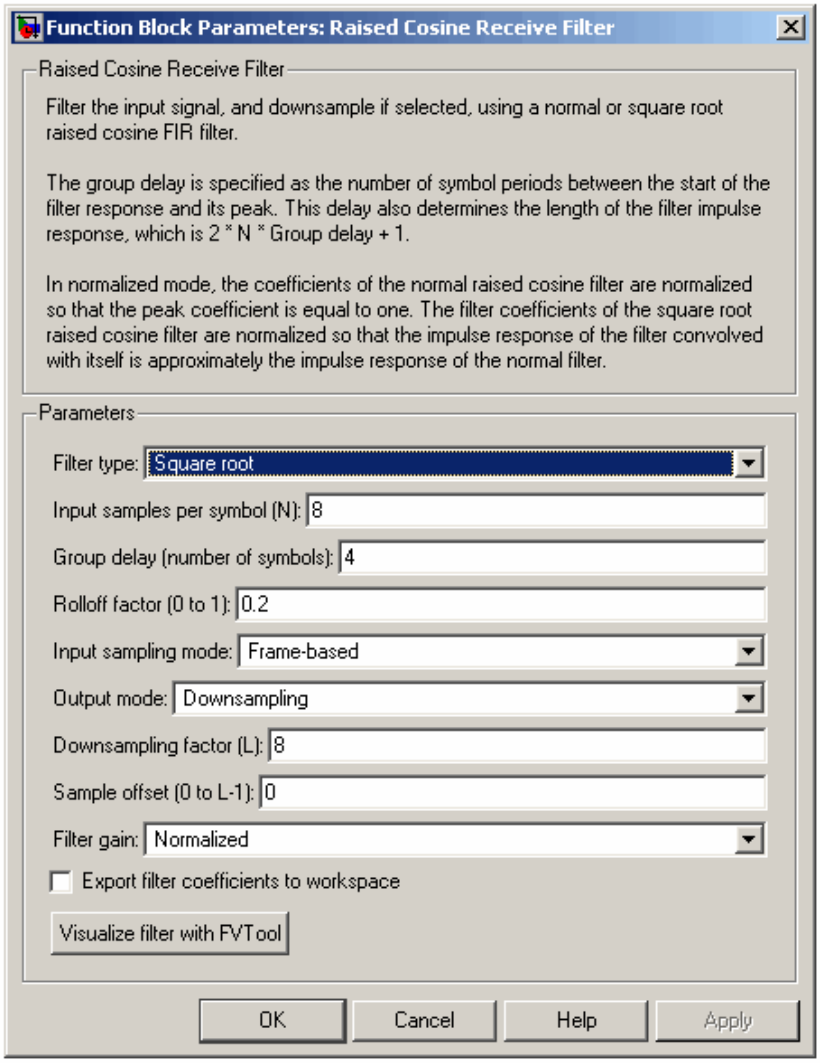
Submuestreando la señal filtrada

Para que el bloque submuestree la señal filtrada, se fija el parámetro modo de salida a submuestreo. Si L es el valor del parámetro factor de submuestreo, entonces el bloque retiene 1/L de las muestras, eligiéndolas como sigue:

- Si el parámetro offset de muestra es cero, entonces el bloque elige las muestras de la señal filtrada elegida indexadas por 1, L+1, 2*L+1, 3*L+1, etc.
- Si el parámetro offset de muestra es un entero positivo menor que L, entonces el bloque descarta inicialmente la cantidad de muestras de la señal filtrada y submuestrea los datos restantes como en el caso anterior.

Para preservar la señal filtrada entera y evitar el submuestreo, se fija el modo de salida en ninguno. Esto es apropiado, por ejemplo, cuando la salida del bloque del filtro entra a otro bloque que dentro de sus funciones realiza submuestreo.

Caja de diálogo



Filter type (Tipo de filtro)
El tipo de filtro coseno elevado: raíz cuadrada o normal.

Input samples per symbol (Muestras de entrada por símbolo)
Un entero mayor que 1 representando la cantidad de muestras por símbolo en la señal de entrada.

Group delay (Retardo de grupo)
Un entero positivo que representa la cantidad de períodos de símbolo entre el comienzo de la respuesta del filtro y su pico.

Rolloff factor (Factor de rolloff)
El factor de rolloff del filtro, un número real entre 0 y 1.

Input sampling mode (Modo de muestra de entrada)
El tipo de la señal de entrada: basado en tramas o en muestras.

Output mode (Modo de salida)
Determina si el bloque submuestrea o no la señal luego de filtrarla. Las opciones son Submuestreo o ninguno.

Downsampling factor (Factor de submuestreo)
El factor por el cual el bloque submuestrea la señal luego de filtrarla. Este campo aparece solo si el modo de salida está fijado en submuestreo.

Sample offset (Offset de muestra)
La cantidad de muestras filtradas que el bloque descarta antes de submuestrear. Este campo aparece sólo si el modo de salida es submuestreo.

Filter gain (Gancia del filtro)
Determina como el bloque escala los coeficientes del filtro. Las opciones son normalizado o especificado por el usuario.

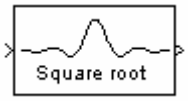
Linear amplitude filter gain (Ganancia lineal del filtro)
Escalar positivo usado para escalar los coeficientes del filtro. Este campo aparece sólo si se fijó ganancia del filtro como especificado por el usuario.

Export filter coefficients to workspace (Exportar coeficientes del filtro al workspace)
Si se marca esta casilla, entonces el bloque crea una variable en el workspace de MATLAB que contiene los coeficientes del filtro.

Coefficient variable name (Nombre de la variable de coeficientes)
El nombre de la variable a crear en el workspace de MATLAB. Este campo aparece sólo si se seleccionó exportar coeficientes del filtro al workspace.

Visualize filter with FVTool (Visualizar filtro con FVTool)
Si se cliquea este botón, entonces MATLAB lanza la herramienta Visualización de Filtro, fvtool, para analizar el filtro coseno elevado cuando se apliquen cambios a los parámetros del filtro.

F.6 Filtro transmisor:



Características del filtro

El parámetro tipo de filtro determina que tipo de filtro usa el bloque; las opciones son normal o raíz cuadrada.

La respuesta al impulso de un filtro coseno elevado normal con factor de rolloff R y período de símbolo T es

$$h(t) = \frac{\sin\left(\frac{\pi t}{T}\right)}{\left(\frac{\pi t}{T}\right)} \cdot \frac{\cos\left(\frac{\pi R t}{T}\right)}{\left(1 - \frac{4R^2t^2}{T^2}\right)}$$

La respuesta al impulso de un filtro raíz cuadrada de coseno elevado con factor de rolloff es

$$h(t) = 4R.\frac{\cos\left(\frac{(1+R)\pi t}{T}\right) + \frac{\sin\left(\frac{(1-R)\pi t}{T}\right)}{\frac{4Rt}{T}}}{\pi\sqrt{T\left(1-\left(\frac{4Rt}{T}\right)^2\right)}}$$

El parámetro retardo de grupo es la cantidad de períodos de símbolo entre el comienzo de la respuesta del filtro y el pico de la respuesta del filtro. El retardo de grupo y el factor de sobremuestreo, N, determinan el largo de la respuesta al impulso del filtro, que es 2 * N * retardo de grupo + 1.

El parámetro factor de rolloff es el factor rolloff del filtro. Debe ser un número real entre 0 y 1. El factor de rolloff determina el exceso de ancho de banda del filtro. Por ejemplo, un factor de rolloff 0,5 significa que el ancho de banda del filtro es 1,5 veces la frecuencia de muestreo de entrada.

El parámetro ganancia del filtro indica como el bloque normaliza los coeficientes del filtro. Si se elige normalizado, entonces el bloque usa un escalado automático:

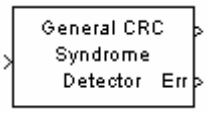
- Si el tipo de filtro es normal, entonces el bloque normaliza los coeficientes del filtro tal que los coeficientes pico sean iguales a 1.
- Si el tipo de filtro es raíz cuadrada, entonces el bloque normaliza los coeficientes del filtro tal que la convolución del filtro consigo mismo produzca un filtro coseno elevado normal cuyos coeficientes pico sean iguales a 1.

Si el parámetro ganancia del filtro es seleccionado para ser especificado por el usuario, entonces la ganancia pasabanda del filtro es:

- $20\log_{10}[(Factor\ de\ sobremuestreo\ (N)]x\ ganancia\ lineal\ del\ filtro)$ para un filtro nomal.
- $20\log_{10}[\left(\sqrt{Factor\ de\ sobremuestreo\ (N)}\right]\ x\ ganancia\ lineal\ del\ filtro)$ para un filtro raíz cuadrada.

El cuadro de diálogo contiene contiene los mismos parámetros que para el caso receptor explicado en el punto anterior.

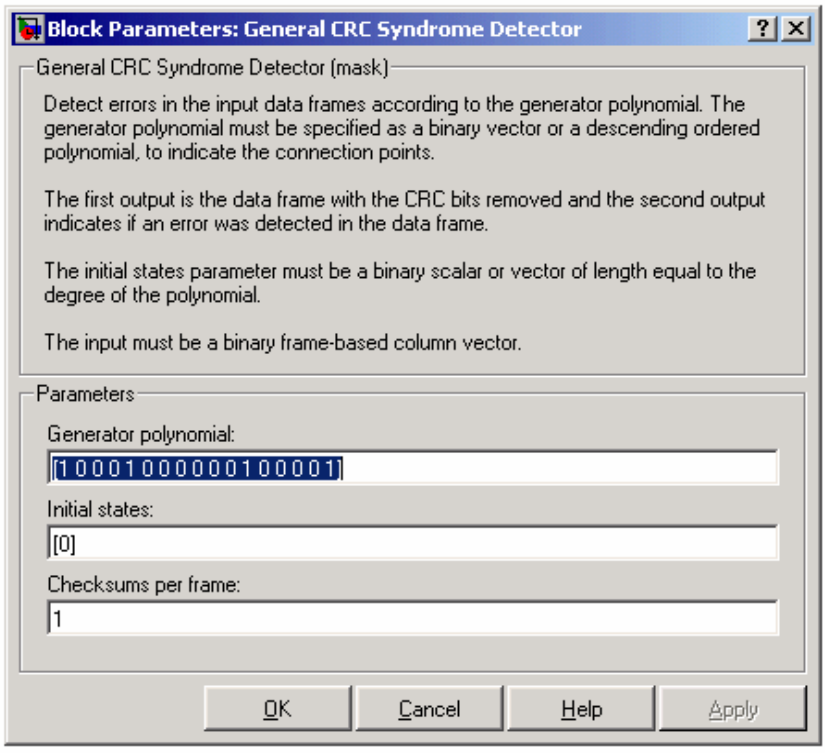
F.7 CRC Detector



Este bloque computa checksums para la trama entera de entrada. La segunda salida del bloque es un vector cuyo tamaño es la cantidad de checksums y cuyas entradas son 0 si la compitación de checksum da un valor cero y 1 en el caso contrario. La primera salida es el conjunto de mensajes con los checksums retirados.

Se especifica la cantidad de checksums que el bloque calcula para cada trama con el parámetro Checksums per frame. Si el valor de este parámetro es k, el tamaño de la trama de entrada es n y el grado del polinomio generador es r, entonces k debe dividir n - k*r, que es el tamaño del mensaje.

Caja de diálogo:



Generator polynomial (Polinomio generador)
Un vector fila entero o binario que especifica el polinomio generador, en orden descendente de potencies.

Initial states (Estados iniciales)
Un vector fila escalar o binario de largo igual al grado del polinomio generador, especificando el estado inicial de los shift registers internos.

Checksums per frame (Checksums por trama)
Un entero positivo que especifica la cantidad de checksums que el bloque calcula por cada trama de entrada.

F.8 Multipath Rayleigh Fading Channel:

El bloque de canal multicamino Rayleigh implementa una simulación bandabase de un canal de propagación multicamino Rayleigh. Este canal es útil para modelado de sistemas de comunicación inalámbrica.

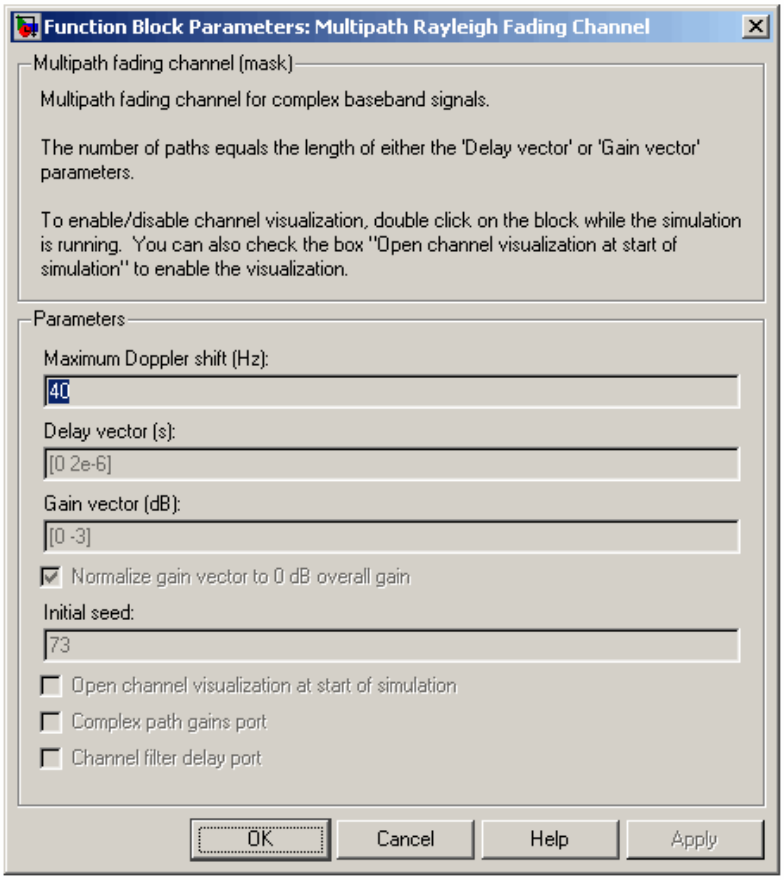
Movimiento relativo entre el transmisor y el receptor causa corrimientos Doppler en la frecuencia de la señal. La densidad espectral de potencia Jakes determina el espectro del proceso Rayleigh.

En los parámetros del bloque, el vector de retraso especifica el retraso temporal para cada camino. Si la casilla de Normalizar vector de ganancia a ganancia general 0 dB está sin marcar, entonces el vector de ganancia especifica la ganancia para cada camino. Si la casilla está marcada, entonces el bloque usa un vector de ganancia múltiple en lugar del vector de ganancia mismo, eligiendo el factor de escalado tal que la ganancia efectiva del canal considerando todos los caminos sea 0 dB.

La cantidad de caminos es el largo del vector de retraso o vector de ganancia, el que sea más largo.

El bloque multiplica la señal de entrada por muestras de un proceso aleatorio complejo de distribución Rayleigh. El parámetro escalar semilla inicial da la semilla para el generador de números aleatorios.

Caja de diálogo



Maximum Doppler shift (Hz) (Máximo desplazamiento Doppler)
Un escalar positivo que indica el máximo desplazamiento Doppler.

Delay vector (s) (Vector de retardo)
Un vector que especifica el retardo de propagación para cada camino.

Gain vector (dB) (Vector de Ganancia)
Un vector que especifica la ganancia para cada camino.

Normalize gain vector to 0 dB overall gain (Normalizar vector de ganancia a ganancia general 0 dB)
Chequear esta casilla causa que el bloque escale el parámetro vector de ganancia tal que la ganancia efectiva del canal (considerando todos los caminos) es de 0 decibeles.

Initial seed (Semilla inicial)
La semilla escalar para el generador de ruido gaussiano.

Algoritmo

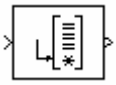
Esta implementación está basada en un simulador de forma directo descripto en Jeruchim, Michel C., Balaban, Philip, and Shanmugan, K. Sam, Simulation of Communication Systems, Second edition, New York, Kluwer Academic/Plenum, 2000.

Algunas aplicaciones inalámbricas, tales como los sistemas estandar GSM (Sistema Global para comunicaciones Móviles), prefieren especificar desplazamientos Doppler en términos de la velocidad del móvil. Si el móvil se mueve a una velocidad v haciendo un ángulo θ con la dirección de la onda de movimiento, entonces el desplazamiento Doppler es

$$fd = (v*f/c)* \cos \theta$$

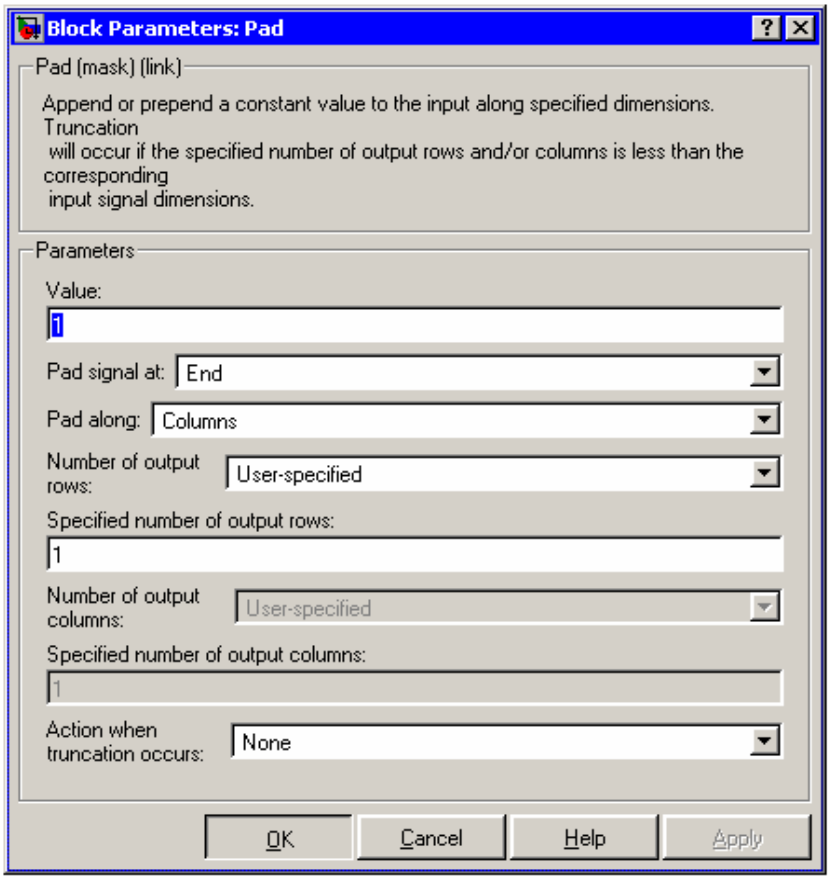
donde f es la frecuencia de la portadora de transmisión y c es la velocidad de la luz. La frecuencia Doppler es el desplazamiento máximo Doppler surgiendo del movimiento del móvil.

F.9 Pad



Este bloque cambia las dimensiones de la matriz de entrada de Mi-por-Ni a Mo-por-No rellenando o truncando las columnas, las filas o columnas y filas.

Caja de diálogo:



Value (Valor)

El valor escalar con el cual rellenar la matriz de entrada.

Pad signal at (Rellenar señal en)
La matriz de entrada puede ser rellenada al principio de las filas y/o columnas o al final de las filas y/o columnas.

Pad along (Rellenar a lo largo)
La dirección a lo largo de lo cual rellenar o truncar. Columnas especifica que la dimensión de filas debería ser cambiada a Mo. Filas especifica que la dimensión de columnas debería ser cambiada a No. Columnas y filas especifica que ambas dimensiones de filas y columnas debería ser cambiada.

Number of output rows (Cantidad de filas de entrada)
La cantidad total de filas de salida. Cuando se elige especificado por el usuario, se ingresa un valor escalar en el parámetro cantidad especificada de filas de salida. Cuando se elige próxima potencia de dos, el bloque rellena las columnas de la matriz de entrada hasta que la cantidad de filas es igual a una potencia de dos. Cuando la cantidad de filas ya es una potencia de dos, el bloque no rellena la matriz de entrada.

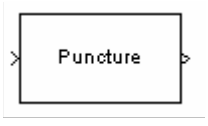
Specified number of output rows (Cantidad de filas de salida)
La cantidad deseada de filas en la salida, Mo. Este parámetro es habilitado cuando selecciona Columnas o Columnas y filas en el menú de rellenar a lo largo y se elige especificado por el usuario en el parámetro cantidad de filas de salida.

Number of output columns (Cantidad de columnas de salida)
La cantidad total de columnas de salida.

Specified number of output columns (Cantidad especificada de columnas de salida)
La cantidad deseada de columnas en la salida, No. Este parámetro es habilitado cuando selecciona Filas o Columnas y filas en el menú de rellenar a lo largo y se elige especificado por el usuario en el parámetro cantidad de columnas de salida.

Action when truncation occurs (Acción cuando ocurre truncado)
Se puede elegir entre ninguna acción, Advertencia y Error para cuando la matriz de entrada es truncada. Advertencia despliega una advertencia y Error muestra una caja de diálogo de error y termina la simulación.

F.10 Puncture



Este bloque crea un vector de salida removiendo elementos seleccionados del vector de entrada y preservando otros. La entrada puede ser un vector real o

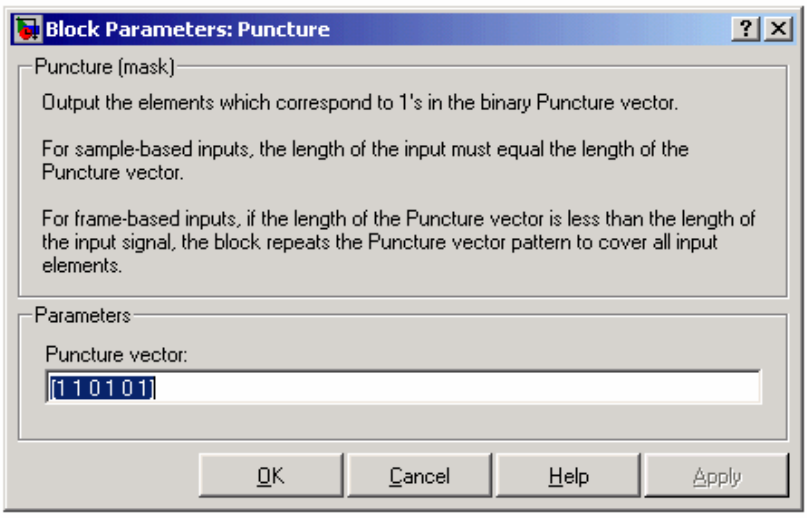
complejo de longitud K. El bloque determina que elementos remover o preservar usando el parámetro binario vector de pinchaje:

- Si Puncture vector(k) = 0, entonces el k-ésimo elemento del vector de entrada no forma parte del vector de salida.

- Si Puncture vector(k) = 1, entonces el k-ésimo elemento del vector de entrada es preservado en el vector de salida.

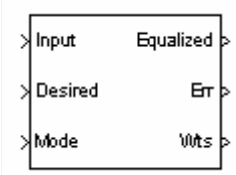
Aquí k varía entre 1 y K. Los elementos preservados aparecen en el vector de salida en el mismo orden en el cual aparecen en el vector de entrada.

Caja de diálogo



Puncture vector (Vector de borrado)
Un vector binario cuyo patrón de 0s (1s) indica que elementos de la entrada el bloque debería remover (preservar).

F.11 RLS linear equalizer:

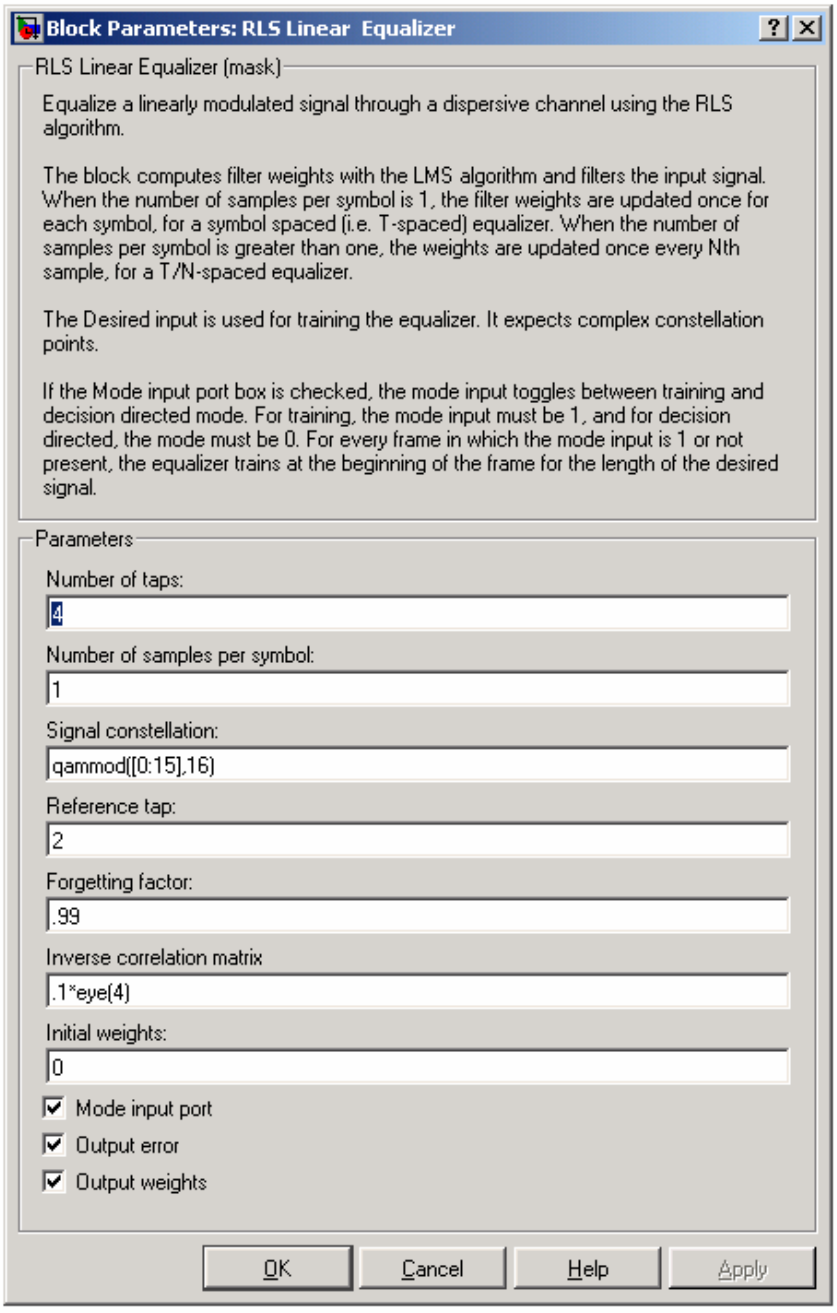


El bloque ecualizador lineal RLS usa un ecualizador lineal y el algoritmo RLS para ecualizar una señal modulada linealmente en banda base a través de un canal dispersivo. Durante la simulación, el bloque usa el algoritmo RLS para actualizar los coeficientes, una vez por símbolo. Si el parámetro cantidad de muestras por símbolo es 1, entonces el bloque implementa un ecualizador símbolo-espaciado; de otro modo, el bloque implementa un ecualizador espaciado fraccionalmente.

Se puede configurar el bloque para que tenga uno o más de estos puertos extra:

- Mode (Modo), es un Puerto de entrada para definir si se usa entrenamiento o si se dirige por decisiones.
- Err (Error) salida de la señal de error, que es la diferencia entre la salida ecualizada y la señal de referencia. La señal de referencia consiste de símbolos de entrenamiento en el modo entrenamiento y símbolos detectados en el otro caso.
- Weights (Pesos), salida que da los pesos de los coeficientes del filtro.

Caja de diálogo



Number of taps (Cantidad de coeficientes)
La cantidad de coeficientes en el filtro del ecualizador lineal.

Number of samples per symbol (Cantidad de muestras por símbolo)
La cantidad de muestras de entrada para cada símbolo.

Signal constellation (Constelación de la señal)
Un vector de números complejos que especifica la constelación para la modulación.

Reference tap (Coeficiente de referencia)
Un entero positivo menor o igual que la cantidad de coeficientes en el ecualizador.

Forgetting factor (Factor de olvido)
El factor de olvido del algoritmo RLS, un número entre 0 y 1.

Inverse correlation matrix (Matriz inversa de correlación)
El valor inicial para la matriz inversa de correlación. La matriz debe ser N-por-N, donde N es la cantidad de coeficientes.

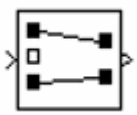
Initial weights (Pesos iniciales)
Un vector que lista los pesos iniciales de los coeficientes.

Mode input port (Puerto de entrada modo)
Si se marca esta casilla, el bloque tiene un puerto de entrada que permite alternar entre los modos entrenamiento y dirigido por decisiones.

Output error (Salida error)
Si se marca esta casilla, el bloque saca la señal error, que es la diferencia entre la señal ecualizada y la señal referencia.

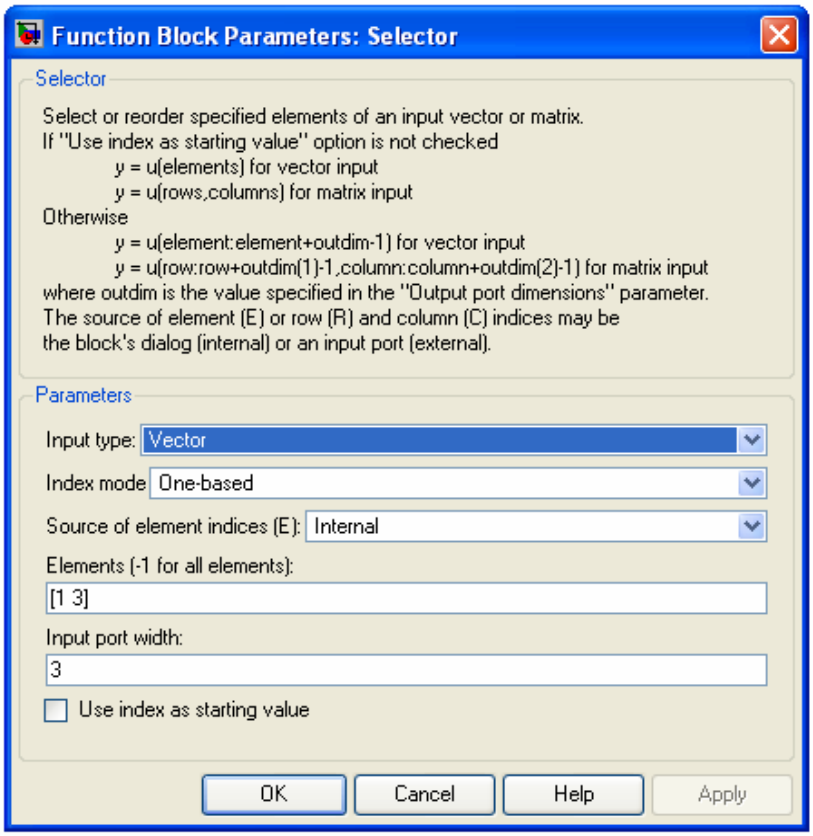
Output weights (Pesos de salida)
Si se marca esta casilla, el bloque saca los pesos actuales.

F.12 Selector:



El bloque selector genera una salida de elementos seleccionados de un vector o una matriz de entrada.

La caja de diálogo aparece como sigue cuando se elige el modo entrada tipo vector.



Input type (Tipo de entrada)
El tipo de la señal de entrada: Vector o Matriz.

Index mode (Modo de índice)
Especifica el modo de indexado: basado en uno o basado en cero.

Source of element indices (Fuente de los índices de elementos)
La fuente de los índices especificando los elementos a seleccionar ya sea interna, o sea, el parámetro Elementos, o externa, o sea una señal de entrada.

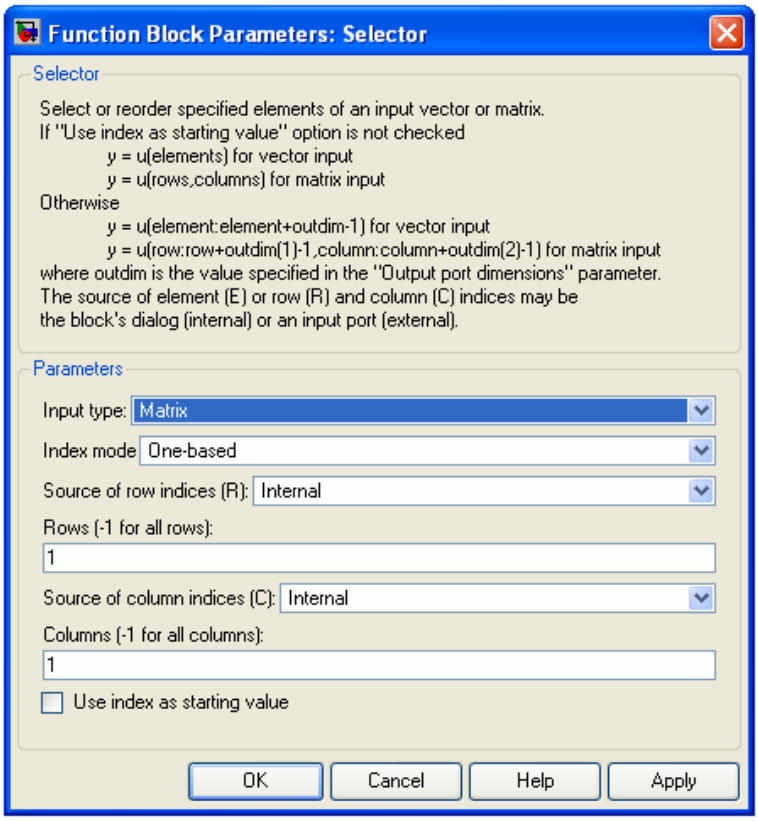
Elements (Elementos)
Los elementos a ser incluidos en el vector de salida.

Input port width (Ancho del Puerto de entrada)
La cantidad de elementos en el vector de entrada.

Use index as starting value (Usar índice como valor inicial)
Especifica que el valor en el campo elementos o el índice de la fuente externa es el índice inicial del rango de elementos cuyo largo es el mismo que el largo especificado en el campo dimensiones del puerto de salida (ver próxima opción).

Output port dimensions (Dimensiones del Puerto de salida)
Este campo aparece solo si se marca Usar índice como valor inicial.
Especifica el ancho de la señal de salida del bloque.

La caja de diálogo aparece como sigue cuando se selecciona el modo entrada tipo matriz.



Input type (Tipo de entrada)
El tipo de la señal de entrada: Vector o Matriz.

Index mode (Modo de índice)
Especifica el modo de indexado: basado en uno o basado en cero.

Source of row indices (Fuente de los índices de fila)
La fuente de los índices que especifican las filas a ser seleccionadas de la matriz de entrada, ya sea interna, o sea el parámetro filas, o externa, o sea, una señal de entrada.

Rows (Filas)
Índices de las filas de las cuales se seleccionan elementos a ser incluidos en la matriz de salida.

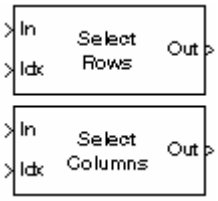
Source of column indices (Fuente de los índices de columna)
La fuente de los índices que especifican las columnas a ser seleccionadas de la matriz de entrada, ya sea interna, o sea el parámetro columnas, o externa, o sea, una señal de entrada.

Columns (Columnas)
Índices de las columnas de las cuales se seleccionan elementos a ser incluidos en la matriz de salida.

Use index as starting value (Usar índice como valor inicial)
Especifica que el valor en los campos filas y columnas o índice de fuente externa especifican los índices de la fila y columna inicial de un rango de elementos cuyo largo es el mismo que las dimensiones especificadas en el campo dimensiones del puerto de salida (ver la próxima opción).

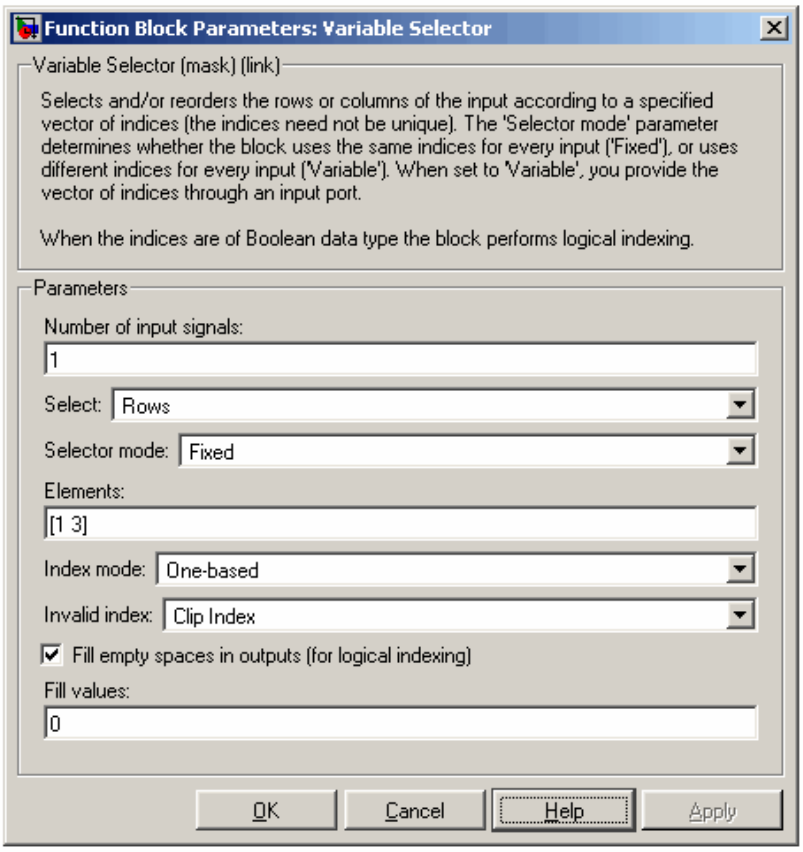
Output port dimensions (Dimensiones del Puerto de salida)
Este campo aparece sólo si se marca la casilla Usar índice como valor inicial. Especifica las dimensiones de la señal de salida del bloque como un vector de dos elementos: [F C] (filas columnas)

F.13 Selector Variable:



Este bloque extrae un subconjunto de filas o columnas de la matriz de entrada u de dimensiones M-por-N en cada puerto de entrada. La cantidad de puertos de entrada y salida se especifica en el parámetro cantidad de señales de entrada.

Caja de diálogo



Number of input signals (Cantidad de señales de entrada)
Especifica la cantidad de señales de entrada. Un puerto de entrada es creado en el bloque para cada señal de entrada.

Select (Seleccionar)
La dimension de la entrada a seleccionar, filas o columnas.

Selector mode (Modo de selector)
El tipo de operación de indexado a realizar, variable o fija. Indexado variable usa la entrada en el puerto Idx para seleccionar filas o columnas de la entrada en el puerto In. Indexado fijo usa el parámetro elementos para seleccionar filas de la entrada en el puerto In y deshabilita el puerto Idx.

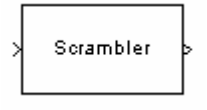
Elements (Elementos)
Un vector conteniendo los índices de las filas o columnas de entrada que aparecerán en la matriz de salida. Este parámetro sólo es visible cuando selecciona Fijo en el parámetro del modo de selector.

Index mode (Modo de índice)
Cuando se fija a basado en unos, un valor de índice 1 se refiere a la primer fila o columna de la entrada. Cuando se fija a basado en ceros, un valor de índice de 0 se refiere a la primer fila o columna de la entrada.

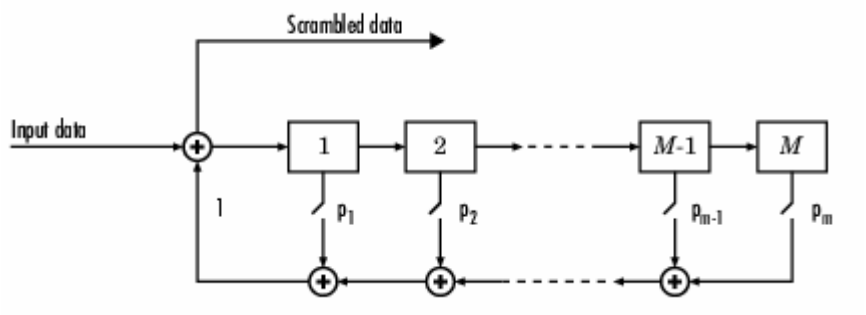
Invalid index (Índice inválido)
Respuesta a un valor de índice inválido. Ajustable.
Llena espacios vacíos en las salidas (para indexado lógico)

Fill values (Valores de llenado)
Especifica los valores de llenado cuando el bloque realiza indexado lógico. Este parámetro es sólo visible cuando se ha seleccionado el parámetro Llenar espacios vacíos en las salidas (para indexado lógico).

F.14 Scrambler:



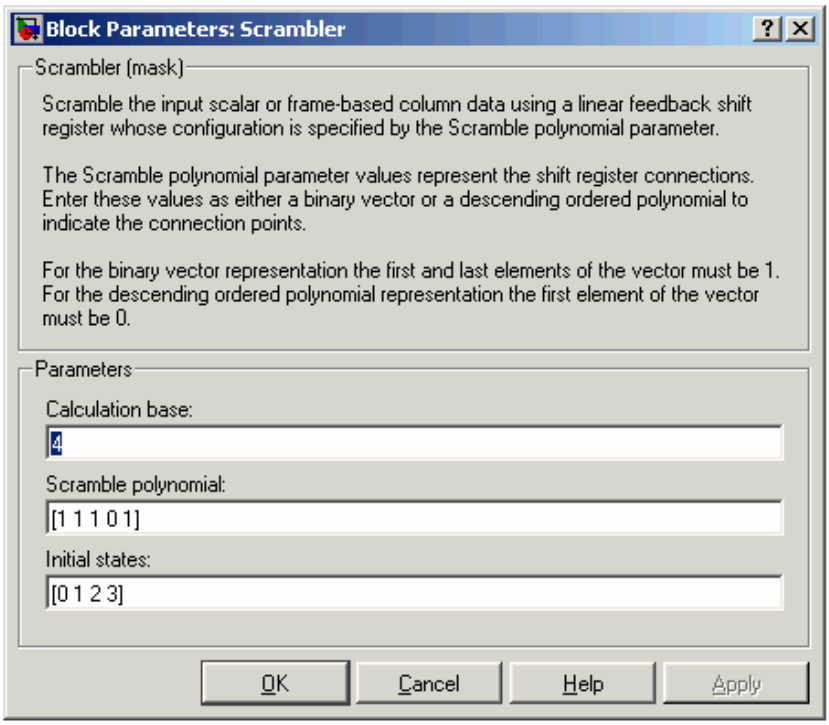
El bloque Scrambler entrevera la señal de entrada, que debe ser un escalar o un vector columna basado en tramas. Si el parámetro base de cálculo es N, entonces los valores de entrada deben ser enteros entre 0 y N-1.



En cada paso temporal, la entrada causa que el contenido de los registros se desplace secuencialmente. Cada interruptor del scambler está encendido o apagado como se define en el parámetro polinomio de scramble. Se puede especificar el polinomio listando sus coeficientes en orden de potencias ascendente de z^{-1}, donde $p(z^{-1}) = 1 + p_1z^{-1} + p_2z^{-2}+...$, o listando las potencias de z que aparecen en el polinomio con un coeficiente de 1. Por ejemplo, p = [1 0 0 0 0 0 1 0 1] y p = [0 -6 -8] representan el polinomio $p(z-1) = 1 + z^{-6} + z^{-8}$.

El parámetro estados iniciales lista los estados de los registros del scrambler cuando la simulación comienza. Los elementos de este vector deben ser enteros entre 0 y N-1. El largo del vector de este parámetro debe ser igual al orden del polinimio de scramble. (Si el parámetro polinomio de scramble es un vector que lista los coeficientes en orden, entonces el orden del polinomio de scramble es uno menos que el largo del vector.)

Caja de Diálogo:



Calculation base (Base de cálculo)
La base de cálculo N. La entrada y la salida de este bloque son enteros en el rango [0, N-1].

Scramble polynomial (Polinomio de scramble)
Un polinomio que define las conexiones en el scrambler.

Initial states (Estados iniciales)
Los estados de los registros del scrambler cuando comienza la simulación.

F.14.1 Verificación de que el bloque scrambler sirve para realizar el bit scrambling:

En la norma vemos que el bit scrambling se describe de la siguiente forma:

$d_{im,k} = (b_{im,k} + y_k) \bmod 2 \qquad k = 1,2,\ldots,B$

e y_k resulta de la siguiente operación:

$y'_\gamma = 0 \qquad -15 < \gamma < 1$

$y'_\gamma = 1 \qquad \gamma = 1$

$y'_\gamma = \left(\sum_{x=1}^{16} g_x \cdot y'_{\gamma - x} \right) \bmod 2 \qquad 1 < \gamma \le B$,

donde $g = \{g_1, g_2, \ldots, g_{16}\} = \{0,0,0,0,0,0,0,0,0,0,1,0,1,1,0,1\}$,

$y_k = y'_k \qquad k = 1,2,\ldots,B.$

Viendo lo anterior y el esquema del scrambler en la figura 4.1 vemos que:

Dado que $b_{im,k}$ es la entrada y $d_{im,k}$ es la salida, si el otro sumando en el sumador cumple con la definición de y_k este bloque es la implementación del bit scrambling.

$y_k=y'_k$ siendo $y'_k = \left(\sum_{x=1}^{16} g_x \cdot y'_{k-x} \right) \bmod 2$

$y'_k=0$ $-15<k<1$, $y'_1=1$

Luego

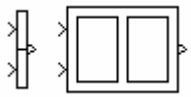
$y'_2=g_1.y'_1+g_2.y'_0+.....+g_{16}.y'_{-14}$

$y'_3= g_1.y'_2+g_2.y'_1+.....+g_{16}.y'_{-13}$

Entonces se ve que esto es equivalente al esquema de la figura 4.1

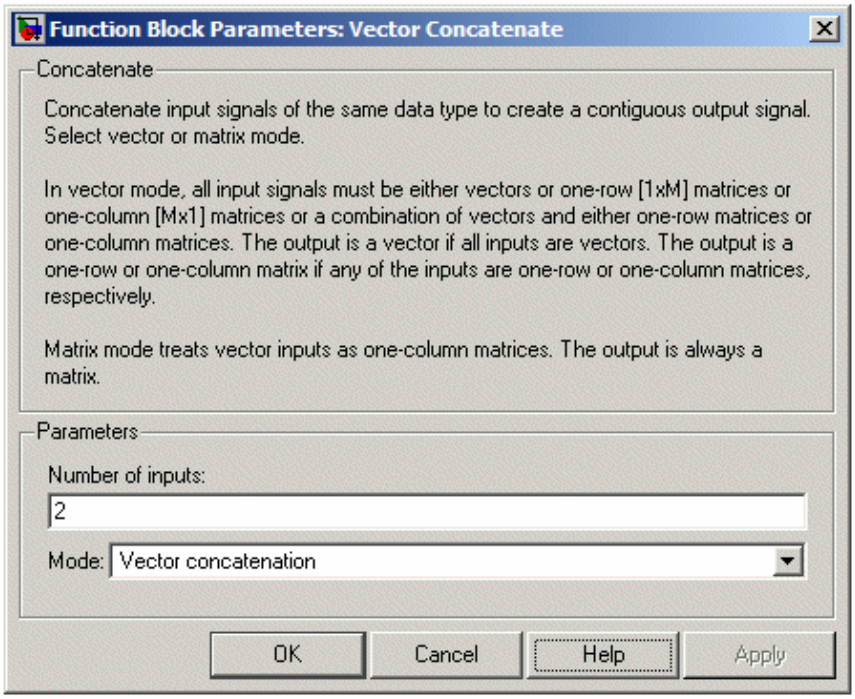
Si M=16, $p_1 \ldots p_{16}$ serían $g_1 \ldots g_{16}$ y lo que entra al primer bloque es para k=2, y'-14, lo que entra al segundo bloque es y'-13 y así sucesivamente.
Por lo tanto, el scrambler sirve para el bit scrambling.

F.15 Vector Concatenate:



Este bloque concatena las señales en sus entradas para crear una señal de salida cuyos elementos residen en lugares contiguos en memoria. Opera en modo vector o matriz, dependiendo de cómo esté fijado el parámetro mode.

Parámetros y caja de diálogo

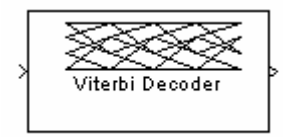


Number of inputs (Cantidad de entradas)
Cantidad de entradas al bloque

Mode (Modo)
Especifica el tipo de concatenación realizado por este bloque. Las opciones son:

Vector concatenation (Concatenación de vectores)
Horizontal matrix concatenation (Concatenación horizontal de matrices
Vertical matrix concatenation (Concatenación vertical de matrices)

F.16 Viterbi Decoder



Tamaños de entrada y salida

Si el código convolucional usa un alfabeto de 2^n símbolos posibles, el largo del vector de entrada al bloque es L*n para algún entero positivo L. Similarmente, si los datos decodificados usan un alfabeto de 2^k posibles símbolos de salida, el largo del vector de salida del bloque es L*k. El entero L es la cantidad de tramas que el bloque procesa en cada paso.

La entrada puede ser un vector basado en muestras con L=1 o un vector columna basado en tramas con algún entero positivo L.

Retardo de decodificación y profundidad de rastreo hacia atrás

El parámetro profundidad de rastreo hacia atrás, D, influye en el retardo de decodificación. El retardo de decodificación es la cantidad de símbolos cero que precede el primer símbolo decodificado en la salida.

Si la señal de entrada es basada en muestras, el retardo de decodificación consiste de D símbolos cero.

Si la señal de entrada es basada en tramas y el parámetro modo de operación está fijado en continuo, el retardo de decodificación consiste de D símbolos cero.

Si el parámetro modo de operación está fijado en truncado o terminado, no hay retardo de salida y el parámetro profundidad de rastreo debe ser menor o igual que la cantidad de símbolos en cada trama.

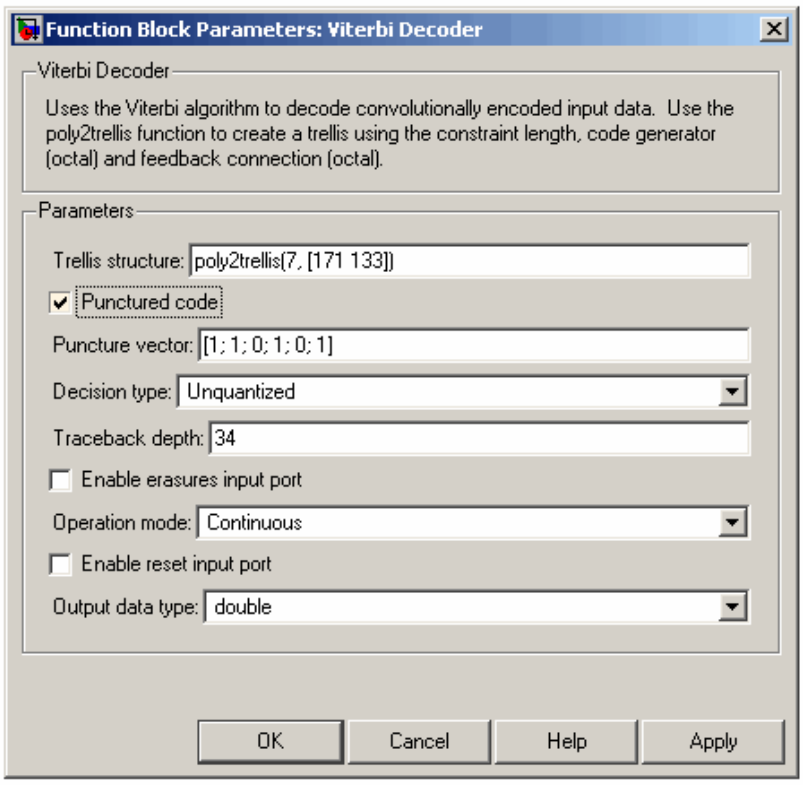
Si la tasa de codificación es 1/2, una profundidad de rastreo es cerca de cinco veces la longitud de restricción del código.

Puerto de reset

El puerto de reset es utilizable sólo cuando el parámetro modo de operación está fijado en continuo. Chequeando la casilla Habilitar puerto de entrada de reset causa que el bloque tenga una entrada adicional, marcada Rst. Cuando la entrada Rst es distinta de cero, el decodificador regresa a su estado inicial al configurar su memoria interna como sigue:

- Fija la métrica de estado todos cero a cero.
- Fija las otras métricas de estado al valor máximo.
- Fija la memoria de rastreo a cero.

Caja de diálogo:



Trellis structure (Estructura Trellis)
Estructura MATLAB que contiene la descripción trellis del codificador convolucional. (detallada en D.2.1).

Punctured code (Código de pinchaje)
Seleccione esta casilla para pinchar el código de entrada. El campo, vector de pinchaje, aparece.

Puncture vector (Vector de pinchaje)
Vector con el patrón constante de borrado usado en el transmisor. El vector de borrado es un patrón de 1s y 0s, donde los 0s indican los bits pinchados. Este campo aparece cuando la casilla Código de pinchaje está seleccionada.

Decision type (Tipo de decision)

Unquantized (Sin cuantizar), Hard Decision o Soft Decision.

Number of soft decision bits (Cantidad de bits de decisión blanda)
La cantidad de bits de decisión blanda usados para representar cada entrada. Este campo está activo sólo cuando el tipo de decisión está fijado en decisión blanda.

Traceback depth (Profundidad de rastreo)
La cantidad de ramas de trellis usada para construir cada camino de rastreo.

Enable erasures input port (Puerto de entrada habilitar borrados)
Cuando se marca esa casilla, el decodificador abre un puerto de entrada llamado Era. A través de este puerto, se puede especificar un vector patrón de borrado de 1s y 0s, donde los 1s indican los bits borrados.

Para estos borrados en el flujo de datos entrante, el decodificador no actualiza la métrica de la rama. Los anchos y tiempos de muestra de los puertos de entrada de datos y de borrado deben ser iguales.

Operation mode (Modo de operación)
Método para transicionar entre sucesivas tramas de entrada. Para entradas basadas en trama, las opciones son continuo, terminado y truncado. Entradas basadas en muestra deben usar el modo continuo.

Enable reset input port (Permitir puerto de reset)
Cuando se chequea esta casilla, el decodificador abre un puerto de entrada llamado Rst. Si se le da un valor de entrada distinto de cero a este puerto causa que el bloque fije su memoria interna al estado inicial antes de procesar los datos de entrada.

G. Códigos implementados:

G.1 inter1.m – Código para implementador el intercalador de la codificación Turbo

A.
```
function indx = inter1(x)
%Implementa el Intercalador del codificador Turbo según norma 25.212
% Inicializaciones
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
k = 0;
A = 0;
s(1) = 1;
q(1) = 1;
Q = 7;
PV = [29 31 37 41  43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137
139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251
257;2 3 2 6 3 5 2 2 2 2 7 5 3 2 3 5 2 5 2 6 3 3 2 3 2 2 6 5 2 5 2 2 2 19 5 2 3 2 3 2 6 3 7 7 6 3];
if (2281 <= x <= 2480) | (3161 <= x <= 3210)
   T = [19 9 14 4 0 2 5 7 12 18 16 13 17 15 3 1 6 11 8 10];
else T = [19 9 14 4 0 2 5 7 12 18 10 8 13 17 3 1 16 6 15 11];
end
T = T + 1;
```

```
i = 0;
while i <= length(PV)
    i = i + 1;
    if x <= 20*(PV(1,i)+1)
        p = PV(1,i);
        v = PV(2,i);
        break
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determinar cantidad de columnas de la matriz
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if x <= 20*(p-1)
    c = p - 1;
end
if (20*(p-1) < x) & (x <= 20*p)
    c = p;
end
if x > 20*p
    c = p + 1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Otos vectores auxiliares
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j = 2 : (p-1)
   s(j) = mod(v*s(j-1),p);
end
i = 2;
while i <= 20
    if gcd((p-1),Q) == 1
        q(i) = Q;
        i = i + 1;
        Q = Q + 1;
    else
        Q = Q + 1;
    end
end
for i = 1 : 20
    r(T(i)) = q(i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Armado de la matriz segun la relación entre 'c' y 'p'
for i = 1 : 20
    for j = 1 : p-1
        h = mod(j*r(i),p-1) + 1;
        U(i,j)= s(h)+(i-1)*c;
    end
end
if c == p
    U = U + 1;
    for i = 1 : 20
        U(i,p)= 1 + (i-1)*c;
    end
```

```
end
if c == (p + 1)
   U = U + 1;
   for i = 1 : 20
      U(i,p)= 1 + (i-1)*c;
      U(i,p+1)= p + 1 + (i-1)*c;
   end
   if x == 20*c
      A = U(20,p+1);
      U(20,p+1) = U(20,1);
      U(20,1) = A;
   end
end
for i = 1 : 20
   for j =1 : c
      UT(T(i),j) = U(i,j);
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Esto es solo descomponer la matriz en un vector.
for j = 1 : c
   for i =1 : 20
      if UT(i,j) > x
         k = k + 1;
      else Z(20*(j-1)+i-k) = UT(i,j);
      end
   end
end
indx = Z;
```

G.2 salturbo.m – Implementa la salida del codificador turbo

```
%Define la salida del Codificador Turbo
function s = salturbo(x)
k = 1;
for i = 1 :3: 3*x-2
    h(i) = k;
    h(i+1) = k + 1;
    h(i+2) = 2*x + 7 + k;
    k = k + 2;
    %Crea vector con un bit sistemático, uno de paridad 1 y uno de
paridad
    %2 sucesivamente
end

k = 2*x + 1;
%Agrega bits de Trellis:
for i = 3*x + 1 : 3*x + 6
    h(i) = k;
    k = k + 1;
end
k = 4*x + 6;
for i = 3*x + 7 : 3*x + 12
    k = k + 1;
    h(i) = k;
end
s = h;
```

G.3 Código para definer el Multiport Selector 1 en Decodificador

Demux1

```
%Define la primer salida del Multiport Selector1 en el Decodificador
function r = demux1(x)
sal=ones(1,x-2);
k = 0;
for i = 1 : 3 : 3*x-2
    k = k + 1;
    sal(k) = i;
    %Es un vector que toma uno cada tres bits, empezando por el
primero,
    %formando asi el flujo de bits sistemáticos.
end
r = sal;
```

Demux2

```
%Define la segunda salida del Multiport Selector en el Decodificador
function r = demux2(x)
k = 0;
for i = 2 : 3 : 3*x-1
    k = k + 1;
    l(k) = i;
    %Es un vector que toma uno cada tres bits, empezando por el
segundo,
    %formando asi el flujo de bits de paridad 1.
end
k = x;
for i = 3*x+1 : 3*x+6
    k = k + 1;
    l(k) = i;
    %Agrega los 6 bits de Trellis de la paridad 1
end
r = l;
```

Demux3

```
%Define la tercer salida del Multiport Selector en el Decodificador
function r = demux3(x)
k = 0;
for i = 3 : 3 : 3*x
    k = k + 1;
    l(k) = i;
    %Es un vector que toma uno cada tres bits, empezando por el
tercero,
    %formando asi el flujo de bits de paridad 2.
end
k = x;
for i = 3*x+7 : 3*x+12
    k = k + 1;
    l(k) = i;
    %Agrega los 6 bits de Trellis de la paridad 2
end
r = l;
```

G.4 intermedioDC(x)

```
function s = intermedioDC(x)
%Define la entrada al bloque puncture

k = 1;
for i = 1 :2: 2*x - 1
    h(i) = k;
    h(i+1) = x + k;
    k = k + 1;
    %Crea un vector del tipo [1,x+1,2,x+2...x,2x]
end
for i = 2*x + 1 : 2*x + 6
    h(i) = i;
    %Agrega los últimos 6 bits de la entrada, los bits de Trellis
end
s = h;
```

G.5 Pinchados

```
function indx = pinchados(Px, X)
%Crea el vector patrón de borrado del Rate Matching para cada bloque
de
%código. Px indica si es el primer o segundo bloque de Puncture, X es
el
%bloque de código.

%Inicializaciones para flujo 1
eini=28002;
eminus=(28002-399);
eplus=28002;
Xi=28002;

patsis=ones(28002,1);
pat1=ones(28002,1);
e=eini;
cont1=0;
m=1;
%Armado del vector patrón de borrado para flujo 1
while m <= Xi
    e = e-eminus;
    if e <= 0
        cont1=cont1+1;
        pat1(m)=0;
        e = e + eplus;
    end
    m = m + 1;
end

%Inicializaciones para flujo 2
eini2=28002;
eminus2=2*(28002-399);
eplus2=2*28002;
Xi=28002;
pat2=ones(28002,1);
e2=eini2;
cont2=0;
```

```
m=1;
%Armado del vector patrón de borrado para flujo 2
while m <= Xi
    e2 = e2-eminus2;
    if e2 <= 0
        cont2=cont2+1;
        pat2(m)=0;
        e2 = e2 + eplus2;
    end
    m = m + 1;
end
k = 1 + X*4667;

%Armado del vector de borrado
if Px == 1
%bloque de Punture 1
    for i = 1 :2: 9325
        vit(i) = patsis(k);
        vit(i+1) = pat1(k);
        k = k + 1;
    end
    vit(9327) = patsis(k+1);
    vit(9328) = pat1(k+1);
    vit(9329) = patsis(k+2);
    vit(9330) = pat1(k+2);
    vit(9331) = patsis(k+3);
    vit(9332) = pat1(k+3);
else
%bloque de Punture 2
    for i = 1 :2: 9325
        vit(i) = patsis(k);
        vit(i+1) = pat2(k);
        k = k + 1;
    end
    vit(9327) = patsis(k+1);
    vit(9328) = pat2(k+1);
    vit(9329) = patsis(k+2);
    vit(9330) = pat2(k+2);
    vit(9331) = patsis(k+3);
    vit(9332) = pat2(k+3);
end
indx = vit;
```

G.6 De_inter1

```
function s = de_inter1(x)
%Implementa el inverso del interleaving del codificador Turbo (TS
25.212)
D = inter1(x);
for i = 1 : x
    Z(D(i))= i;
end
s = Z;
```

G.7 Función de inicialización de Rate Matching

No estaba definida la acción a tomar para formato flexible que es nuestro caso, por lo tanto, se agregaron las siguientes líneas, definiendo los parámetros necesarios para correr el algoritmo de rate matching:

```
% Get Ndata1 using slotFormat

Ndata1 = 640;

% P(=numPhCH) is the num of Physical channels at CCtrCh signalled by
higher layers

Ndata = numPhCH*3*(Ndata1);

Ntti=28002;

Nir=172800;

deltaNimax=-55206;


function deltaNimax = wcdma_ratematchinginit(numBits, tti,
RMAttribute, posTrCh, slotFormat, numPhCH)
% WCDMA_RATEMATCHINGINT Computes ANi for the Wcdma Rate Matching block
% included in the Wcdma Application example.
%
% ANi is an intermediate variable that indicates
% whether the Rate Matching block needs to puncture or repeat the
incoming data stream.

% Copyright 1996-2002 The MathWorks, Inc.
% $Revision: 1.2 $  $Date: 2002/04/11 00:48:53 $

numTrCh = length(numBits);

%--- Check for Fixed or Flexible position of TrCh
if posTrCh == 0

    % Max number of bits per TFS(l) (currently l=0) per Transport
Channel
    maxNumBits = max(numBits,1);

    %Compute the number of frames in a TT1 for all the transport
channels
    F = tti./10;

    % Compute Ni
    Ni = maxNumBits./F;

    % Get Ndata1 and Ndata2 using slotFormat
    Ndata1 = slotformattable(slotFormat+1, 4);
    Ndata2 = slotformattable(slotFormat+1, 5);

    % P(=numPhCH) is the num of Physical channels at CCtrCh signalled
by higher layers
    Ndata = numPhCH*15*(Ndata1+Ndata2);
```

```
    % Calculate the Variable Z (defined for all TFC j) (currently j=0)
    den = RMAttribute*Ni'; % Denominator value for Z

    for i=1:numTrCh

        Z(i)= floor((RMAttribute(1:i)*Ni(1:i)')* Ndata/den);
        if (i>1)
            deltaNi(i) = Z(i) - Z(i-1) - Ni(i);
        else
            deltaNi(1) = Z(1) - Ni(1);
        end
    end

    deltaNimax = F.*deltaNi;

    % deltaNimax is the variable generated by the initialization
    % program of the Rate Matching code and used by certain other
    % blocks in the model

    %*****************************************%
    % Code for Compressed Mode
    %*****************************************%

    Npimax = 0;

    deltaNimax = deltaNimax - Npimax;

else % Flexible Position

     % Get Ndata1 using slotFormat
    Ndata1 = 640;

    % P(=numPhCH) is the num of Physical channels at CCtrCh signalled
by higher layers
    Ndata = numPhCH*3*(Ndata1);
    deltaNimax=-27603*2;

end
```

Referencias y Bibliografía

[1] Qualcomm CDMA Technologies White Paper, HSDPA for Improved Downlink Data Transfer Octubre 2004

[2] Tommi Heikkilä, S-72.333 Postgraduate Course in Radio Communications, Autumn 2004 - RAKE Receiver

[3] Maribell Sacanamboy Franco, Diseño e implementación de los Turbo Codificadores definidos en los estándares de telecomunicaciones CDMA2000 (TIA/EIA2002.2D) y WCDMA (3GPP TS 25.212 V7.2.0) usando hardware reconfigurable.

[4] A. Burr, Turbo-codes: the ultimate error control codes?

[5] 3GPP, TS 25.212 V5.10.0, Multiplexing and channel coding (FDD) (Release 5)

[6] 3GPP, TS 25.213 V5.6.0, Spreading and modulation (FDD) (Release 5)

[7] Minjae Park1, Woonsik Lee, Performance Comparison of Chip-level Equalizers in the HSDPA System

[8] Rec. UIT-R M.1225, Pautas de evaluación de lãs tecnologias de transmisión radielétrica para lãs IMT-2000.

[9] V. Erceg, K.V. S. Hari, M.S. Smith, *"Channel Models for Fixed Wireless Applications",* IEEE 802.16 Broadband Wireless Access Working Group, 17 Jul 2001

[10] Harteneck, Boloorian, Georgoulis, Tanner, Practical Aspects of an HSDPA 14 Mbps Terminal

[11] http://www.mathworks.com/support/bugreports/details.html?rp=301866

[12] Practical Aspects of an HSDPA 14 Mbps Terminal – Harteneck, Boloorian, Georgoulis, Tanner

[13] Performance Comparison of Chip-Level Equalizers in the HSDPA System – Park, Lee

[14] An Adaptive RLS MIMO Equalizer Algorithm for HSDPA – Dennis R. Morgan

[15] Evaluation of New NLMS and RLS Chip Equalizers using Tentative Decision Data for HSDPA Systems – Ogawa, Dateki, Furukawa

[16] Practical Aspects of an HSDPA 14 Mbps Terminal – Harteneck, Boloorian, Georgoulis

[17] Performance Comparison of Chip-Level Equalizers in the HSDPA System – Park, Lee

[18] An Adaptive RLS MIMO Equalizer Algorithm for HSDPA – Dennis R. Morgan

[19] http://en.wikipedia.org/wiki/Recursive_least_squares_filter
[20] http://en.wikipedia.org/wiki/Multipath

Bibliografía adicional:

- 3GPP, TS 25.202 V5.3.0, Physical layer – General Description
- 3GPP, TS 25.213 V5.6.0, Spreading and Modulation (Release5)
- 3GPP, TS 25.848 V4.0.0, Physical layer aspects of UTRA HSDPA
- 3GPP, TS 25.858 V5.0.0, Vocabulary
- 3GPP, TS 25.990 V3.0.0, Physical layer aspects (Release5)
- Concepts of High Speed Downlink Packet Access: Bringing Increased Throughput and Efficiency to W-CDMA - Application Note – Agilrent Technologies
- Harri Holma and Antti Toskal - WCDMA FOR UMTS Radio Access for Third Generation Mobile Communications - Third Edition
- John Wiley HSDPA HSUPA for UMTS - Jun.2006
- Adaptive Chip Level Equalization for HSDPA – A. Bastug, S. Sesia, D. Slock
- An Adaptive RLS MIMO Equalizer Algorithm for HSDPA – D. Morgan
- Advanced High Speed Packet Access Receiver for WCDMA Multicode Transmissions with High-Order Modulation – H. Wang, V. Haikola, J. Lilleberg
- Channel Equalization in HSDPA Receivers: Trade-Off between Performance and Complexity with a Variable Oversampling – M. Schumann, M. Bucker, S. Hessell, U. Langmann
- Chip-level Channel Equalization with Rake-like Structures for Multicode Downlink WCDMA Communications – L. Fathi, G. Jourdaint, M. Arndt
- Multicode Interference Canceller with Chip Equalization for HSDPA Systems – Q. Wu, Z. Gao
- Design Considerations for High-Data-Rate UMTS FDD User Equpiment – R. Tanner, C. Luschi, G. Howard
- WCDMA HSDPA Network Performance with Receive Diversity and LMMSE Chip Equalization – T. Nihtila, J. Kurjenniemi, M. Lampinent, T. Ristaniemit
- A WCDMA/HSDPA Baseband Processor – C. Huang, H. Ma
- Design and Real-Time Measurement of HSDPA Equalizers – S. Geirhogel, C. Mehlfihrer, M. Rupp
- Performance Analysis of Different HSDPA Equalizers Under the Measured Outdoor Channels – C. Li, J. Wu, J. Huang, I. Tang

- A Macroanalysis of Hsdpa Receiver Models – A. Bastug, D. Slock
- Iterative Methods for the G-RAKE Receiver in HSDPA – N. Murugesapillai, T. Baykas, A. Yongacoglu
- System Performance of HSDPA with MMSE Receiver – F. Wangt, R. Lovet, A. Ghosht
- Evaluation of Throughput and Coverage Employing Multipath Interference Canceller in High-Speed Downlink Packet Access in Multipath Fading Channel – K. Higuchi, A. Morimoto, S. Abeta, M. Sawahashi
- Performance Comparison of Chip-level Equalizers in the HSDPA System – M. Park, W. Lee, M. Lee, H. Lee
- Pilot-aided Adaptive Chip Equalizer Receiver for Interference Suppression in DS-CDMA Forward Link – F. Petre, M. Moonent, M. Engels, B. Gyselinckx, H. De Man
- Pilot Cancellation Iterative Chip Equalization for MIMO HSDPA – S. Fei, X. Liu, Q. Wu, J. Wang
- HSDPA Terminal – M. Harteneck, C. Luschi
- Evaluation of New NLMS and RLS Chip Equalizers using Tentative Decision Data for HSDPA Systems – D. Ogawa, T. Dateki, H. Furukawa
- Power Control technique for High Speed Downlink Packet Access system – A. Mousa
- Variable Orthogonality Factor: a Simple Interface between Link and System Level Simulation for High Speed Downlink Packet Access – A. Seeger, M. Sikora, A. Klein
- Advanced Receiver Architectures for HSDPA and their Performance Benefits - A. Ghosh, R. Kobylinski
- On the Rake receiver performance – H. Boujemaa, M. Siala
- On the Performance of a Turbo-Equalizer including Blind Equalizer over Time and Frequency-selective Channel. Comparison with an OFDM System - M. Hélard, P.J. Bouvet, C. Langlais, Y.M. Morgan, I. Siaud
- On Numerical Robustness of Constrained RLS-Like Algorithms – A. L. L. Ramos, J. A. Apolinário Jr., M. L. R. de Campos