6.- SIMULACIÓN

6.1.- Introducción

Como ya comentamos en la Introducción inicial y hemos venido repitiendo en Capítulos anteriores, el objetivo de este Proyecto Fin de Carrera es la Simulación de un sistema de comunicaciones basado en el nuevo estándar para la televisión digital, DVB-T2. Hasta ahora, en los Capítulos 2, 3, 4 y 5 nos hemos limitado a exponer una serie de conceptos necesarios para asimilar de una manera correcta el funcionamiento de un sistema de comunicaciones, con conceptos tan importantes como la de modulación ortogonal o la introducción a las distintas técnicas de transmisión digital promovidas por el organismo DVB.

La finalidad de este Capítulo será la de guiar al lector a través de la simulación que se ha llevado a cabo, utilizando la aplicación Simulink de Matlab.

En primer lugar, queremos ofrecer una visión general de nuestro esquema de simulación. Ya en una segunda etapa, realizaremos una distinción entre los bloques del transmisor y los del receptor, realizando en todo momento una comparación entre nuestro esquema y el propuesto en el estándar y explicando y detallando cada uno de los bloques que hemos empleado. Explicaremos, a su vez, todos y cada uno de los parámetros con los que hemos configurado los bloques.

Para finalizar el capítulo, mostraremos al lector los resultados que se han obtenido durante la simulación.

6.2.- Visión general

El esquema que hemos seguido a la hora de realizar la simulación se muestra en la Figura 6.1:

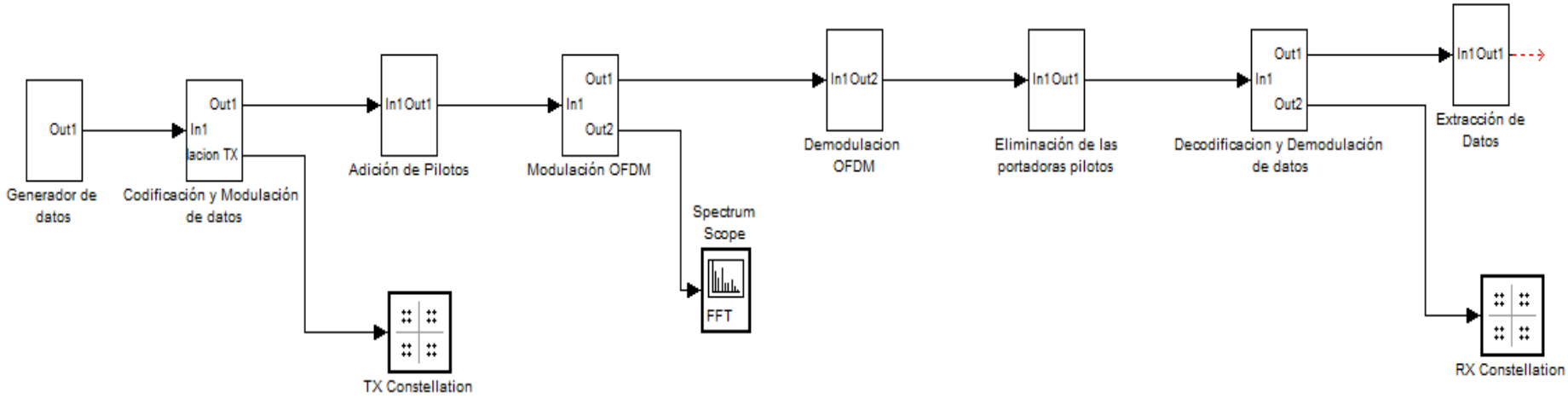

Fig. 6.1: Esquema general de modulación

Antes de empezar a comentar las principales características de los bloques que conforman el esquema seguido para la simulación del sistema de comunicaciones, queríamos dejar constancia del valor de los distintos parámetros que se han tenido en cuenta durante la elaboración de este proyecto:

- Período Elemental: T = 7/64 μs para sistemas de ancho de banda 8 MHz.
- Duración del Intervalo de Guarda: 256T.
- Tamaño de la FFT: 8k.
- Número de portadoras activas: 6817.
- Longitud de la trama T2: L_F = 270 símbolos.
- Duración de la trama: T_F = 250 ms.
- Modulación seguida: QPSK.
- Patrón de Pilotos: PP7.
- Duración de un símbolo: Ts = 924 μs.

Otra asunto a tener en cuenta, es que muchos de los bloques que hemos utilizado dependen de variables que hemos almacenado en un fichero (estos ficheros aparecerán posteriormente en un Anexo). Para que Simulink tenga en cuenta estos ficheros y cargue los mismos en Matlab, hemos proporcionado la ruta a Simulink para que al ejecutar el programa, realice la carga de estas variables automáticamente. Otra posible opción hubiera sido cargar nosotros mismos estos ficheros en Matlab mediante el comando conocido **load**.

A continuación, empezaremos a describir las principales características de los distintos bloques utilizados.

6.3.- Transmisor

Como todos sabemos, el transmisor de un sistema de comunicaciones es el conjunto de bloques encargados de generar las señales que van a propagarse a través del canal (en nuestro caso, hemos considerado un canal ideal durante el proceso de simulación) y que deben tener una serie de características en función del estándar de transmisión que estemos considerando.

La Figura 3.2 nos muestra el esquema general del transmisor, tal y como hicimos anteriormente con el esquema general.

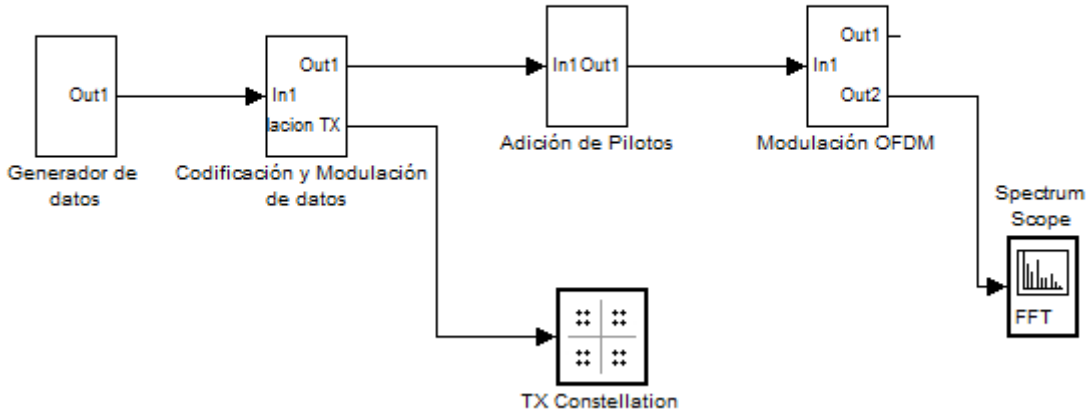

Fig. 6.2: Esquema general del transmisor

A continuación, pasaremos a describir cada uno de los bloques que componen esta primera etapa del sistema de comunicaciones. Una cosa que iremos haciendo será la de comparar el esquema propuesto por el organismo que elaboró el estándar y el esquema que hemos seguido nosotros para realizar el proceso de simulación. En principio, las diferencias serán muy pequeñas.

6.3.1.- Generador de datos

Como su nombre bien indica, este bloque será el encargado de ir generando la señal de datos para un sistema basado en DVB-T2. Para ello, primero generaremos una señal de datos con el bloque "Random Integer", para posteriormente actuar sobre ella como se muestra en la siguiente Figura:

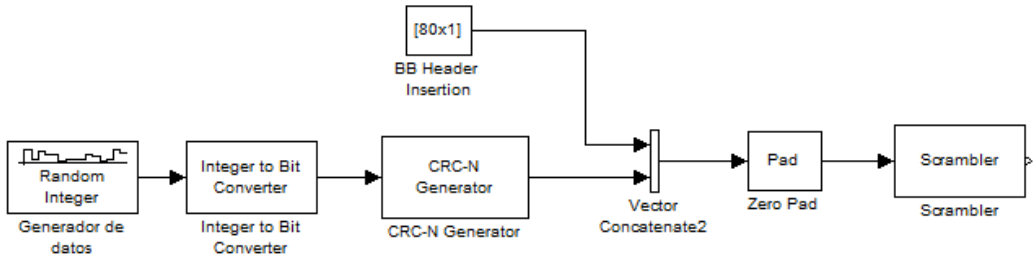

Fig. 6.3: Esquema del Generador de datos

El esquema propuesto por el estándar es el siguiente:

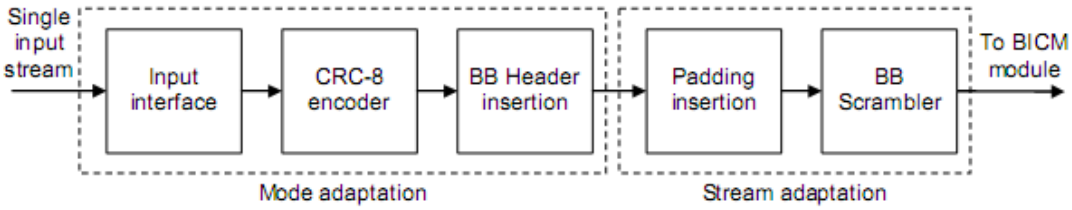

Fig. 6.4: Esquema del Generador de datos propuesto por el estándar

Pasamos a estudiar cada uno de los bloques que conforman el Generador de datos.

6.3.1.1.- Generador de los datos

Este bloque, "Random Integer Generator", será el encargado de generar la señal que posteriormente enviaremos por el canal de transmisión.

Los parámetros que hemos utilizado para este bloque son los siguientes:

- **M-ary Number:** indica el rango de valores del vector de salida. En nuestro caso, como vamos a implementar una señal QPSK, este valor será **4**.
- **Initial Seed:** Las señales generadas por este bloque necesitan un valor con el que generar los datos. Realmente con este bloque obtenemos datos

pseudo aleatorios. Por tanto para este campo no existe un valor predeterminado.

- **Sample Time:** Con este valor indicamos el período de muestreo de nuestra señal. El valor que nosotros hemos utilizado lo hemos obtenido de la siguiente manera. Nos dice el estándar que la duración de un símbolo de datos para una modulación OFDM de 8k para sistemas de ancho de banda de 8 MHz es de 896 μs. Esta operación sale de multiplicar el número de portadoras del símbolo OFDM, en nuestro caso, 8192 por el valor del período elemental, que como vimos anteriormente, en nuestro caso es de T = 7/64 μs (8192 * T = 896 μs). Por otra parte, hemos tomado el valor del intervalo de guarda como 1/32. De esta manera, la duración del símbolo total es de 896 μs + 1/32 veces esta duración, es decir, 924 μs. Para obtener el periodo de muestreo, tendremos que dividir el tiempo de duración total del símbolo por el número de datos que se vayan a enviar, en nuestro caso, como veremos posteriormente, es de 3948. Todo esto nos lleva a que nuestro periodo de muestro es de 924e-6/3948.

- **Frame-based outputs:** Para la generación de señales OFDM en Simulink necesitamos medirlas en tramas. Por ello, deberemos marcar esta opción.

- **Samples per Time:** Es el número de muestras que se transmiten en cada trama. El estándar nos dice que tenemos que utilizar tramas de 188 bytes. Nosotros generaremos, simultáneamente, 21 bloques de 188 bytes, es decir, nuestro valor para este campo será el de 21*188 = **3948**. El motivo es el siguiente: necesitamos a la entrada del bloque del codificador BCH, que se verá más adelante, 32208 bits. Si pasásemos los 188 bytes a bits obtendríamos 1504 bits, lo cual es insuficiente. Tendríamos que añadir un buffer que almacenase los bits hasta llegar a ese valor, lo cual provocaría en el sistema retrasos innecesarios. Es por ello por lo que hemos optado por la primera solución.

- **Output data type:** Como para este primer bloque no nos hace falta precisión, hemos usado **uint8**.

6.3.1.2.- Integer to Bit Converter

Como su nombre indica, este será el bloque encargado de pasar de bytes a bits. Sus parámetros son:

- **Number of bits per Integer:** Como ya hemos dicho, queremos pasar de bytes a bits, luego este valor tiene que ser 8.
- **Output data type:** Como posteriormente necesitamos trabajar a nivel de bits, su salida será de tipo **boolean**.

6.3.1.3.- CRC-N Generator

El estándar DVB-T2 especifica que debemos incluir en la generación de la señal un algoritmo que permita comprobar la fiabilidad y la no alternación de los datos. Aquí es donde entra nuestro "CRC-N Generator". No obstante, no en todos los casos tiene que utilizarse este algoritmo.

Sabemos que existen en DVB-T2, hasta cuatro opciones para el formato de la señal de entrada. Son:

- Transporte en cadena (Transport Stream, TS), caracterizado por tamaño fijo de los paquetes de usuario, de 188 x 8 bits, donde el primer byte es un byte de sincronismo.
- Encapsulado Genérico de cadenas (Generic encapsulated Stream, GSE), caracterizado por una longitud variable de los paquetes de usuario.
- Cadenas continuas genéricas (Generic Continuous Stream, GCS), que al igual que en el caso anterior, también presenta una longitud variable de paquetes.
- Cadenas genéricas de longitud fija de paquetes (Generic Fixe-length Packetised Stream, GFPS). Se diferencia del primer caso en que los paquetes de usuario pueden tomar valores de hasta 64K.

Nosotros, para la simulación, nos hemos decantado por la primera de las opciones, es decir, por TS. Sólo para esta opción, se ha definido el uso del algoritmo CRC.

De acuerdo a lo establecido por el estándar, los campos de este bloque son los siguientes:

- **CRC method:** Como hemos comentado, el estándar recoge que sea CRC-8. El polinomio generador utilizado, de acuerdo con el estándar es: $G(x) = x^8 + x^7 + x^6 + x^4 + x^2 + 1$.
- **Inital States:** Se inicializan los registros internos con valor cero.
- **Checksums per Frames:** Es el número de comprobaciones que se realiza por trama. Nosotros le hemos dado a este campo el valor **1**.

6.3.1.4.- Base Band (BB) Header Insertion

El estándar recoge que tiene que añadirse una cabecera en banda base para describir el formato de los datos que se transmiten. Esta cabecera tendrá una longitud fija de 10 bytes, o lo que es lo mismo, una longitud de 80 bits.

Los campos de esta cabecera son los siguientes:

- **MATYPE** (2 bytes): describe el formato de la cadena de entrada.
  - **Primer byte**
    - **Campo TS/GS** (2 bits), define el formato de la cadena de entrada: GFPS(00), TS(11), GCS(01) o GSE(10)
    - **Campo SIS/MIS** (1 bit), si se transmite una única portadora (1) o múltiples portadoras (0).
    - **Campo CCM/ACM** (1 bit), si es codificación y modulación constante (1) o si es codificación y modulación variable (0).
    - **ISSYI** (1 bit), es el indicador de sincronización de la cadena de entrada: 1 si está activo y 0 si no lo está.
    - **NPD** (1 bit), para indicar si está activo el borrado de paquetes vacíos: 1 si está activo, 0 si no lo está
    - **EXT** (2 bits): para uso futuro. Se ponen a cero.
  - **Segundo byte**

    Sólo se usa cuando el campo ISI/MIS = 0. En caso contrario, está reservado para un uso futuro.

Así, el valor de campo MATYPE en nuestro caso será: 1111000000000000.

- **UPL** (2 bytes): Indica la longitud del paquete de usuario en bits. En nuestro caso es de 188 bytes, por tanto: 0000001011110000.
- **DFL** (2 bytes): Indica la longitud del campo de datos en bits. En nuestro caso son 3948 bytes: 1111011011000000.
- **SYNC** (1 byte): Copia del sincronismo de byte: 01000111 (47_{HEX}).
- **SYNCD** (2 bytes): 0000000000000000. Indica que la trama está alineada con el comienzo de los datos.
- **CRC-8 MODE** (1 byte). CRC-8 detecta errores en los primeros 9 bytes de la cabecera. Toma el valor: 00000000.

La cabecera se le añade a la trama de datos a través del bloque de Simulink "Vector Concatenate".

6.3.1.5.- Vector Concatenate

Como hemos comentado anteriormente, este bloque se usa para concatenar la cabecera en banda base con la trama de datos.

Los parámetros usados son:

- **Number of inputs:** Con este campo indicamos el número de vectores a concatenar. En nuestro caso, obviamente, son **2**.
- **Mode:** Indicamos el tipo de concatenación que se llevará a cabo. Nosotros hemos tomado **Mutidimensional Array**
- **Concatenate dimensión:** Una vez se ha realizado la concatenación, queremos que nos quede un único vector. Por eso el valor dado a este campo es **1**.

6.3.1.6.- Zero Pad

Ya hemos comentado con anterioridad que a la entrada del codificador BCH necesitábamos 32208 bits. Una vez que hemos añadido la cabecera en banda base a la trama de datos, la señal que tenemos tiene una longitud de 31672 bits, algo lejos de los 32208 bits que hemos dicho que necesitamos. Por tanto, nos hace falta un bloque que nos aporte esta diferencia.

Esta es la función del "Zero Pad". Añade una cantidad extra, bien de ceros o bien de unos, a la señal que tiene a la entrada.

Los parámetros utilizados son los siguientes:

- **Pad over:** Como trabajamos con vectores columnas, el valor de este campo no puede ser otro que **Columns**.
- **Pad value source:** Nos interesa indicarle dónde queremos que nos añada la información "extra". Por ello seleccionamos la opción **Especify via dialog**.
- **Pad Value:** Queremos que añada **ceros**.
- **Output column mode:** Con este parámetro le indicamos la cantidad de ceros que tiene que añadir. Hay dos opciones: o bien añade cero hasta la siguiente potencia de dos o bien le especificamos hasta qué valor tiene que añadir datos. Elegimos la segunda opción, luego marcamos **User-specified**.
- **Column size:** Esta opción sólo aparece cuando en el campo anterior hemos marcado la segunda opción. Con este parámetro indicamos la longitud de la señal que queremos a la salida. Nosotros, como ya hemos comentado, ponemos **32208**.
- **Pad signal at:** Indica el lugar dentro de la trama donde queremos añadir los ceros. Nos interesa tenerlos al final, por lo que hemos marcado la opción **End**.
- **Action when truncation occurs:** En este caso, como estamos seguros de que no se va a producir, marcamos **None**.

6.3.1.7.- Scrambler

Se utiliza para aleatorizar la BBRAME. La secuencia de aleatorización estará sincronizada con la BBFRAME, comenzando por el bit más significativo y terminando después del bit $K_{bch.}$

El polinomio para pseudo aleatorizar la secuencia binaria viene dado por: $G(x)=1 + x^{14} + x^{15}$. Por otra parte, precargamos los registros internos del bloque con la secuencia: 100101010000000.

Los parámetros utilizados son:

- **Calculation base:** será **2**, ya que estamos trabajando en binario.
- **Scramble polynomial:** en este parámetro introducimos el polinomio en binario. Así tendremos: 1100000000000001.
- **Initial states:** es la condición inicial para los registros internos. Como ya hemos dicho, le damos el siguiente valor: 100101010000000.

Con este bloque, finaliza esta primera etapa del transmisor.

6.3.2.- Codificación y Modulación de datos

Durante esta segunda etapa del transmisor, la señal generada con anterioridad es sometida a distintos procesos de codificación con el objetivo de dotarla de una mayor robustez. Una vez la señal ha sido codificada, procederemos a su modulación (en nuestro caso, como ya indicamos anteriormente, se tratará de una modulación QPSK), para posteriormente aplicar sobre la misma distintos procesos de entrelazado. Así, si existe una ráfaga de datos erróneos, con los diferentes procesos de entrelazado que se aplican sobre la señal, podremos dispersar estos datos con el objetivo de que no afecten únicamente a un símbolo.

El esquema que hemos seguido en esta segunda etapa es el siguiente:

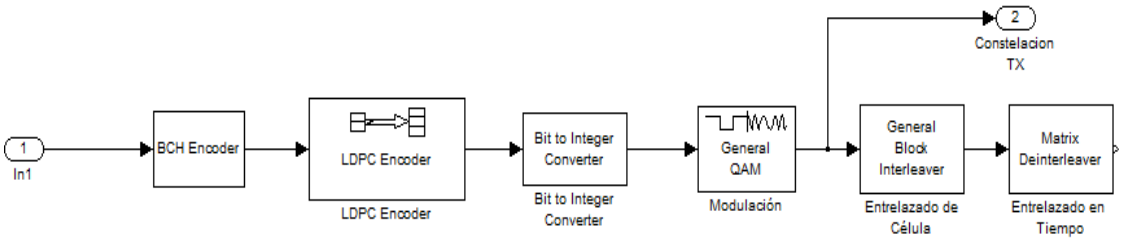

Fig. 6.5: Esquema de la etapa Codificación y Modulación de datos

Comparándolo con el esquema propuesto en el estándar, vemos que ambos son muy parecidos.

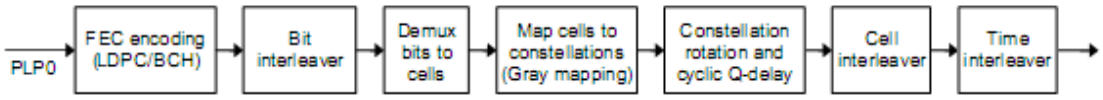

Fig. 6.6: Esquema de Codificación y Modulación de datos propuesto por el estándar

6.3.2.1.- Codificación FEC

Cada BBFRAME transmitida (K_{bch} bits) será procesada por un subsistema de codificación FEC (BCH/LDPC) para generar una FECFRAME (N_{ldpc} bits). Los bits de comprobación de paridad (BCHFEC), introducidos por el codificador BCH se añadirán después de la BBFRAME, mientras que los bits de comprobación de paridad (LDPCFEC) generados por el codificador LDPC se añaden detrás del campo BCHFEC. La **Fig.6.7** muestra el formato de la trama una vez llevada a cabo la codificación.

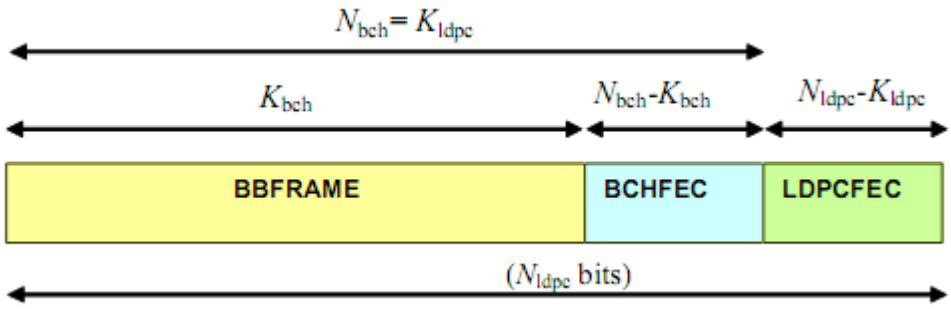

Fig. 6.7: Formato de los datos antes de la etapa de modulación

6.3.2.1.1.- Codificación BCH

Como acabamos de comentar, todas las tramas BBFRAME deben pasar por el codificador BCH, generándose una serie de bits de comprobación de paridad que permitirán al sistema comprobar si han existido errores durante la transmisión. En nuestro caso, el estándar define que para una tasa de código LDPC igual a 1/2, que es el que nosotros hemos utilizado, el codificador BCH genera 192 bits de comprobación de paridad, que se añadirán al final de la trama de datos. Así a la salida de este bloque tendremos una señal de 32400 bits.

Los parámetros que hemos utilizado han sido:

- **Generator Polynomial Vector**: Como su nombre indica, debemos indicar en este campo cuál es el polinomio generador. Este polinomio viene definido en el estándar.

6.3.2.1.2.- Codificación LDPC

Este bloque es muy similar al visto anteriormente. Su función es la de generar bits de comprobación de paridad, que se añadirán al final de la trama, que permiten conocer al sistema si se han producido errores durante la transmisión.

Para este bloque, nos hemos apoyado en una serie de funciones que ya venían implementadas en Matlab (versión R2007b). De esta forma, sólo hemos tenido que llamar a la función, pasándole la tasa de codificación LDPC que hemos empleado durante la simulación (en nuestro caso 1/2).

Parámetros utilizados:

- **Parity-check Matrix**: Como acabamos de comentar, llamamos en este campo a la función ya recogida en Matlab, pasándole el valor adecuado de la tasa de codificación LDPC: **dvbt2ldpc (1/2)**.

6.3.2.2.- Bit to Integer Converter

A partir de este bloque queremos trabajar a nivel de símbolos transmitidos y no de bits como lo veníamos haciendo con anterioridad. Por tanto, la finalidad que presenta este bloque es, sencillamente, la de pasar de bits a símbolos. Para ello, sólo será necesario tener en cuenta la modulación que va a seguirse durante la simulación.

Los parámetros que deberán tenerse en cuenta son los siguientes:

- **Number of bits per Integer:** Como ya hemos dicho, queremos pasar de bits a símbolos, luego en nuestro caso este valor tiene que ser 2.
- **Output data type:** En cuanto al tipo de salida de este bloque, dado que estamos seguros que todos van a ser enteros positivos, ponemos **uint8**.

6.3.2.3.- Modulación

Este bloque "General QAM Modulator" será el encargado de llevar a cabo la modulación de nuestra señal para convertirla en una señal QPSK. Los parámetros utilizados han sido:

- **Signal Constellation:** Le damos a este campo las posiciones para una constelación QPSK, ya normalizada, de acuerdo al mapeado indicado en el estándar.
- **Output data type:** Como los datos pueden ser flotantes y de gran longitud, elegimos la opción **double**.

La constelación que se recibirá tendrá que ser:

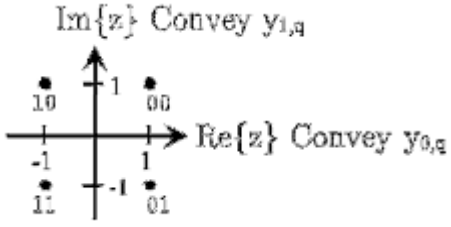

Fig. 6.8: Esquema de la constelación transmitida

6.3.2.4.- Entrelazado de célula

El pseudo aleatorio entrelazado de células, ilustrado en la **Fig. 3.13**, dispersa uniformemente las células de la FEC para asegurar en el receptor una distribución no correlada de posibles distorsiones de canal y de interferencias que hayan incidido sobre la FEC.

Para llevar a cabo este proceso, utilizaremos un bloque general de entrelazado, "General Block Interleaver", al que le pasamos la variable en la que hemos almacenado el vector de entrelazado utilizado y que hemos desarrollado de acuerdo al algoritmo que aparece recogido en el estándar y que incluiremos más adelante en el Anexo. Este bloque lo que hace es reordenar el vector de la entrada de acuerdo al parámetro que se le pasa.

6.3.2.5.- Entrelazado en Tiempo

Su función es muy similar a la del Entrelazado de células, ya que el objetivo de este bloque es impedir que un símbolo de la trama se vea completamente afectado por un error.

El bloque tiene que actuar de la siguiente forma:

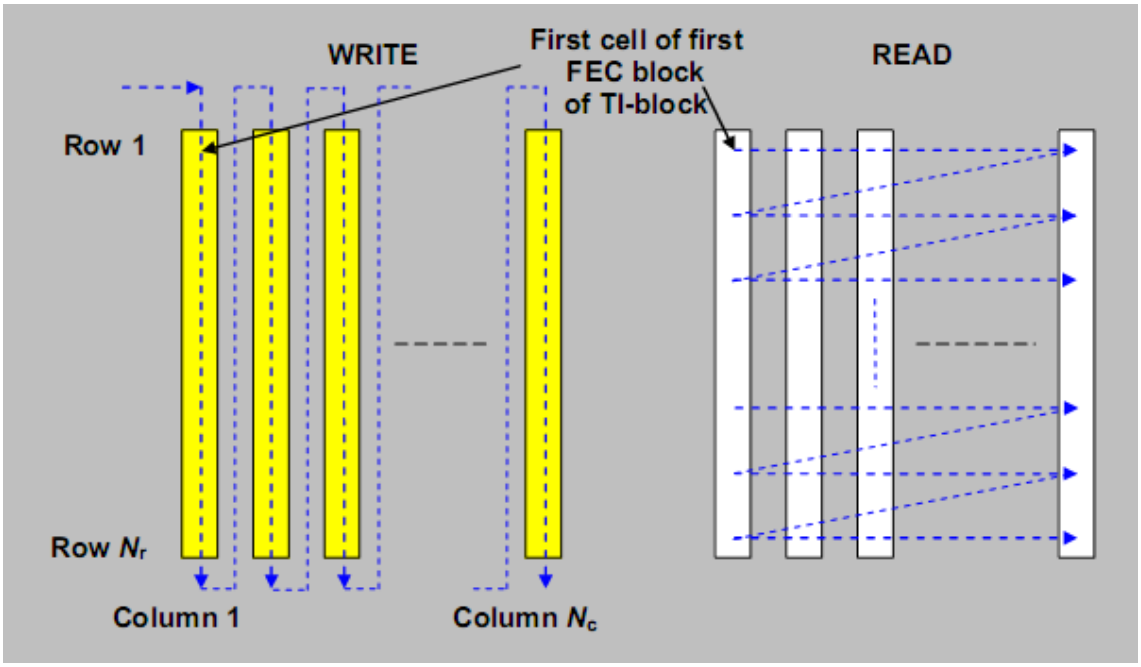

Fig. 6.9: Entrelazado en Tiempo

Para conseguir que nuestro bloque de Simulink realice la misma función, nos hemos apoyado en el bloque "Matriz Deinterleaver". Este bloque almacena el vector de entrada en una matriz por columnas y saca el vector de salida fila a fila.

Los parámetros que tenemos que utilizar para este bloque también son facilitados por el estándar DVB-T2. Así para un número de células de 32400 por bloque LDPC, necesitaremos un número de filas, N_r, igual a 6480.

Los parámetros del bloque son los siguientes:

- **Number of Rows:** que como hemos dicho viene dado por el estándar, siendo en nuestro caso **6480**.
- **Number of Columns:** este dato es fácilmente calculable con los datos que ya sabemos. Si el número de células a meter en la matriz es de 32400 y tenemos 6480 filas, el número de columnas vendrá dado por 32400/6480 = **5**.

Con este bloque ya tenemos finalizado la etapa de Codificación y Modulación de datos. Ya a partir de este momento, tenemos nuestra señal generada en la primera etapa perfectamente codificada, modulada y entrelazada, de forma que evitamos que los errores se propaguen por un solo símbolo.

6.3.3.- Adición de Portadoras Pilotos

Ya hemos hablado con anterioridad de las portadoras pilotos. Como sabemos, las portadoras pilotos son portadoras dentro del símbolo OFDM moduladas con información de referencia cuyos valores son conocidos por el receptor.

La finalidad de estas portadoras es la de ser usadas para sincronización de trama, sincronización en frecuencia, sincronización en tiempo, estimación del canal y/o para la identificación del modo de transmisión.

En este punto veremos cómo se generan estas portadoras y cómo se añaden al resto de datos del símbolo.

El esquema que seguiremos durante todo este aparatado es el siguiente:

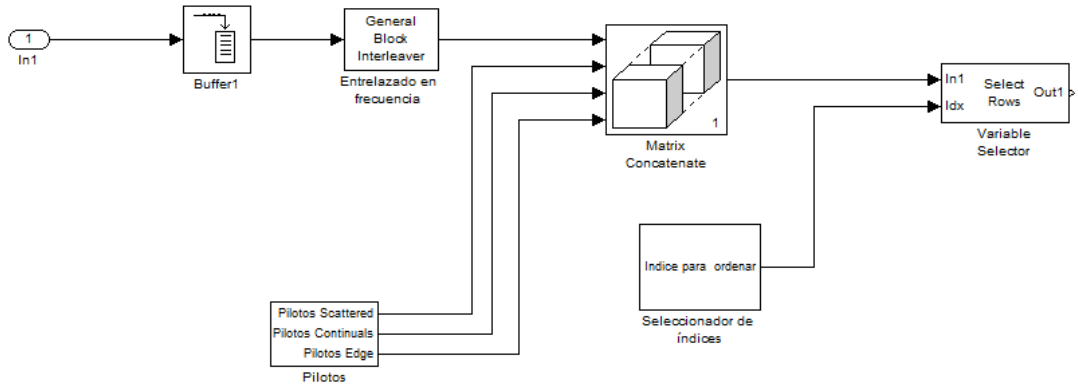

Fig. 6.10: Esquema de Adición de pilotos

Como hemos venido haciendo hasta ahora, comparamos con el esquema propuesto por el estándar:

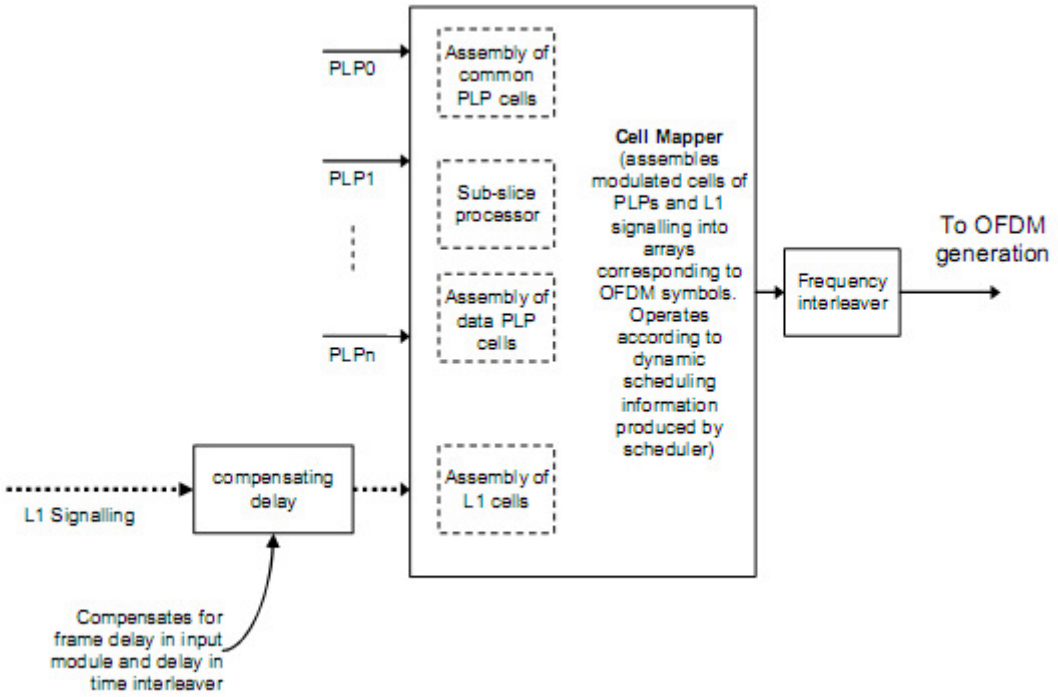

Fig. 6.11: Esquema del Generador de tramas propuesto por el estándar

El esquema propuesto por el estándar se correspondería con los dos primeros sub-bloques del nuestro, de ahí la diferencia existente, ya que nosotros incluimos en el esquema de Adición de Pilotos, parte de los sub-bloques que aparecerán en el esquema siguiente del estándar.

6.3.3.1.- Buffer

El objetivo de este Buffer es el de almacenar la información que le llega en forma de FEC, recordemos que eran 32400 símbolos, e ir sacando la información en tramas de 6698 símbolos. Esta cantidad (6698) es el número de portadoras activas en un símbolo OFDM. Por tanto, a partir de este sub-bloque, empezaremos a trabajar a nivel de símbolo OFDM.

Los parámetros utilizados son:

- **Output Buffer size:** Ya hemos dicho que este valor será **6698**.
- **Buffer Overlap:** Queremos que la información salga por vectores pero que ésta no se repita. Por tanto, el valor de este campo será **cero**.
- **Initial Conditions:** se pondrá a cero, ya que no queremos que se transmita nada mientras el Buffer se llena.

6.3.3.2.- Entrelazado en Frecuencia

El objetivo del entrelazado en frecuencia, operando en las células de datos dentro de un símbolo OFDM, es mapear las células de datos procedentes del Buffer en las portadoras activas en cada símbolo. El procedimiento de este entrelazador ya fue explicado en el punto **3.8.1**.

Para llevar a cabo esta función de manera adecuada, haremos uso del "General Block Interleaver", cuyo funcionamiento ya se explicó en el punto **6.3.2.4**.

Al igual que en el caso anterior, le pasamos una variable en la que hemos almacenado el vector de entrelazado. Dicho vector viene de la implementación del algoritmo fijado en el estándar y que se adjuntará posteriormente en el Anexo.

6.3.3.3.- Pilotos

Este sub-bloque será el encargado de generar las portadoras pilotos que posteriormente se añadirán a los símbolos de datos.

El esquema que hemos seguido para la generación de estas portadoras es el siguiente:

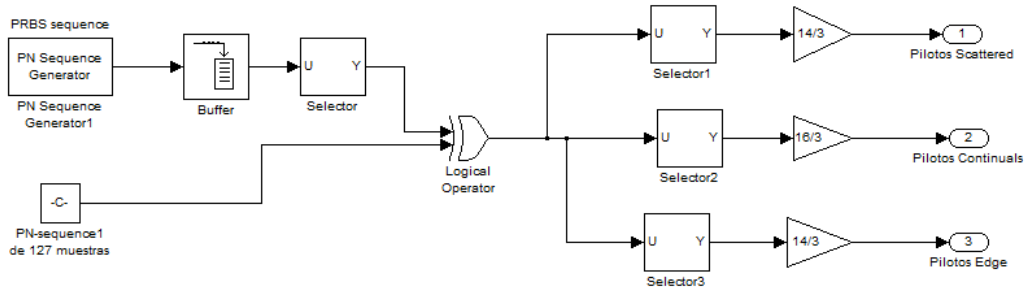

Fig. 6.12: Esquema de la Generación de Pilotos

Pasamos a explicar cada uno de los sub-bloques que componen el esquema de Generación de Pilotos.

6.3.3.3.1.- PRBS Sequence

Para generar esta secuencia, utilizamos el bloque de Simulink "PN Sequence Generator" con los siguientes parámetros:

- **Generator Polynominal:** En este campo tenemos que incluir el vector generador. Dicho vector, indicado en el estándar, es: **[11 2 0]**.
- **Initial States:** Con este campo inicializamos los registros internos del bloque. Este campo, según el estándar, vale: **[1 1 1 1 1 1 1 1 1 1 1 1]**.
- **Output mask source:** En este campo tenemos que indicar si queremos que el bloque genere la señal (Dialog Parameter) o bien modulará una señal entrante (Input Port). En nuestro caso, queremos que nos genere la señal, por lo que marcamos la primera opción.

- **Output mask Vector:** Indica el retraso originado por el bloque. Por tanto, nos interesará que este campo valga **0**.
- **Sample Time:** Es el tiempo de muestreo. Lo ponemos a -1, ya que no nos interesa.
- **Frame-based outputs:** Ya hemos dicho que nos interesa que la señal de salida esté basada en trama, luego tenemos que tener marcado esta opción.
- **Samples per Frame:** Es el número de muestras por trama. Viene dado por la suma de todas las portadoras pilotos. Así, el valor de este campo es de **119**.
- **Output data Type:** Especificamos el tipo de datos que va a tener el vector que se genere. Se escogerá el tipo **boolean**.

6.3.3.3.2.- PN-Sequence

Para generar la secuencia que nos interesa para este bloque (mirar el punto **3.8.2.1.2**), utilizamos el bloque "Constant". Los parámetros que hemos utilizado son los siguientes:

- **Constant Value:** En este campo incluimos la parte de la secuencia general que nos interesa en binario. En este caso cogemos los primeros 119 bits.
- **Sampling Mode:** Al igual que nos ocurriese en el caso anterior, nos interesa que a la salida haya tramas, en vez de muestras sueltas. Por este motivo marcamos **Frame based**.
- **Frame Period:** Igualmente marcamos esta opción con **-1**.

6.3.3.3.3.- Selector

Con este bloque pretendemos realizar una reordenación del vector que tiene a la entrada. El motivo viene dado por la secuencia de referencia $r_{l,k}$, como vemos en el punto **3.8.2.1**. Así, según vemos, la secuencia PRBS antes de realizar la operación Xor, está desplazada en secuencia un valor de K_{ext}. El valor de K_{ext} para una modulación 8k es de 48.

Los parámetros utilizados son:

- **Index Option:** Con él seleccionamos la forma en el que le decimos al bloque como queremos realizar la selección. Nosotros hemos elegido la opción **Index vector**.
- **Index:** Ponemos en este campo el orden en el que queremos que este bloque nos saque el vector de salida. Así, para el caso de las *scattered pilots* pondremos **[1:72]**; para las pilotos continuas, **[73:117]**; mientras que las pilotos de fin de símbolo, pondremos **[118:119].**
- **Input port size:** Le indicamos el tamaño del vector de entrada. Así, pondremos en este campo el valor **119**, que como vimos es el tamaño de la secuencia PRBS.

6.3.3.3.4.- Logical Operator

Este bloque se va a encargar de realizar la operación entre la secuencia PRBS y la secuencia PN. De acuerdo, con el estándar, esta operación será una Xor. Los parámetros con los que vamos a configurar este bloque son:

- **Operator:** En nuestro caso, seleccionamos Xor, que es la operación que nos interesa.
- **Number of input ports:** Queremos multiplicar las secuencias PRBS y PN, por lo que este campo valdrá **2**.
- **Icon shape:** Marcamos **distinctive**, para que nos muestre el bloque con el símbolo de la operación de la Xor.
- **Sample Time:** Ponemos -1, ya que este parámetro no es interesante.

6.3.3.3.5.- Selectors

El bloque Selector vimos anteriormente que lo usábamos para llevar a cabo la reordenación del vector que tenía a su entrada. Otra aplicación de este bloque, que es la que nos interesa en esta ocasión, es la de seleccionar un conjunto de muestras dentro de un rango mayor.

El objetivo de esta acción es la de seleccionar, dentro del conjunto de la secuencia para las portadoras pilotos, aquellas muestras que vamos a usar para generar las portadoras pilotos continuas, las que vamos a utilizar para generar las portadoras pilotos dispersas y las que van a corresponder a las portadoras de los extremos.

Los parámetros que hemos empleado son:

- **Index option:** Al igual que antes, marcamos la opción **Index Vector**.
- **Input port size:** Define el tamaño del vector de entrada
- **Index:** Hemos dejado este parámetro para el final ya que es el único diferente en los tres bloques que tenemos. Como ya hemos comentado, con este campo vamos a seleccionar el subconjunto de muestras que nos interesan. Este dato es proporcionado por el estándar. Así, este nos dice que:
  - El número de portadoras pilotos en los extremos será 2.
  - El número de portadoras continuas para una FFT de 8k es 44.
  - Y el número de portadoras dispersas (scattered) viene dada por la siguiente expresión:

    $$k \bmod(D_x \cdot D_y) = D_x \bmod(l \cdot D_y), \text{ para portadora normal}$$

    $$(k-k_{ext}) \bmod(D_x \cdot D_y) = D_x \bmod(l \cdot D_y), \text{ para portadora extendida}$$

    donde "l" es el símbolo OFDM, $k \in [k_{min}, k_{max}]$, y D_x y D_y valen para el patrón de pilotos PP7 (que recordamos que es el que estamos usando) 24 y 4, respectivamente. Ello nos lleva a 72 portadoras.

  Así, el dato que tendremos que poner en cada uno de los bloques será, precisamente, el número de portadoras de cada tipo.

6.3.3.3.6.- Ganancia

El objetivo de este bloque es el de dar cierta ganancia a cada una de las subportadoras pilotos, datos todos ellos facilitados por el estándar. Haciendo un resumen, el estándar nos dice que la ganancia para cada tipo de portadoras pilotos es la siguiente:

Tipo de Portadora Piloto	Ganancia
Portadoras Pilotos Continuas	7/3
Portadoras Pilotos Dispersas	8/3
Portadoras Pilotos en los Extremos	$\sqrt{31}/5$

Será estos valores de ganancia los que tendremos que incluir en el parámetro "**Gain**" de este bloque. El resto de parámetros los dejamos tal cual.

6.3.3.4.- Matrix Concatenate

La función de este bloque es la misma que la que se explicó en el punto **6.3.1.5** en el bloque Vector Concatenate, es decir, su función es la de concatenar distintas tramas que tiene a la entrada. El orden es el siguiente: numerando las entradas de forma decreciente, primero extrae la entrada que se encuentra en la primera posición, luego la de la posición 2 y así sucesivamente.

Los parámetros son los mismos que los utilizados en el caso anterior. La única diferencia es que en el campo "**Number of Inputs"** tendremos que poner el número de entradas en el bloque, esto es, **4**.

6.3.3.5.- Seleccionador de Índices

En este apartado vamos a generar un subsistema que escoja, en cada intervalo de tiempo de símbolo (en nuestro caso 924 μs), un vector con las posiciones que deberán ocupar, tanto las portadoras pilotos como los datos transmitidos, en cada trama generada. Para realizar dicha función, nos ayudaremos de los siguientes bloques:

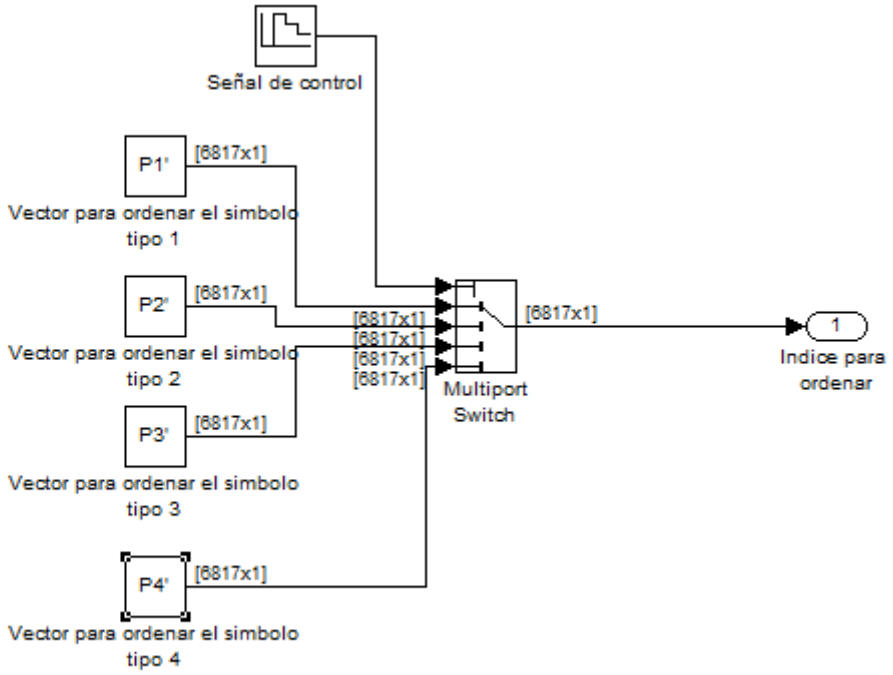

Fig. 6.13: Esquema del Seleccionador de Índices

Pasamos a describir las principales propiedades de estos bloques:

6.3.3.5.1.- Vector para ordenar los símbolos

Cada uno de estos cuatro bloques "Constant" va a indicar la posición tanto de las portadoras de datos como de las portadoras pilotos dentro de la trama. Los parámetros utilizados son:

- **Constant Value:** En este campo indicamos la posición tanto de las portadoras como de los datos. En nuestro caso, hemos indicado el nombre de las variables donde hemos almacenado estas posiciones. Se adjuntará posteriormente en el Anexo el valor de cada una de estas variables.

Los demás parámetros no son importantes.

6.3.3.5.2.- Multiport Switch

Este bloque será el encargado de seleccionar en cada intervalo de tiempo de símbolo una de las secuencias transmitidas por los bloques "Constant". El "Multiport Switch" necesita para su funcionamiento una señal de control que le diga cuándo y cómo debe cambiar la entrada. Esta señal debe ser generada desde el exterior, como veremos en el siguiente punto.

En cuanto a los parámetros utilizados destacamos:

- **Number of inputs:** En este campo indicamos el número de señales a la entrada de este bloque. En nuestro caso será de **4**.
- **Used zero-based indexing:** Con este campo indicamos si permitimos o no el instante cero dentro de la señal de control habilitando esta opción. Nosotros la hemos marcado.
- **Sample Time:** Por defecto, la dejamos a -1.

6.3.3.5.3.- Señal de Control

Con este bloque generamos una señal periódica e indicamos al bloque "Multiport Switch" cuando debe conmutar a una entrada. Lo que realmente estamos haciendo con esta función es generar los distintos tipos de símbolos OFDM que hay en una misma trama T2.

El bloque se ha configurado de la siguiente manera:

- **Vector of outputs value:** En este campo vamos a generar la señal que queremos que se repita en el tiempo. Queremos empezar transmitiendo el símbolo 1, luego nuestra señal será: **[0 1 2 3]**, es decir, le estamos indicando que cada Sample Time, saque primero la secuencia que haya en el primer puerto, en segundo lugar la que haya en el segundo puerto y así sucesivamente.
- **Sample Time:** En este campo irá el tiempo en el que permanecerá constante la señal. Nos interesa que la entrada se mantenga constante cada 924 μs, es decir, cada **Ts**.

Llegados a este punto tenemos la señal perfectamente "ordenada", es decir, los datos y la información piloto en sus respectivas portadoras, y lo único que falta para poder transmitir la señal es proceder a su modulación. Este apartado lo vemos a continuación.

6.3.3.6.- Variable Selector

Con este bloque queremos seleccionar y reordenar las filas de la señal de entrada de acuerdo a lo especificado por el puerto ld_x, que en nuestro caso le estamos pasando las posiciones de los datos y de las portadoras pilotos para los distintos símbolos OFDM.

Los parámetros que hemos utilizado son los siguientes:

- **Number of input signal:** tenemos un vector a la entrada de este bloque, luego ponemos **1**.
- **Select:** Tenemos un vector columna, luego nos interesa reordenar las filas. Seleccionamos **Rows**.
- **Select mode:** determina si el bloque usa los mismos índices para todas las entradas o si emplea diferentes índices, en función de la entrada. Hemos seleccionado la opción **Variable**.
- **Index mode:** Indicamos si queremos que la primera muestra se contabilice como índice 0 o índice 1. Marcamos la opción **One-based**.

6.3.4.- Modulación de la Señal

En este apartado vamos a ver la modulación de una señal T2. Hasta ahora, tenemos una señal con 6817 portadoras (6698 portadoras de datos, 2 portadoras pilotos de fin de símbolo, 44 portadoras pilotos continuas y 72 portadoras dispersas). Para realizar la modulación, los símbolos deben tener 8192 portadoras.

El esquema que hemos seguido para esta última parte del transmisor, es el siguiente:

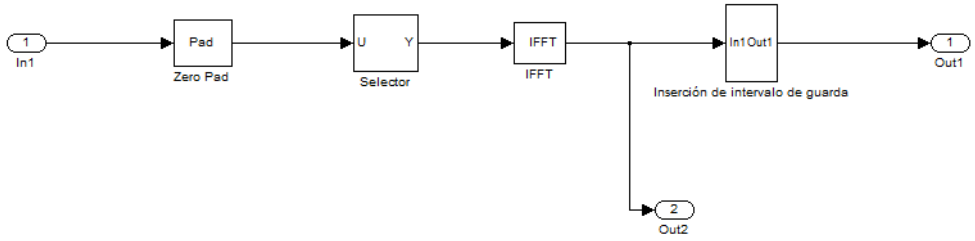

Fig. 6.14: Modulación de la señal

Comparándolo con el esquema propuesto por el estándar:

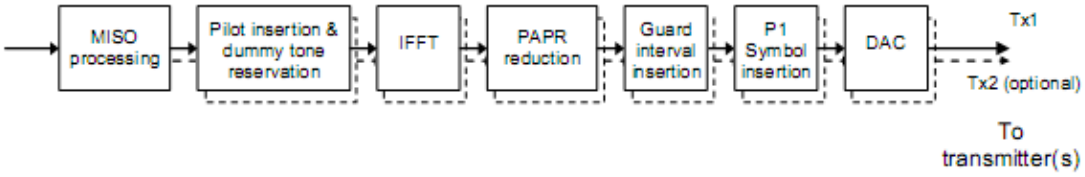

Fig. 6.15: Esquema de Modulación de la señal propuesto por el estándar

Las diferencias más importantes entre ambos esquemas fueron descritas en el punto **6.3.3**. Así, nosotros en nuestro esquema hemos considerado la inserción de pilotos dentro del esquema Generador de Tramas. Por lo demás, a grandes rasgos, ambos esquemas son idénticos.

Vamos a empezar a describir las principales características de cada uno de nuestros bloques.

6.3.4.1.- Zero Pad

Como hemos comenzado diciendo en el punto **6.3.4**, tenemos, justo antes de realizar el proceso de modulación, una señal con 6817 portadoras. Sabemos que para poder aplicar la anti-transformada discreta de Fourier, el número de portadoras de los símbolos debe tener base 2. Por tanto, el objetivo de este bloque será añadir información (en nuestro caso, nos interesa que se añadan ceros) a la señal de entrada hasta alcanzar el tamaño deseado.

Hemos configurado el bloque con los siguientes valores:

- **Pad over:** Como trabajamos con vectores columnas, el valor de este campo no puede ser otro que **Columns**.
- **Pad value source:** Nos interesa indicarle dónde queremos que nos añada la información "extra". Por ello seleccionamos la opción **Especify via dialog**.
- **Pad Value:** Queremos que añada **ceros**.
- **Output column mode:** Con este parámetro le indicamos la cantidad de ceros que tiene que añadir. Hay dos opciones: o bien añade cero hasta la siguiente potencia de dos o bien le especificamos hasta qué valor tiene que añadir datos. Elegimos la segunda opción, luego marcamos **User-specified**.
- **Column size:** Esta opción sólo aparece cuando en el campo anterior hemos marcado la segunda opción. Con este parámetro indicamos la longitud de la señal que queremos a la salida. En nuestro caso ponemos **8192**.
- **Pad signal at:** Indica el lugar dentro de la trama donde queremos añadir los ceros. Nos interesa tenerlos al final, por lo que hemos marcado la opción **End**.
- **Action when truncation occurs:** En este caso, como estamos seguros de que no se va a producir, marcamos **None**.

6.3.4.2.- Selector

En el bloque anterior, lo que hicimos fue añadir ceros a la señal que ya teníamos. Estos ceros se añadieron al final de la trama. Sin embargo, esta posición de los ceros no nos interesa para que el espectro nos quede centrado. Así lo que haremos será situar estos ceros justo en las posiciones medias de los símbolos y los datos y portadoras pilotos en ambos extremos. Con esta nueva reordenación de las portadoras dentro de un símbolo estaremos obteniendo, al realizar la anti-transformada discreta de Fourier, una señal en banda base.

Los parámetros más importantes con los que hemos configurado el bloque "Selector" son:

- **Number of inputs:** Sólo tenemos un vector a la entrada, luego este valor será **1**.
- **Index mode:** Seleccionamos si queremos empezar a contar el vector desde cero o desde uno. Nosotros hemos marcado la opción de **One based**.
- **Index Option.** Como ya hemos explicado varias veces, elegimos la opción **Index Vector**, ya que con ella podemos "decirle" al bloque como queremos reordenar el vector de entrada.
- **Index:** En este campo determinamos la forma de reordenar el vector. Como queremos que los ceros insertados en el zero pad queden en el centro del símbolo, hacemos: **[1:3408 6818:8192 3409:6817]**.
- **Input port size:** Seleccionamos el tamaño del vector de entrada. En nuestro caso, **8192**.

6.3.4.3.- IFFT

Queda claro que con este bloque lo que vamos a realizar es la anti-transformada discreta de Fourier. Nuestro objetivo es pasar la señal del dominio de la frecuencia al dominio del tiempo; de esta forma la señal estaría lista para poder transmitirse por cualquier canal.

Los parámetros utilizados para este bloque son los que ya venían por defecto.

6.3.4.4.- Inserción del Intervalo de Guarda

El Intervalo de Guarda o Prefijo Cíclico es una copia de la última parte del símbolo OFDM que se antepone al símbolo OFDM. De esta forma, la señal que se transmite se vuelve periódica, lo que juega un papel decisivo para evitar la ISI (interferencia intersímbolo) y la ICI (interferencia inter portadoras), aunque conlleva una reducción en la relación señal a ruido (eficiencia).

Como comentamos al principio del capítulo, hemos elegido una duración del Intervalo de Guarda de 1/32 veces la duración del símbolo. Si nuestro símbolo tiene 8192 muestras, entonces nuestro intervalo de guarda va a tener 256 muestras.

El esquema que vamos a seguir para la inserción del Intervalo de Guarda es el siguiente:

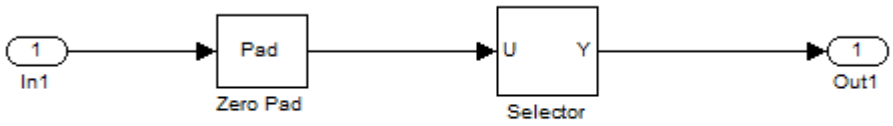

Fig. 6.16: Esquema de la inserción del Intervalo de Guarda

La función de ambos bloques es la que venimos viendo con anterioridad. Así con el bloque "Zero Pad" se van a añadir tantos ceros a la señal de entrada como queramos; en nuestro caso la señal de entrada tiene 8192 muestras y necesitamos que a la salida haya 8448 (8192 + 256 del Intervalo de Guarda), es decir, el bloque añade 256 ceros.

La función del bloque "Selector", que también hemos visto, es la de reordenar el vector que tiene a la entrada. Dijimos que nos interesaba insertar el Intervalo de Guarda antes de que empezara el símbolo y que éste era una copia de la última parte del símbolo. Por este motivo, en el campo *Index* pondremos: **[7937:8192 1:8192]**. Con ello le estamos indicando que ponga una copia de las últimas 256 muestras de la trama al principio del vector y que coloca las 8192 muestras que teníamos al principio justo detrás de estas primeras.

Llegados a este punto ya tenemos nuestra señal DVB-T2 perfectamente modulada y lista para transmitirse a través del canal.

En el siguiente punto veremos los bloques para la recepción perfecta de esta señal.

6.4.- Receptor

El receptor es el bloque encargado de extraer la señal transmitida del canal de comunicaciones. La finalidad de este bloque que presentamos a continuación, no es otra que la de recuperar los datos transmitidos inicialmente. Si los procesos de transmisión y de recepción se han realizado correctamente, los datos que obtenemos al finalizar todo el proceso de recepción deben ser iguales a los datos que se transmitieron en la primera etapa del transmisor, es decir, en el Generador de Datos.

El esquema que hemos venido siguiendo para realizar el conjunto de la recepción es:

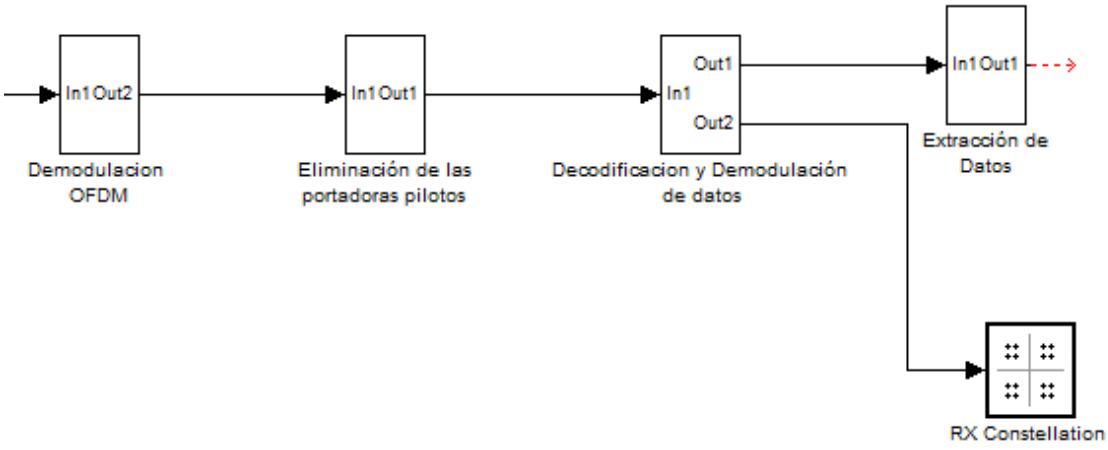

Fig. 6.17: Esquema general del receptor

Haciendo un breve resumen de lo que veremos un poco después, nuestro receptor constará, de una etapa de demodulación, en la que la señal pasará del dominio del tiempo al de la frecuencia, una etapa de extracción de las portadoras pilotos introducidas durante la transmisión y una fase de decodificación de la señal.

Vamos a empezar a ver cada una de las etapas que componen la recepción profundizando en cada una de ellas.

6.4.1.- Demodulación OFDM

Realiza el proceso inverso al visto anteriormente en el punto **6.3.4**, esto es, se va a encargar de realizar la demodulación de la señal, sabiendo que ésta va a llegar en símbolos de 8448 muestras, que recordemos que eran 8192 portadoras y 256 muestras del intervalo de guarda.

Lo primero que tendremos que hacer será "eliminar" el intervalo de guarda para proceder a la demodulación de la señal.

El esquema seguido es el siguiente:

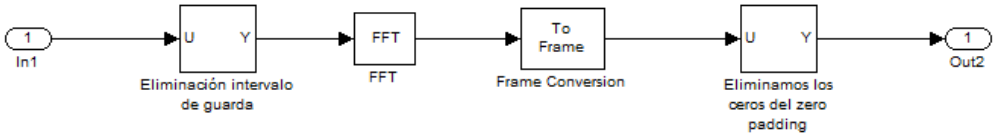

Fig. 6.18: Esquema de Demodulación de la Señal

6.4.1.1.- Eliminación del Intervalo de Guarda

Como ya explicamos en el apartado dedicado al receptor, una vez procedíamos a la modulación de los datos, introducíamos un intervalo de guarda para reducir la ISI y la ICI. Pues bien, antes de demodular los datos de cada uno de los símbolos, es necesario eliminar este intervalo de guarda.

Para realizar esta función vamos a emplear un "Selector". Los parámetros son los mismos que los que venimos explicando a lo largo de este Proyecto. La única salvedad está en el parámetro Index.

- **Index:** El intervalo de guarda ocupa las primeras 256 muestras de la señal recibida. Como nos interesa eliminarlo, seleccionamos todas las muestras a partir de la última muestra del intervalo de guarda, es decir, seleccionamos el rango **[257 8448]**.

Así, a la salida de este bloque volvemos a tener una señal de 8192 muestras. Ahora sí, podemos proceder a su demodulación.

6.4.1.2.- FFT

Con el bloque "FFT" vamos a aplicar la Transformada Discreta de Fourier a la señal que tiene a la entrada. Con este bloque lo que estamos haciendo es pasar la señal que hemos transmitido del dominio del tiempo al dominio frecuencial, que es lo que nos interesa.

Los distintos campos los vamos a dejar con los valores indicados por Matlab, ya que no son muy relevantes.

Un dato importante a tener en cuenta con este bloque (y, por ende, también en el bloque IFFT) es que, a pesar de que la señal a su entrada se estructura en tramas, a la salida la señal pasa a estructurarse en muestras.

6.4.1.3.- Frame Conversion

Este bloque viene a solucionar el problema que acabamos de ver. Para poder operar con las distintas señales, como para aplicar retrasos sobre las mismas, éstas deben estar organizadas en tramas. Es por ello por lo que vamos a utilizar este bloque.

Para su uso correcto, tenemos que indicar el tipo de señal que queremos a su salida. En nuestro caso, tenemos que seleccionar **Frame based**.

6.4.1.4.- Eliminación de ceros

Llegados a este punto, tenemos que recordar que en la etapa de *modulación de datos*, para poder realizar correctamente el proceso de anti-Transformada Discreta de Fourier, tuvimos que añadir ceros de forma que el tamaño del vector fuese potencia de 2 (en nuestro caso, 8192 portadoras). También hay que tener en cuenta que colocamos

estos valores en las posiciones centrales de las diferentes tramas, de forma que tuviésemos una señal en banda base.

Lo que queremos realizar con este bloque es, precisamente, eliminar esta información "extra" que se añadió. Para ello empleamos un bloque "Selector", de forma que podamos seleccionar los datos que realmente nos interesan.

Así, en el campo **Index** tenemos que indicarle a Simulink qué datos de la entrada queremos que nos saque a la salida. Teniendo en cuenta que los ceros que se añadieron están en las posiciones centrales, indicamos: **[1:3408 4784:8192]**, es decir, de cada vector de entrada vamos a extraer las primeras 3408 muestras y las últimas 3407 muestras.

A la salida de esta etapa de *Demodulación OFDM* tendremos un vector de 6817 portadoras. Lo que buscaremos, a continuación, será eliminar las portadoras pilotos de cada símbolo OFDM.

6.4.2.- Eliminación de las portadoras Pilotos

En esta fase del receptor pretendemos, en primer lugar, eliminar de cada uno de los símbolos OFDM las distintas portadoras pilotos, que servían al receptor para facilitar la sincronización de trama, sincronización en frecuencia, sincronización en tiempo, estimación del canal y/o para la identificación del modo de transmisión. Otra función que vamos a realizar será el desentrelazado en frecuencia.

El esquema en el que nos vamos a apoyar es el siguiente:

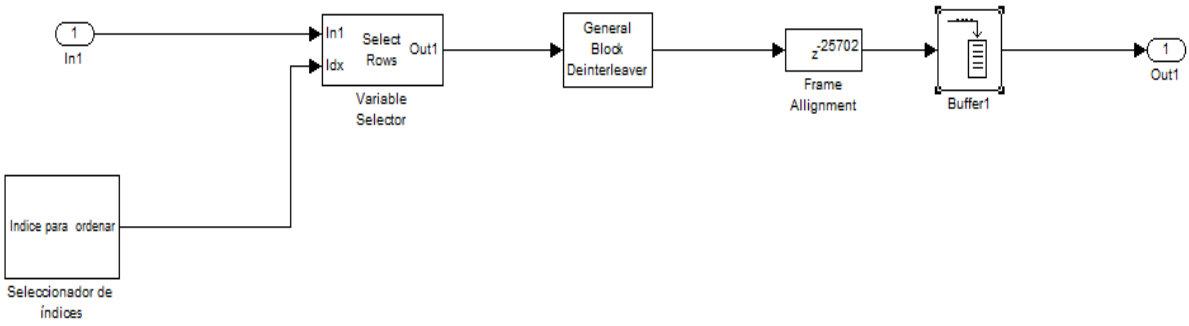

Fig. 6.19: Esquema de Eliminación de las Portadoras pilotos

Pasamos a explicar los distintos bloques que componen esta nueva fase del receptor.

6.4.2.1.- Índice para ordenar

Tal y como hicimos en el punto **6.3.3.5**, con este bloque pretendemos generar un vector para indicarle a Simulink como queremos que reordene la señal que entra en el bloque "Variable Selector".

El esquema de este bloque es el siguiente:

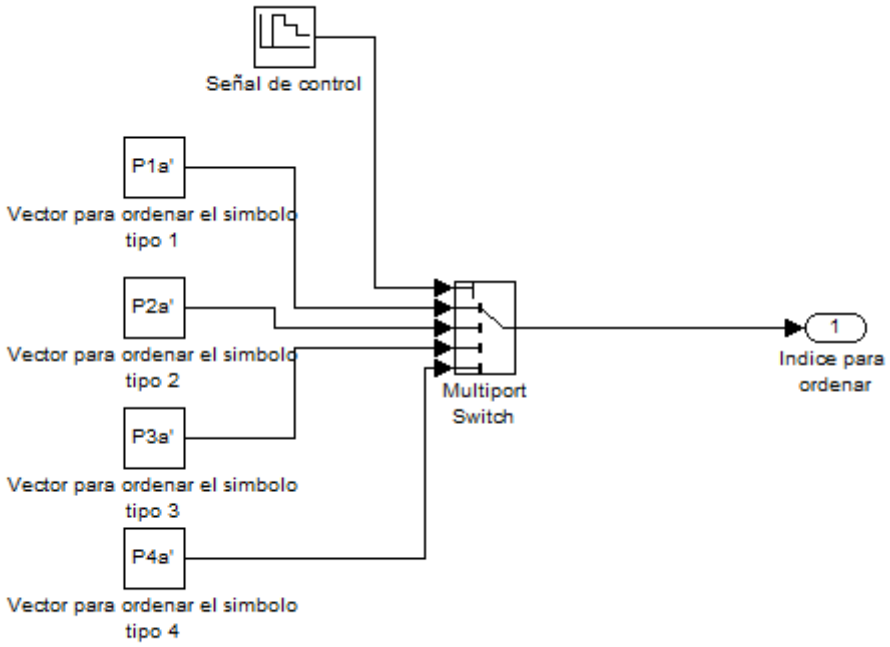

Fig. 6.20: Esquema para generar los vectores de reordenación

Pasamos a describir cada uno de los elementos que componen esta fase.

6.4.2.1.1.- Vector para ordenar el símbolo

Cada uno de estos cuatro bloques "Constant" va a indicar la posición de los datos dentro de cada trama. Los parámetros utilizados son:

- **Constant Value:** En este campo indicamos la posición de los datos transmitidos dentro de cada símbolo.

Los demás parámetros no son relevantes.

6.4.2.1.2.- Multiport Switch

Este bloque será el encargado de seleccionar en cada intervalo de tiempo de símbolo una de las secuencias transmitidas por los bloques "Constant". El "Multiport Switch" necesita para su funcionamiento una señal de control que le diga cuándo y cómo debe cambiar la entrada. Esta señal debe ser generada desde el exterior, como veremos en el siguiente punto.

En cuanto a los parámetros utilizados destacamos:

- **Number of inputs:** En este campo indicamos el número de señales a la entrada de este bloque. En nuestro caso será de **4**.
- **Used zero-based indexing:** Con este campo indicamos si permitimos o no el instante cero dentro de la señal de control habilitando esta opción. Nosotros la hemos marcado.
- **Sample Time:** Por defecto, la dejamos a -1.

6.4.2.1.3.- Señal de Control

Con este bloque generamos una señal periódica e indicamos al bloque "Multiport Switch" cuándo debe conmutar a una entrada. Lo que realmente estamos haciendo con esta función es generar los distintos tipos de símbolos OFDM que hay que una misma trama T2.

El bloque se ha configurado de la siguiente manera:

- **Vector of outputs value:** En este campo vamos a generar la señal que queremos que se repita en el tiempo. Tenemos que repetir la misma secuencia que la que se usó en el transmisor, esto es **[0 3 2 1]**.
- **Sample Time:** En este campo irá el tiempo en el que permanecerá constante la señal. Nos interesa que la entrada se mantenga constante cada 924 μs, es decir, cada **Ts**.

6.4.2.2.- Variable Selector

Con este bloque queremos seleccionar y reordenar las filas de la señal de entrada de acuerdo a lo especificado por el puerto ld_x. En nuestro caso le vamos a indicar las posiciones de los datos para los distintos símbolos OFDM.

Los parámetros que hemos utilizado son los siguientes:

- **Number of input signal:** Tenemos un vector a la entrada de este bloque, luego ponemos **1**.
- **Select:** Tenemos un vector columna, luego nos interesa reordenar las filas. Seleccionamos **Rows**.
- **Select mode:** Determina si el bloque usa los mismos índices para todas las entradas o si emplea diferentes índices, en función de la entrada. Hemos seleccionado la opción **Variable**.
- **Index mode:** Indicamos si queremos que la primera muestra se contabilice como índice 0 o índice 1. Marcamos la opción **One-based**.

6.4.2.3.- Desentrelazado en Frecuencia

El objetivo del desentrelazado en frecuencia es, precisamente, invertir el entrelazado que se realizó entre los datos de un símbolo OFDM, como vimos en el punto **6.3.3.2**.

Para llevar a cabo esta función de manera adecuada, haremos uso del bloque "General Block Deinterleaver".

Es muy importante que le pasemos la misma información que al bloque que realiza el entrelazado (en este caso, a ambos bloques les pasamos el vector des/entrelazador). En caso contrario, los datos que se obtendrían tras la recepción serían diferentes a los transmitidos.

6.4.2.4.- Frame Allignement

Este es uno de los bloques más importantes de toda la etapa de recepción. En primer lugar vamos a explicar qué es lo que hace para posteriormente hablar de la finalidad del bloque.

Este bloque lo que hace es retrasar la señal que tiene a su entrada un número de muestras determinado, que será especificado por el usuario.

Los parámetros con los que se ha configurado dicho bloque son:

- **Delay units:** En este campo especificamos si queremos retrasar la señal en muestras (samples) o en tramas (Frames). Nosotros hemos seleccionado la opción **Samples**.
- **Delay (samples):** Tenemos que indicar el número de muestras en los que se retrasará la señal.
- **Initial Conditions:** Especificamos lo que queremos que se transmita mientras se lleva a cabo el retraso de la señal de entrada.

El resto de parámetros no son relevantes.

El motivo por el que añadimos este bloque es el siguiente: el buffer que incluimos en la etapa de *Adición de Portadoras Pilotos* (ver punto **6.3.3**) lo que hacía era almacenar la trama FEC procedente de la etapa de *Codificación de Datos*. Pues bien, mientras en el buffer se almacenan todos los datos, éste va transmitiendo aquellos que tenía pre-cargado (en nuestro caso, dijimos que los valores iniciales del buffer valían cero), con lo que se genera un vector de ceros del tamaño de salida del buffer, esto es, de tamaño 6698. Esto implica que existe una diferencia entre el conjunto de datos ya codificados en el transmisor y el conjunto de datos que se obtienen tras realizar el proceso de eliminación de las portadoras pilotos de 6698 muestras. Esto es muy importante tenerlo en cuenta para el bloque siguiente, un buffer que almacena las muestras de los distintos símbolos OFDM en una trama FEC de 32400 elementos. Al tener un vector de 6698 muestras que no teníamos en el transmisor, las tramas FEC en recepción no van a coincidir con las tramas FEC de transmisión, o dicho de otro modo, los datos que se van a recibir no van a ser los mismos que fueron transmitidos.

Así, al retrasar la señal una cantidad de muestras igual a **25702**, o lo que es lo mismo, añadiendo 25702 ceros al inicio del primer vector que llega, tendremos una primera trama FEC de 32400 ceros, pero el resto de tramas sí que van a coincidir con las que se transmitieron. Somos conscientes de que estamos añadiendo un retraso a la señal, pero en contrapartida, nos estamos asegurando que la señal recibida va a coincidir con la transmitida.

6.4.2.5.- Buffer

Este buffer se va a encargar de almacenar los datos de los símbolos que le van llegando, 6698, y sacar tramas de 32400 elementos. Ya a partir de este bloque vamos a dejar de trabajar a nivel de símbolos para volver a trabajar con tramas FEC (FECFRAME).

Los parámetros utilizados son:

- **Output Buffer size:** Ya hemos dicho que este valor será **32400**.
- **Buffer Overlap:** Queremos que la información salga por vectores pero que ésta no se repita. Por tanto, el valor de este campo será **cero**.
- **Initial Conditions:** se pondrá a cero, ya que no queremos que se transmita nada mientras el Buffer se llena.

Haciendo un breve resumen de lo visto hasta ahora en el receptor, hemos realizado la demodulación de la señal recibida y nos hemos quedado sólo con los datos, es decir, hemos eliminado las distintas portadoras pilotos. A partir de este momento dejamos de hablar de símbolos OFDM ya que vamos a trabajar a nivel de trama FEC.

En la siguiente etapa del receptor, se procede a la decodificación de la trama FEC.

6.4.3.- Decodificación y Demodulación de Datos

Una vez hemos llevado a cabo todo el proceso de eliminación de portadoras pilotos y nos encontramos trabajando con tramas FEC en lugar de símbolos OFDM, tenemos que proceder a la decodificación y demodulación de estas tramas.

En primer lugar, las tramas pasan por una fase de desentrelazado. Recordamos que el entrelazado se llevó a cabo durante la etapa de *Codificación y Modulación de Datos* con el fin de dispersar posibles errores que dañaran la trama.

Una vez realizado este desentrelazado, se procederá a la demodulación de la señal para acabar decodificando la trama recibida.

El esquema que se ha venido siguiendo durante toda esta etapa es el siguiente:

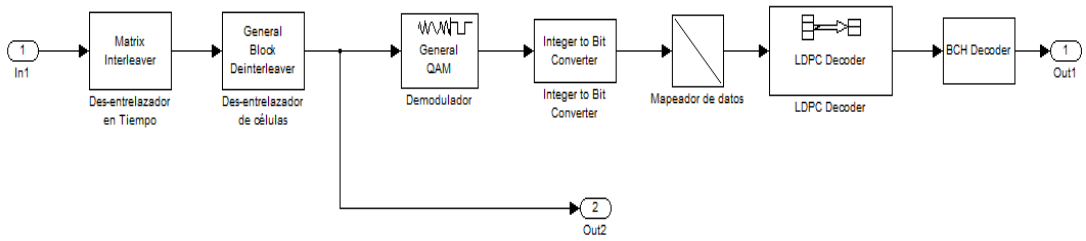

Fig. 6.21: Esquema del proceso de Decodificación de datos

Aunque la gran mayoría de los bloques ya se han visto, vamos a comentar la función de los mismos y los parámetros más importantes con los que los hemos configurado.

6.4.3.1.- Desentrelazado en Tiempo

Si recordamos el funcionamiento del entrelazador en tiempo (ver punto **6.3.2.5**), lo que este bloque hacía era colocar los datos que recibía en una matriz por columnas y leerlos por filas.

Pues bien, parece claro que en esta ocasión debemos actuar de forma complementaria. Para ello, vamos a utilizar el bloque "Matriz Interleaver". Este bloque rellena una matriz por filas y envía el contenido de dicha matriz leído por columnas.

Los valores para estos campos son los mismos que para el bloque que usamos con anterioridad. Esto es:

- **Number of rows:** 6840.
- **Number of columns:** 5.

Otra solución por la que podríamos haber optado era la de haber usado el mismo bloque que en la etapa de *Codificación*, pero invirtiendo los parámetros; es decir, en **Number of rows** haber puesto 5 y en **Number of columns** 6840. El resultado no hubiese variado.

6.4.3.2.- Desentrelazador de células

Nos encontramos con un caso muy parecido que al del punto anterior. Necesitamos contrarrestar el entrelazado que realizamos durante la fase de *Codificación.*

Para llevar a cabo la función de aleatorizar las células que componen una trama hicimos uso del bloque "General Block Interleaver". Lo que nosotros estamos buscando ahora es realizar el proceso opuesto: queremos "devolver" cada célula a su posición inicial. Por ello en esta ocasión emplearemos el bloque "General Block Deinterleaver". Como elementos, tendremos que pasarle el mismo vector que para realizar el entrelazado de la trama. Como bien se indicó con anterioridad, le pasamos el nombre del vector que tenemos implementado en nuestro fichero de variables y que se adjuntará en el Anexo.

6.4.3.3.- Demodulador

El objetivo de toda demodulación es recuperar la señal transmitida originariamente.

Para conseguir ésto, hemos utilizado el bloque "General QAM Demodulator BaseBand", ya que nuestra señal modulada era una 4-QAM.

Los parámetros con los que se ha configurado dicho bloque han sido:

- **Signal Constellation:** Le damos a este campo las posiciones para una constelación QPSK, ya normalizada, de acuerdo al mapeado indicado en el estándar.
- **Output type:** Indicamos el tipo de los elementos de salida. En este caso van a tratarse de enteros.
- **Output data type:** Como el bloque "LDPC Decoder" necesitan que los datos a su entada sean del tipo **double**, seleccionamos esta opción.

6.4.3.4.- Integer to Bit converter

Como su nombre hace intuir, este bloque se va a encargar de realizar la conversión de enteros a bits.

Una vez demodulada la señal, hemos obtenido un vector de números enteros, representando cada uno de ellos a un símbolo (al ser una señal 4-QAM, estos símbolos van del 0 al 3). Con este conversor, pasamos a trabajar a nivel de bits.

Los parámetros con los que se ha configurado dicho bloque han sido:

- **Number of bits per Integer:** ya hemos comentado que se trata de una señal 4-QAM, luego el número de bits por símbolo es **2**.
- **Output data type:** Al igual que en el bloque anterior, seleccionamos la opción **double**.

6.4.3.5.- Mapeador de Datos

Su objetivo no es otro que el de mapear los distintos bits de entrada para "crear" una nueva salida.

Los valores que le hemos dado a dicho bloque son los siguientes:

- **Vector of input values:** Como tenemos un vector binario, los datos sólo pueden tomar dos valores: ó 0 ó 1. Así nuestro vector será **[0 1]**.

- **Table data:** En ella escribimos la conversión que queremos de los datos. En nuestro caso nos interesa **[1 0]**, es decir transformamos los ceros en unos y viceversa.
- **Lookup method:** Describe el método para realizar el mapeado. Nosotros hemos utilizado **Interpolation-Extrapolation**.
- **Sample time: -1**, por defecto.

Necesitamos este bloque para realizar la decodificación de la señal de manera correcta. Sin él, nuestro decodificador no funcionaría correctamente y a la salida no obtendríamos los datos esperados. De hecho, los datos que obtendríamos presentan la misma secuencia que antes de realizar la *codificación LDPC* de los mismos pero con los valores invertidos, es decir, donde ahora obtenemos un uno antes de la codificación había un cero y viceversa.

6.4.3.6.- Decodificación LDPC

Su función es la de eliminar los bits de comprobación de paridad, que se encuentran al final de la trama y que permitían al sistema conocer si se habían producido errores durante la transmisión.

Para este bloque, y al igual que para la *Codificación LDPC* (ver punto **6.3.2.1.2**) nos hemos apoyado en una serie de funciones que ya venían implementadas en Matlab (versión R2007b). De esta forma, sólo hemos tenido que indicar el nombre de la función, pasándole la tasa de codificación LDPC que hemos empleado durante la simulación (en nuestro caso 1/2).

Los parámetros para este bloque son los siguientes:

- **Parity check matrix:** Aquí nos apoyamos en la función que ya se encuentra implementada en Matlab. Así escribimos: **dvbt2ldpc (1/2)**.
- **Output format:** Sólo queremos obtener la información de la trama codificada, luego marcamos **Information part**.
- **Decision type:** Marcamos la opción **Hard decisión**.
- **Output data type:** Vamos a trabajar a partir de ahora con números binarios, por lo que marcamos **boolean**.

6.4.3.7.- BCH Decoder

La función de este bloque también es la de eliminar parte de la información que se generó durante la *Codificación BCH* para hacer que la señal que se estaba transmitiendo fuera más robusta.

Para llevar a cabo esta función hemos utilizado un bloque que utiliza una función que ya venía implementada en Matlab. Se trata de una *S-function* y se denomina *scomberlekamp.* Nosotros sólo hemos tenido que pasarle los valores adecuados, según lo especificado por dicha función.

Con esto ya tenemos finalizada la fase de *Decodificación y Demodulación de datos*. Recordamos que a partir de este momento dejamos de trabajar con FECFRAMEs para trabajar a nivel de BBFRAMEs.

6.4.4.- Extracción de datos

En estos momentos, nuestra BBFRAME está perfectamente preparada para que podamos acceder a la información transmitida durante la etapa de *Generación de datos,* en el transmisor (ver punto **6.3.1**).

Para poder recuperar esta información, la fase de extracción de datos va a contar de los siguientes elementos:

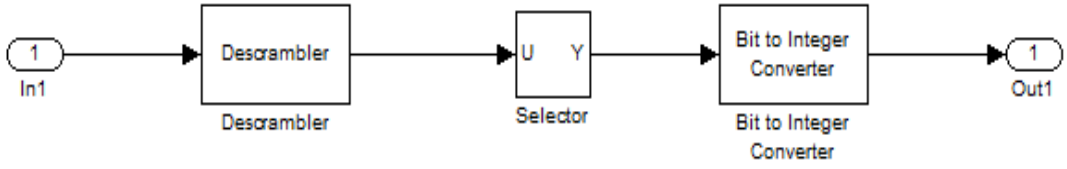

Fig 6.22: Esquema de la etapa Extracción de datos

6.4.4.1.- Descrambler

Utilizamos este bloque para invertir el proceso de aleatorización llevado a cabo en la etapa de *Generación de datos* sobre la BBFRAME.

Los parámetros que debemos utilizar para este bloque tienen que ser los mismos que los que empleamos para la aleatorización. Son los siguientes:

- **Calculation base:** Como estamos trabajando con señales binarias, este campo valdrá **2**.
- **Scramble polynomial:** En este parámetro introducimos el polinomio de des-aleatorización que viene indicado en el estándar. Así tendremos el vector 1100000000000001.
- **Initial states:** Es la condición inicial para los registros internos. Volvemos a poner las mismas condiciones iniciales que marcamos para el Scrambler: 100101010000000.

6.4.4.2.- Selector

Una vez hemos procedido a la aleatorización de la BBFRAME, hemos de recordar que durante la transmisión de los datos, procedimos a la inserción tanto de una cabecera en banda base, que describía el formato de los datos que se transmitían, como de información "extra" (que se añadieron en forma de ceros) para que la señal tuviera las dimensiones indicadas por el estándar.

Esta información deberá ser eliminada de nuestra señal. Ése es el objetivo de este nuevo bloque: eliminar todo aquello que no sea información útil.

Convenimos en su momento que la información "extra" se iba a añadir al final de la señal, mientras que la cabecera, obviamente, deberá ir al principio. Esto es muy importante tenerlo en cuenta a la hora de seleccionar el rango de la señal con la que nos interesa quedarnos.

Como la configuración de este bloque ya se ha repetido en numerosas ocasiones a lo largo de este Proyecto, sólo nos vamos a fijar en el campo **Index**. Necesitamos eliminar las primeras 20 muestras de la señal, ya que éstas pertenecían a la cabecera de la misma; por otra parte, necesitamos que a la salida haya 31584 (si hacemos memoria, es el resultado que se obtenía cuando pasábamos a bits los 3948 datos que

transmitíamos). Con todos estas consideraciones, extraemos la señal que se encuentra en el rango **[**81 31664**]**.

6.4.4.3.- Bit to Integer Converter

El fin de este bloque es pasar de la secuencia de bits que tenemos a una secuencia de bytes.

Los parámetros de este bloque van a ser, por tanto**:**

- **Number of bits per Integer:** Como ya hemos dicho, queremos pasar de bits a bytes, luego este valor es **8**.
- **Output data type:** Como tenemos enteros, nosotros hemos seleccionado la opción **uint8** (enteros sin signos), pero no es un parámetro muy relevante.

Con este bloque finalizamos la etapa de recepción. Si todo ha ido de forma correcta, la señal que tenemos que recibir en este punto, aunque retrasada en el tiempo, tiene que ser igual a la señal que transmitimos. Estos retrasos, como ya vimos en su momento, vienen ocasionados por los buffers, que envían los datos que éstos tuvieran almacenados en el momento en que se inicia la transmisión. Esto no imposibilita la recepción óptima de los datos; no obstante sí es muy importante conocer el retraso introducido por el sistema para poder indicar a nuestro equipo cuándo deben empezar a leer los datos.

A continuación, mostraremos los resultados que hemos obtenido durante la etapa de Simulación.

6.4.5.- Resultados

Durante el desarrollo de esta memoria hemos estado viendo tanto el desarrollo teórico como la implementación, mediante Simulink, de un sistema de comunicaciones basado en el nuevo estándar para la televisión digital, DVB-T2.

En este apartado lo que queremos es presentar los resultados que hemos ido obteniendo durante el desarrollo práctico del sistema.

Para obtener las imágenes que más nos interesa mostrar nos hemos apoyado en un par de bloques, que pasamos a enunciar:

- Por un lado, para poder representar las constelaciones recibidas y transmitidas, utilizamos el bloque "Discrete – Time Scatter Plot Scope".
- Para representar el espectro en frecuencia de la señal transmitida, usamos el bloque de Simulink "Spectrum Scope".

Otro bloque bastante interesante para nosotros es "To simout". Este bloque tiene la característica de almacenar en una variable los datos de la señal que tiene a su entrada, con la que después podremos trabajar.

Para verificar que nuestra simulación ha sido correcta, presentaremos los resultados de tres formas distintas.

6.4.5.1.- Constelaciones y Espectro

La primera forma que se nos ocurre de poder verificar que nuestra simulación ha sido correcta, es a través de las constelaciones de las señales transmitidas y recibidas, así como a través del espectro de la señal.

Venimos comentando a lo largo de la memoria que la señal que nosotros hemos modulado es una señal QSPK. Luego la constelación de las diferentes señales que recibimos deben tener la siguiente forma:

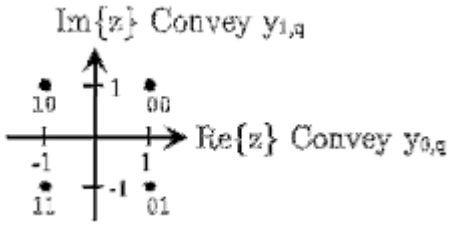

Fig. 6.23: Esquema de una señal QPSK

Las constelaciones que nosotros hemos obtenido, en régimen permanente de la señal, han sido:

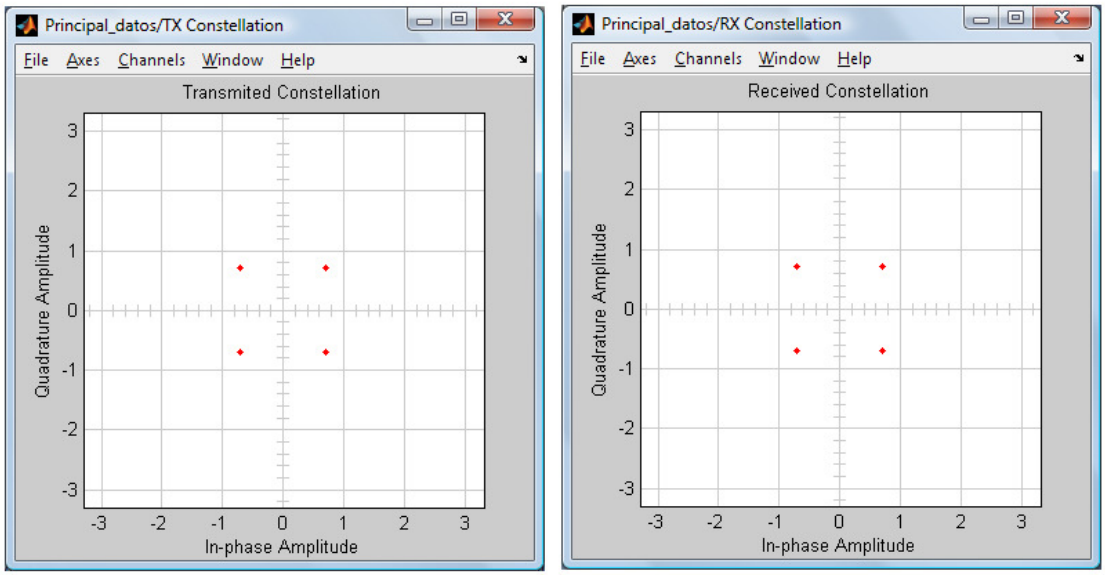

Fig. 6.24: Constelaciones de las señales transmitidas y recibidas

Como puede verse claramente, ambas constelaciones son idénticas a la constelación de una señal QPSK ideal.

Otro elemento que nos indica si la simulación se ha realizado correctamente o no es el espectro de frecuencia de la señal transmitida. El estándar define el siguiente espectro de frecuencia para señales de intervalo de guarda igual a 1/8.

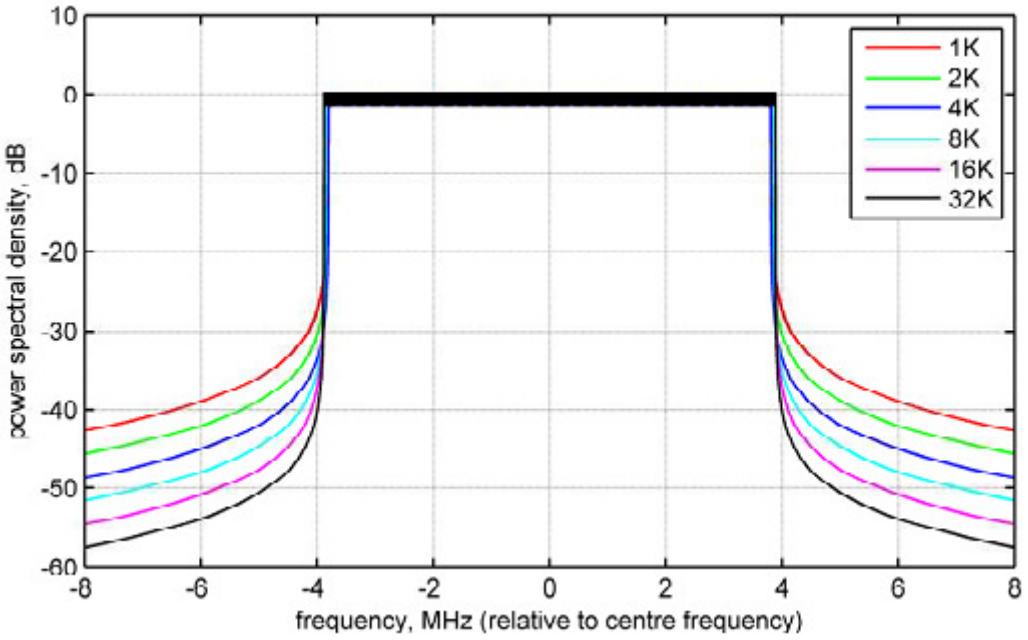

Fig. 6.25: Espectro teórico para una señal DVB-T2

Nosotros, durante el proceso de simulación, hemos obtenido el siguiente espectro:

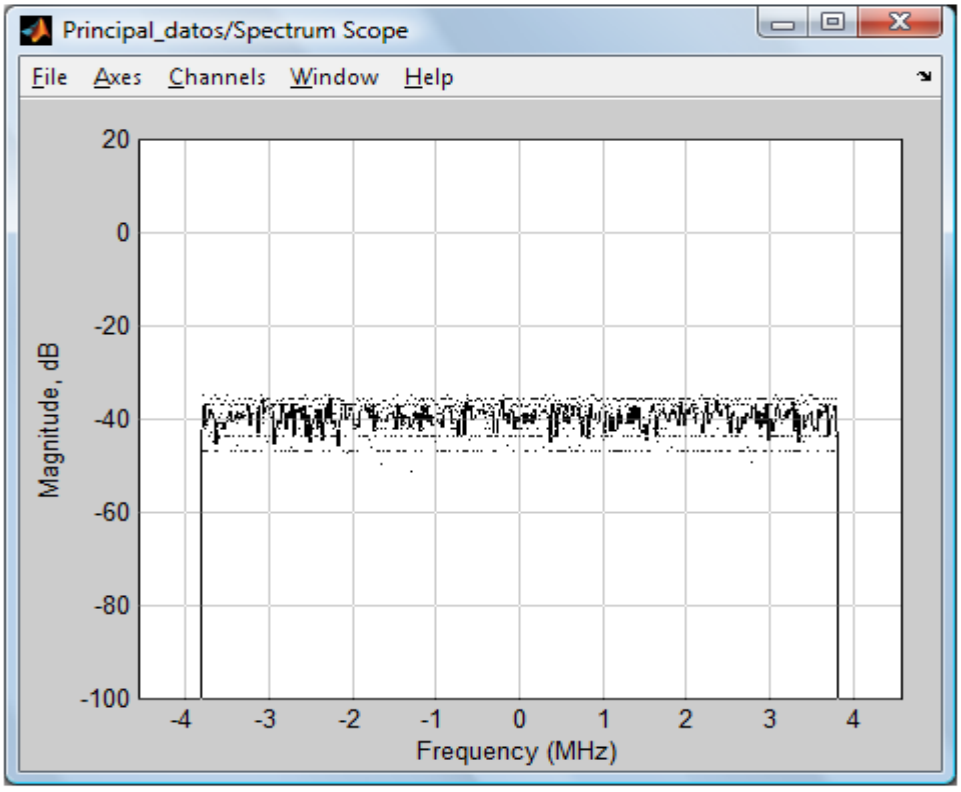

Fig. 6.26: Espectro en frecuencia de la señal transmitida

Comparando ambas figuras, vemos que ambos espectros son casi idénticos.

6.4.5.2.- Variables

Otra forma que tenemos para poder comprobar que los datos recibidos son idénticos a los transmitidos es a través de las variables generadas por los bloques "To simout".

Para que la comparación entre ambas señales sea lo más real posible, lo que hemos hecho ha sido retrasar la señal de entrada al *simout* del transmisor el mismo número de muestras que sabemos que lo está la señal recibida.

Una forma de comparar ambas señales es viendo el conjunto de datos que contienen, a través del *Workspace* de Matlab. En él podemos encontrar toda la información referente a las variables que está utilizando el programa en ese momento. No obstante la tarea de comparar uno a uno todos los datos de ambas variables resulta una tarea demasiado tediosa.

Para realizar esta tarea de una manera mucho más rápida lo que vamos a hacer es trabajar con las variables que se generan. Así, si restamos ambas variables, si todo ha ido correcto, deberíamos obtener un vector de ceros. Para comprobar que efectivamente obtenemos un vector de ceros calculamos el máximo y el mínimo de ese vector.

Los resultados que hemos obtenido, como era de esperar, han sido:

max (recibida1 –transmitida1) = 0

y

min (recibida1 –transmitida1) = 0

siendo *recibida1* y *transmitida1* las variables generadas por los bloques "To simout" del receptor y del transmisor, respectivamente.

6.4.5.3.- BER

La última forma de demostrar que nuestra simulación se ha realizado de manera correcta es a través de la tasa de bits erróneos, es decir, de la BER.

Para calcular la BER de nuestro sistema nos ayudaremos de los siguientes bloques:

- Bloque "Error Rate Calculation". Como es de esperar, este bloque se encarga de calcular la tasa de error del sistema comparando los datos recibidos con los datos de la señal transmitida retrasada. La señal de salida estará constituida por tres elementos: la tasa de error seguida del número de datos erróneos detectados y el número total de datos comparados. El campo más importante que tendremos que configurar en este bloque es **Receive delay**, es decir, el retraso de la señal. Nosotros, como le hemos añadido a la señal transmitida un retraso antes de entrar en este bloque, ponemos en este campo **0**.

- Una vez calculada la tasa de error de nuestro sistema, para poder visualizarla, utilizaremos el bloque "Display", que nos permitirá poder ver simultáneamente los diferentes valores, esto es, tasa de error, número de datos erróneos y número total de datos comparados.

A continuación mostraremos los valores que hemos obtenido para diferentes intervalos de la simulación:

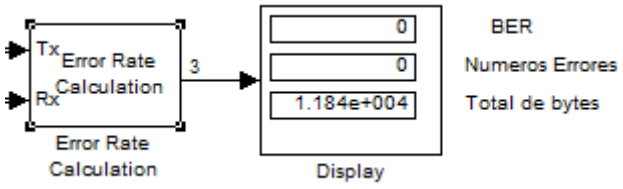

Fig. 6.27: BER a los 4.48·10^(-3) seg.

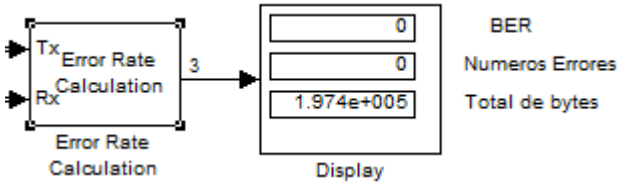

Fig. 6.28: BER a los 0.2 seg

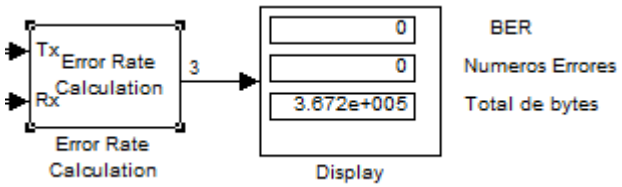

Fig. 6.29: BER a los 0.4 seg.

Observando las distintas figuras, vemos que en todas ellas el número de datos erróneos que se detecta, y por tanto también valor de la BER, es cero.

Luego, con las distintas demostraciones que hemos realizado, todas con el mismo resultado, concluimos que nuestro sistema de comunicaciones funciona de acuerdo a lo especificado en el estándar.