SOFTWARE DE SIMULACIÓN DEL FUNCIONAMIENTO DE UN MODEM SEGÚN LA RECOMENDACIÓN V32BIS DE LA ITU-T

T.S.R. Aholab

D.E.T.

Escuela Superior de Ingenieros de Bilbao

Inma Hernaez <inma@bips.bi.ehu.es>
Txema Rua <txema@bips.bi.ehu.es>
Iker Veiga <iker@bips.bi.ehu.es>

1 ÍNDICE

2 INTRODUCCIÓN

Mediante la presente aplicación se pretende simular el funcionamiento de un módem, que en lo que se refiere a la parte de procesado de señal, va a cumplir con las características definidas en la recomendación V.32bis de la Unión Internacional de Telecomunicaciones (ITU). Asimismo, se ofrecen diversas facilidades de visualización de tal manera que el usuario tendrá la capacidad de monitorizar las señales en diferentes puntos del sistema e incluso pueda introducir las distorsiones típicas de un canal telefónico para comprobar como el módem responde frente a ellas.

El objetivo que se persigue con todo esto es facilitar en lo máximo posible el aprendizaje del funcionamiento de este tipo de sistemas habiendo escogido para ello un módem lo suficientemente simple como para que en el estudio del funcionamiento no sean necesarios conocimientos avanzados de procesado digital de señal pero que al mismo tiempo sirva como base para el estudio posterior de sistemas de comunicaciones con una complejidad mayor.

El sistema se ha diseñado empleando un formato de ventana apto para Windows-9x, no necesita de ninguna aplicación para su instalación y presenta una interfaz simple e intuitiva.

3 CARACTERÍSTICAS GENERALES DEL MODEM

El módem está destinado a su utilización en las conexiones con la red telefónica general conmutada (RTGC) y en circuitos arrendados de tipo telefónico a dos hilos punto a punto. Las principales características del módem son las siguientes:

- Modo de funcionamiento dúplex en la RTGC y en los circuitos arrendados a dos hilos punto a punto.
- Separación de canales de ida y vuelta mediante técnicas de cancelación de eco.
- Modulación de amplitud en cuadratura para cada canal con transmisión síncrona en línea a 2400 símbolos/s.
- La frecuencia de portadora ha de ser 1800 Hz.
- Pueden realizarse en el módem cualquiera de las siguientes velocidades de transmisión de datos.
    - 14 400 bit/s, con codificación en rejilla (Trellis).
    - 12 000 bit/s, con codificación en rejilla.
    - 9600 bit/s, con codificación en rejilla.
    - 7200 bit/s, con codificación en rejilla.
    - 4 800 bit/s, sin codificar.
- El codificador trellis será de tipo 2/3 con ocho estados.
- Aleatorización de datos mediante el empleo de scrambler y descrambler.

4 DIAGRAMA FUNCIONAL. Explicación de los módulos

En la figura 2.1 se muestra el diagrama funcional del módem v.32 bis software. Se trata de un diagrama completo en el que se muestran todos los bloques que constituyen el modulador y el demodulador.

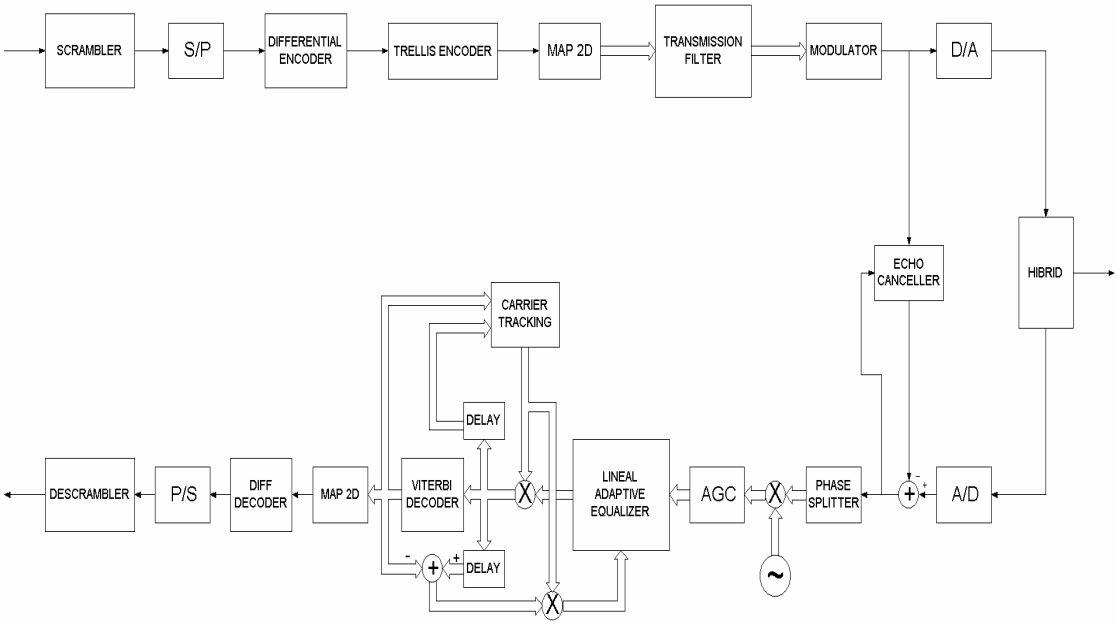


fig 2.1. Diagrama funcional del modem v32bis software

Los bloques de la parte superior (de la bobina híbrida hacia arriba) forman parte del emisor. Son los siguientes (siguiendo el recorrido de la señal):

4.1 Transmisor

4.1.1 Scrambler (aleatorizador)

Se realiza mediante la clase Cscramb que está definida en scramb.h e implementada en scramb.cpp.

En este diseño se ha empleado un conjunto *scrambler-descrambler* autosincronizable es decir, no precisa de ninguna secuencia inicial. Este tipo de *scramblers* se caracterizan por un polinomio que define completamente su funcionamiento. En este caso, se utiliza el polinomio generador especificado en la norma V.32bis para el modo llamada (PGL):

$1 + x^{-18} + x^{-23}$

Se puede realizar el scrambleado de dos formas: una rápida pero que no es intuitiva, cuando se define la constante SCRAMB_FAST en el fichero scramb.h y otra menos eficiente pero más intuitiva cuando no se define la constante. La segunda manera es la implementación directa de la ecuación:

$$y(n) = x(n) + \sum h_k \cdot y(n-k)$$

Donde h(k) son los coeficientes correspondientes al polinomio descrito anteriormente (son 0 o 1) y como son operaciones con bits, las sumas equivalen a la operación XOR.

4.1.2 Conversor serie-paralelo (S/P)

Se realiza mediante la clase CSer_Par que está definida en serpal.h e implementada en scramb.cpp.

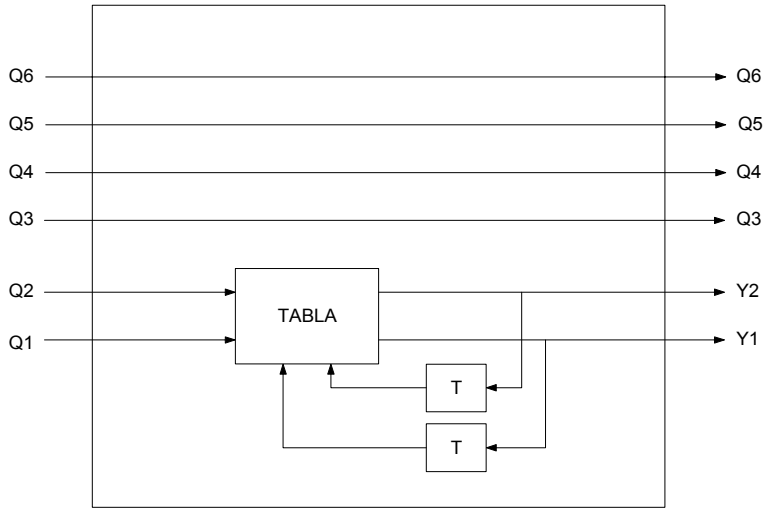
La único que hace es coger los bits que llegan (bits serie) e ir formando palabras con ellos, que serán de diferente longitud en función del Bit Rate seleccionado:

- 4800 bit/s ⇒ longitud 2 bits
- 7200 bit/s ⇒ longitud 3 bits
- 9600 bit/s ⇒ longitud 4 bits
- 12000 bit/s ⇒ longitud 5 bits
- 14400 bit/s ⇒ longitud 6 bits

4.1.3 Codificador diferencial

Se realiza mediante la clase CdiffEncod que está definida en el fichero diffencod.h e implementada en diffencod.cpp.

La codificación diferencial se realiza según el siguiente esquema. Los bits Q3, Q4, Q5 y Q6 solo aparecerán en las velocidades que lo requieran.

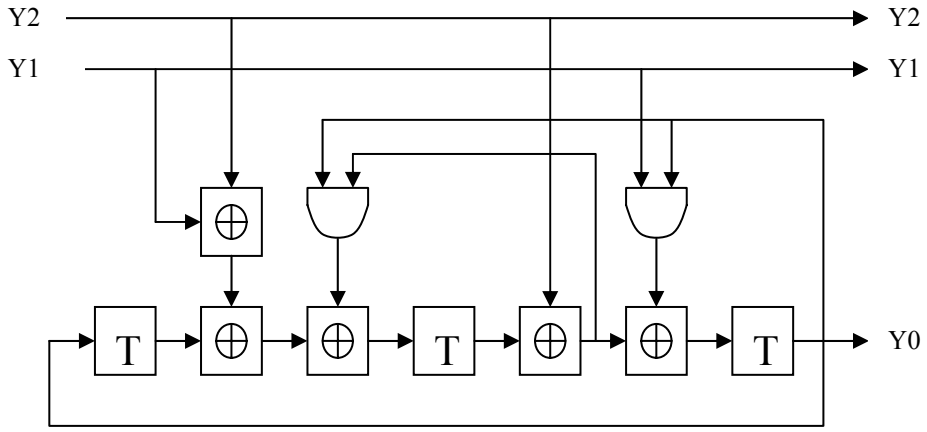


Las tablas vienen definidas en la norma y son diferentes en función de si posteriormente se va a realizar una codificación convolucional o no.

4.1.4 Codificador convolucional (codificador Trellis)

Se realiza mediante la clase CtrellisEncod definida en convencod.h e implementada en convencod.cpp.

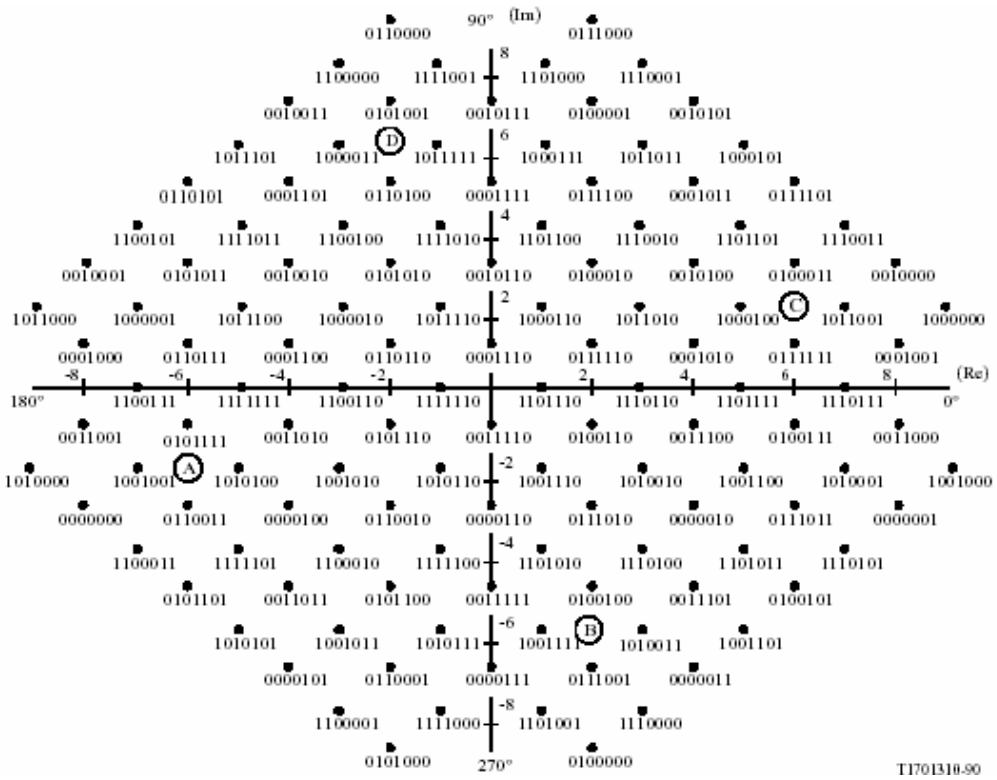
Este módulo sólo entra en funcionamiento a velocidades superiores a 4800 bps y consiste en calcular un bit redundante Y0 a partir de los dos bits Y1 e Y2 que se obtienen después del codificador diferencial. Se trata por lo tanto de un codificador convolucional de velocidad 2/3. El codificador viene definido en la norma y tiene el siguiente esquema:



Para que el proceso sea más eficiente se ha implementado mediante una tabla denominada Mod_Trellis_Map, a la cual se accede con los bits Y1 y Y2 de entrada y el estado anterior, para obtener a la salida los bits Y0, Y1 e Y2 y los 3 bits del estado siguiente.

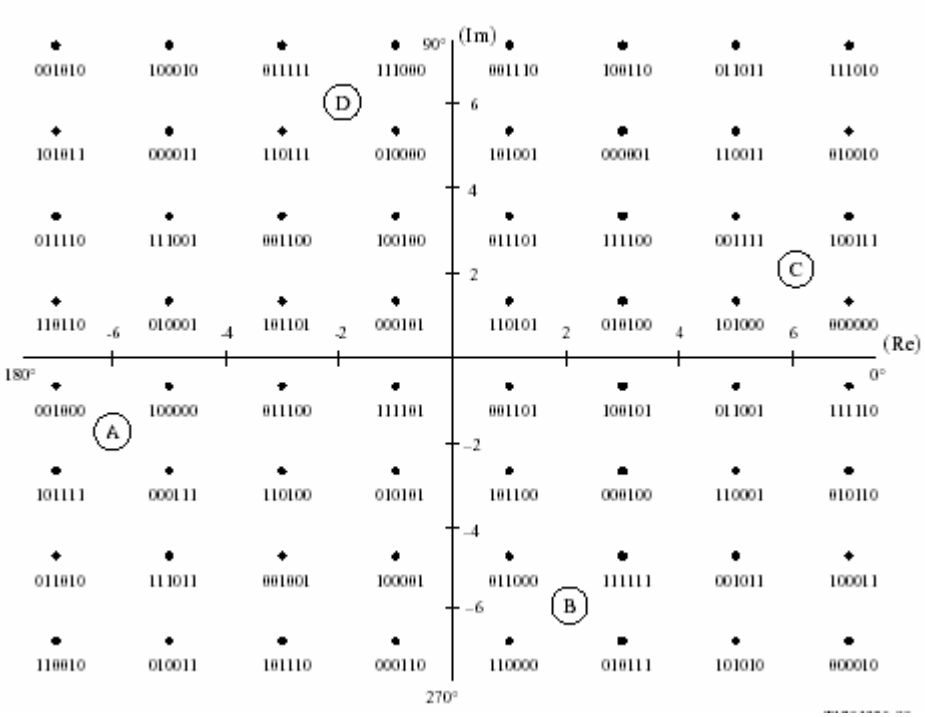
4.1.5 Mapeo 2D de los grupos de bits

Este módulo realiza la conversión de bloque de bits a símbolo de la constelación. Las constelaciones están descritas en la norma V.32bis y son las siguientes:

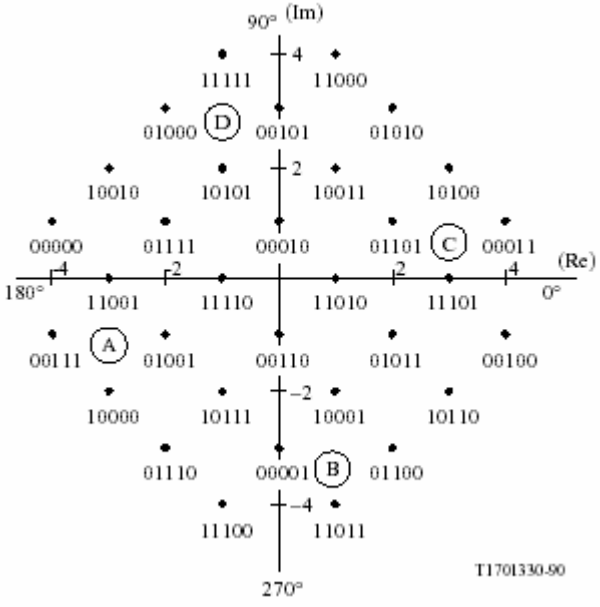


Constelación para la modulación a 14400bps

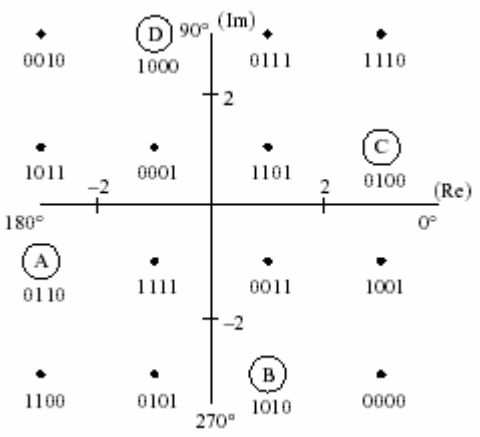
Los números binarios se refieren a Y0 , Y1 , Y2 , Q3 , Q4 , Q5 , Q6 para la constelación de 14.4Kbps. Para el resto de las constelaciones irán desapareciendo los bits menos significativos que no se empean. Por otra parte, A, B, C y D se refieren a los símbolos que se usan en la sincronización de los dos módems, es decir, en los procedimientos de arranque y reacondicionamiento definidos en la norma.



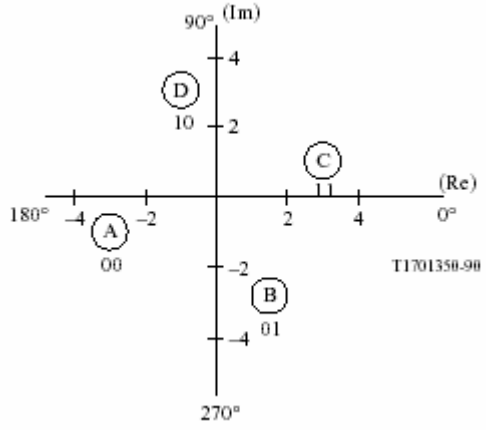
Constelación para la modulación a 12000bps



Constelación para la modulación a 9600bps



Constelación para la modulación a 7200bps



Constelación para la modulación a 4800bps

Este módulo se implementa mediante la clase CConstel definida en constel.h e implementada en constel.cpp. En la clase se guardan como datos miembro todas las constelaciones y la búsqueda del símbolo será simplemente un acceso al objeto de tipo CConstel para lo cual se sobrecarga el operador []. Las constelaciones se inicializan en el fichero constel.cpp.

4.1.6 Filtro de transmisión

Se realiza mediante la clase CtxFilt definida en txfilt.h e implementada en txfilt.cpp.

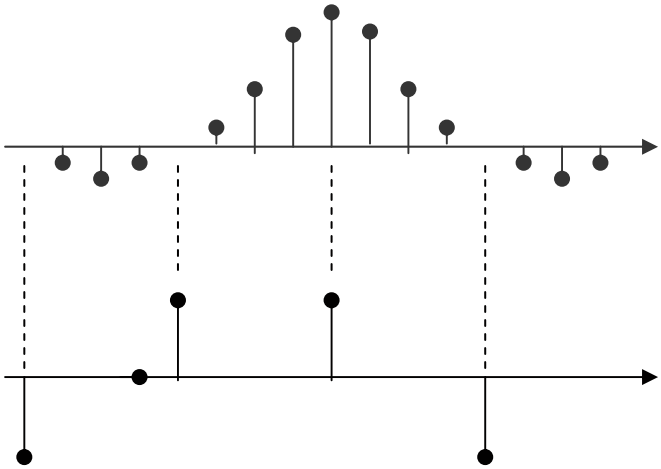
En realidad vamos a disponer de dos filtros que hacen lo mismo: uno para la rama en fase y otro para la rama en cuadratura. Cada una de estas ramas a su vez realiza dos funciones:

- **Interpolación:** Se coge el tren de símbolos (deltas de peso el valor del símbolo) y se interpola metiendo ceros entre las deltas.
- **Filtrado:** Se filtra este tren de deltas interpolado con el filtro de coseno alzado para limitar en banda la señal que se va a transmitir.

La razón de realizar primero una interpolación para posteriormente filtrar es que la frecuencia de muestreo a la que se va a funcionar es de 9600 muestras por segundo (para garantizar el ancho de banda del canal telefónico que es de unos 3KHz, y además ser

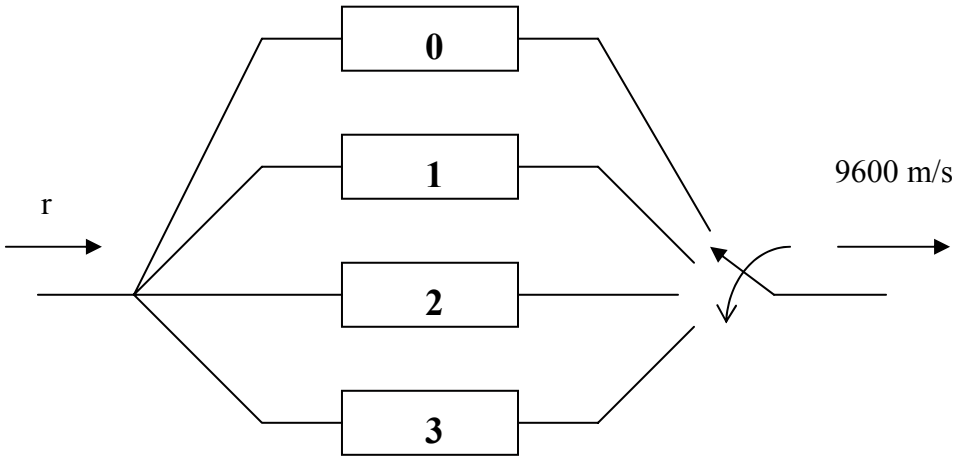
múltiplo de la velocidad de transmisión) y la velocidad en la línea es de 2400 símbolos por segundo. Esto supone necesitar a la salida del filtro 4 muestras por símbolo. Por esta razón, primero se interpola por 4, es decir, se introduce una delta con el peso del símbolo correspondiente y 3 ceros, después se filtra con un filtro con la forma del coseno alzado y se manda al Codec. Con esto conseguimos un flujo de datos de salida de 2400 símbolos por segundo con una frecuencia de muestreo de 9600 Hz.

Para realizar todo esto de una manera eficiente, se emplean filtros polifase. Lo que se hace es dividir el filtro total, que consta de 24 muestras (como cuatro corresponden a cada símbolo, el filtro tiene en cuenta la interferencia de 6 símbolos contiguos), en cuatro filtros de longitud 6. Para estos cuatro filtros sólo es necesario un buffer de estado que se actualiza a velocidad de símbolo (2400 veces por segundo), es decir al haber utilizado todos los filtros del banco de filtros. En la siguiente figura se muestran los fundamentos del filtrado polifase.



Se ve claro como sólo parte de las muestras del filtro original multiplican a muestras diferentes de cero de la señal original. En el siguiente instante de tiempo, las muestras se desplazan hacia la derecha y las muestras del filtro original que corresponden con las muestras distintas de cero de la señal original son diferentes, formando la siguiente fase del filtro polifase

El funcionamiento se muestra en la siguiente figura:



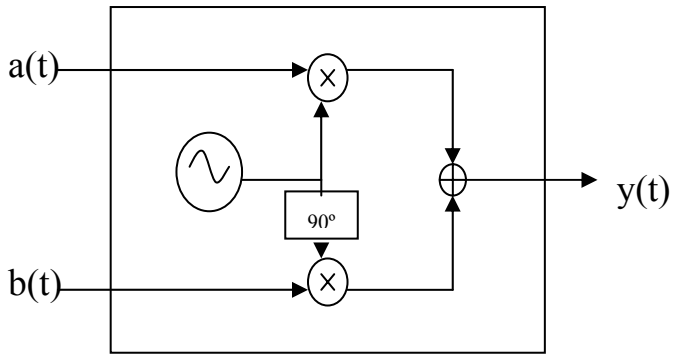
A la entrada del filtro tenemos el flujo de símbolos a velocidad r (2400 símbolos/s) y a la salida hay un conmutador que elige la salida del filtro correspondiente en cada instante para dar el flujo de muestras a 9600 muestras por segundo.

- Especificaciones del filtro: factor de Roll-Off = 0.3 para que la densidad de energía transmitida a 600 Hz y 3000 Hz esté atenuada 4.5 ± 2,5 dB, tal y como se especifica en la norma. Los coeficientes del filtro se obtienen con Matlab.

4.1.7 Modulador

Se realiza mediante la clase CModulator definida en modulator.h e implementada en modulator.cpp.

Su función es modular una portadora de 1800Hz con las dos salidas del filtro de transmisión, siguiendo el esquema de la figura:



Para implementar la generación de la portadora se pueden emplear dos maneras. La primera que se nos ocurre es ir generando continuamente la portadora empleando las funciones sin y cos de la librería. Esta manera tiene la ventaja de que se puede cambiar la frecuencia de la portadora y va a ser necesario emplearla para simular el efecto Doppler típico en los canales telefónicos. Se implementa en la función local_oscilator( ) del fichero carrier.cpp y se va a emplear para la modulación en el transmisor.

La segunda manera es más eficiente: se almacenan los valores de la portadora de 1800Hz en un buffer que se va recorriendo de forma circular aprovechando la periodicidad de la señal. El desfase de 90º se realiza gestionando dos punteros al buffer circular que van a estar distanciados en todo momento 4 posiciones (siendo la longitud del buffer 16 muestras). La razón de que sea esa la longitud es la siguiente: 1800/9600 = 3/16. Esto significa que cada tres periodos (16 muestras) se repite la señal y de esta manera tenemos una señal periódica. Se implementa en la funcion dem_carrier_set( ) del fichero carrier.cpp, y va a ser empleada en la predemodulación.

Para la gestión de la portadora se ha realizado la clase CCarrier que está definida en carrier.h.

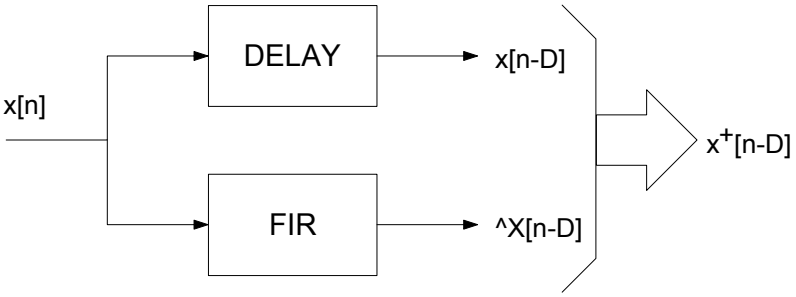
4.2 Receptor

Los bloques de la parte inferior de la figura general del módem (de la bobina híbrida hacia abajo) forman parte del receptor. Son los siguientes (siguiendo el recorrido de la señal):

4.2.1 Phase Splitter (obtención de la señal analítica)

Se realiza mediante la clase CphaseSplitter definida en phasesplit.h e implementada en phasesplit.cpp.

Para realizar el transformador Hilbert que es necesario para obtener la señal analítica, se emplea un filtro FIR cuya respuesta es la de un transformador Hilbert truncada para tener longitud finita. Para que el filtro sea causal hay que retardar la respuesta impulsional (N-1)/2, siendo N la longitud del filtro. Para que la parte real y la imaginaria (la que sale del transformador) de la señal analítica no estén desfasadas, se retardará la parte real (N-1)/2 muestras.



Puesto que los coeficientes impares del transformador Hilbert son cero, para evitar multiplicaciones innecesarias se va a realizar el filtrado de una manera especial: se eliminan los ceros de la respuesta impulsional y se emplean 2 bufferes de estados diferentes alternativamente.

4.2.2 Oscilador local (predemodulación ⇒ obtención del equivalente paso bajo)

Se realiza mediante la clase CPreDem definida en predem.h e implementada en predem.cpp.

Esta función consiste en realizar una demodulación preliminar, es decir pasar de la señal analítica al equivalente paso bajo, mediante la siguiente multiplicación compleja:

$$\tilde{x}[n] = \left(x[n] \cdot cos(\omega \cdot n \cdot T) - \hat{x}[n] \cdot sen(\omega \cdot n \cdot T) \right) + j \cdot \left(\hat{x}[n] \cdot cos(\omega \cdot n \cdot T) + x[n] \cdot sen(\omega \cdot n \cdot T) \right)$$

El error de fase que pueda haber entre las portadoras empleadas en modulación y predemodulación hará que la constelación esté girada un ángulo igual al desfase. La gestión de las portadoras se hará mediante la clase CCarrier ya mencionada, y en concreto para la predemodulación se va a emplear la generación de portadora mediante la función dem_carrier_set ya mencionada.

La predemodulación se hace para las 4 muestras de las que se compone cada símbolo, aunque posteriormente solo hagan falta 2 de ellas. La razón de esto es que se puedan trasladar las muestras predemoduladas a ficheros wav con frecuencia de muestreo de 9600 para su monitorización.

4.2.3 Control automático de ganancia (AGC)

Se realiza mediante la clase CAgc definida en agc.h e implementada en agc.cpp.

Su misión es que a los bloques del receptor que están a continuación les llegue un nivel constante de señal aunque a la entrada haya variaciones. El AGC funciona a 4800muestras por segundo, porque el ecualizador solo necesita 2 de cada 4 muestras correspondientes al símbolo. Por otra parte el control del AGC (el ajuste del factor por el cual se multiplica la amplitud de las muestras predemoduladas) solo se lleva a cabo una vez por cada símbolo.

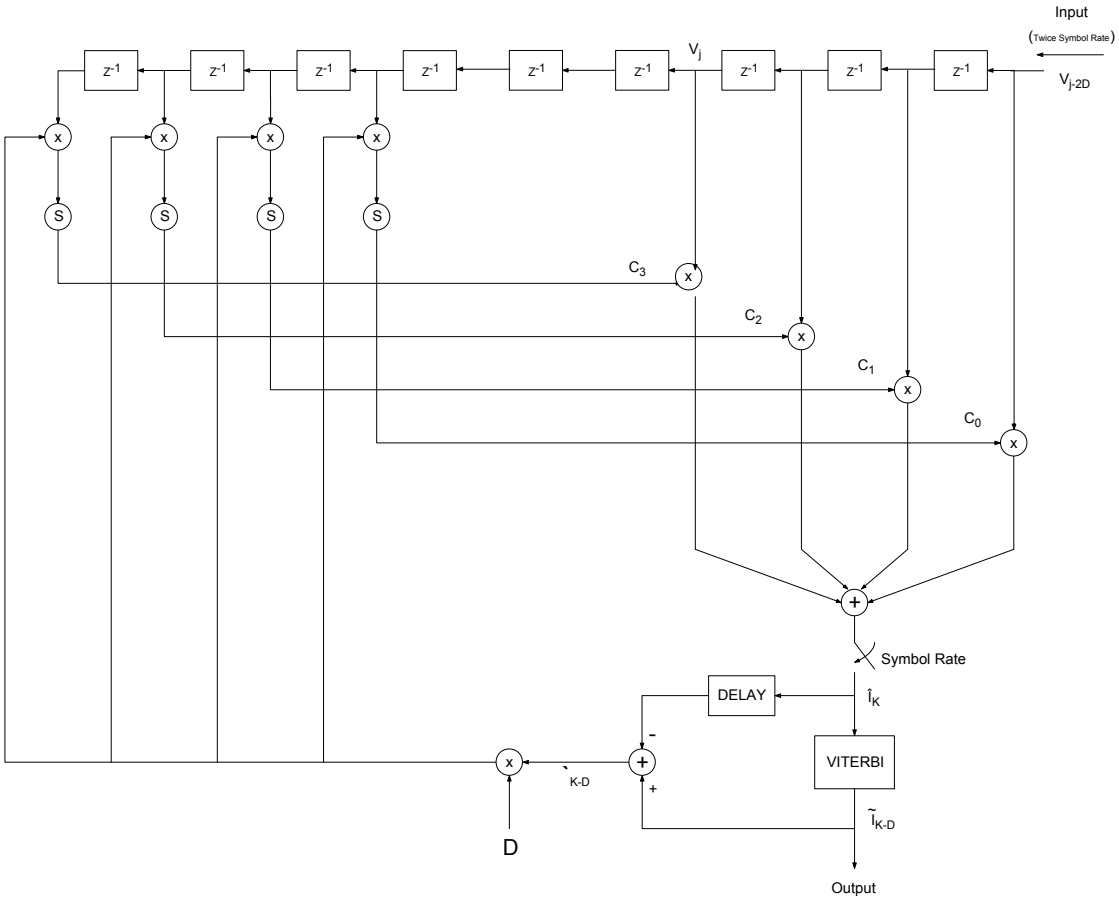
El control del AGC se realiza de la siguiente manera: se cuenta un número fijo de muestras y se guarda el valor máximo de sus módulos. A continuación se actualiza el factor de multiplicación teniendo en cuenta ese valor máximo y unas constantes.

4.2.4 Ecualizador adaptativo

Se realiza mediante la clase CAdaptiveEqualizer definida en equalizer.h e implementada en equalizer.cpp.

Se ha empleado un ecualizador lineal adaptativo fraccionalmente espaciado, es decir, que funciona a doble frecuencia de símbolo pero la salida es a frecuencia de símbolo.

Para la adaptación utiliza el algoritmo LMS aplicando el criterio MSE y tiene la peculiaridad de que la señal de error utilizada para adaptar los coeficientes tiene un retardo introducido por el decodificador de Viterbi, que tal y como se explicará posteriormente, va a ser de 16 símbolos. Esto implica que necesitamos un buffer de estado para el filtro lo suficientemente largo como para abarcar este retardo, ya que para actualizar los coeficientes con el algoritmo LMS hay que utilizar las muestras de la señal de entrada que generaron ese error retardado. El esquema simplificado se representa en la siguiente figura:



En el esquema del ejemplo se ha utilizado un equalizador FIR de 4 coefientes, que como va a doble frecuencia de símbolo va a ser capaz de quitar la interferencia entre símbolos introducida por 2 símbolos. El decodificador de viterbi tiene un retardo de 3 símbolos (D=3), es decir, decide el camino óptimo cada secuencia de 3 símbolos. Esta es la razón de que el buffer de estado del filtro tenga 6 retardos más (2D), ya que ha de almacenar las muestras correspondientes a este retardo de 3 símbolos (6 muestras). Concretamente hacen falta las cuatro muestras más viejas de este buffer, que van a ser las que utilice el algoritmo LMS para actualizar los coeficientes del filtro. Este ajuste se realiza a frecuencia de símbolo, que es cuando se obtiene una muestra de error nueva. Esta muestra de error es la que tiene un retardo D, por lo que no se pueden utilizar directamente las muestras del buffer utilizadas en el filtrado de la muestra actual sino las que se utilizaron D símbolos antes (2D muestras antes).

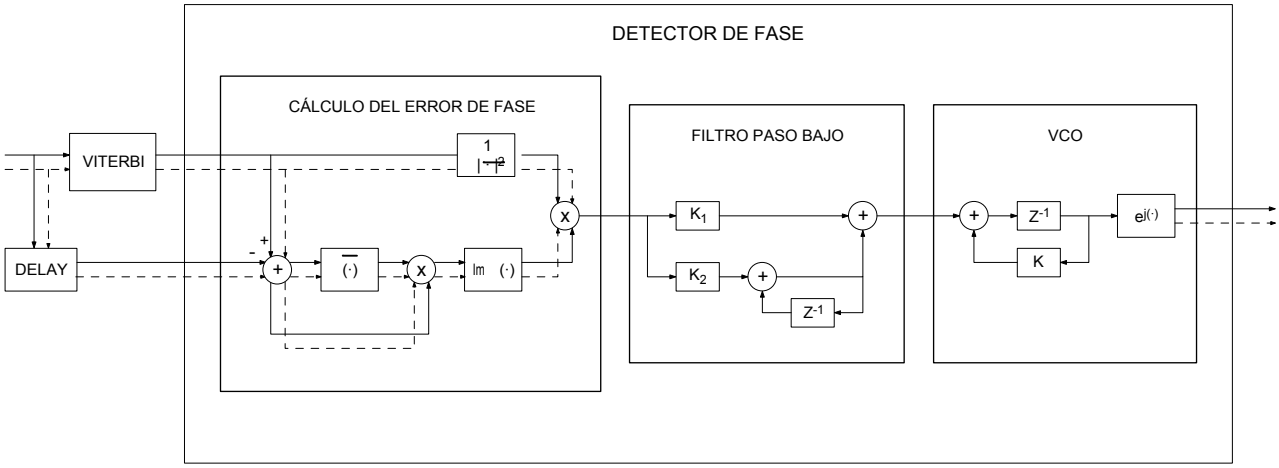
Especificaciones del filtro: tendrá 24 coeficientes (elimina la interferencia entre símbolos introducida por 12 símbolos) y el retardo del ecualizador de Viterbi es de 16 símbolos (D=16), por lo que el buffer de estado del ecualizador necesita almacenar 32 muestras más para utilizar las doce más viejas en el ajuste de los coeficientes (la longitud del buffer de estado será de 56 muestras). Todo esto es sólo para una rama, pero hay que tener en cuenta que el ecualizador funciona con dos ramas en paralelo, para la señal en fase y en cuadratura.

Otra particularidad que es necesario tener en cuenta es que el ecualizador va a funcionar con la constelación girada, por lo tanto, el error que se realimenta tiene que ser girado también para que la adaptación pueda realizarse. Para hacer eso se empleará el mismo factor que para corregir la señal de salida del ecualizador, y ese factor será lo que se obtenga del Carrier Tracking.

4.2.5 Detección de portadora (carrier tracking)

Se realiza mediante la clase CCarrierDetect definida en carrdet.h e implementada en carrdet.cpp.

Su función es corregir el error de fase que se comete en la predemodulación como ya se ha comentado. Para conseguirlo se implementa el PLL de la figura:



En el filtro paso bajo, la constante K1 valdrá 0.1 y K2 0.01. Los valores de estas constantes han sido establecidos por prueba y error. La constante del VCO vale 0.8 y es una constante de estabilización.

4.2.6 Decodificador de Viterbi

Se realiza mediante la clase CviterbiDecoder definida en decoder.h e implementada en decoder.cpp.

Este bloque se encarga de realizar la búsqueda de la secuencia de símbolos más probable utilizando el algoritmo de Viterbi. Se puede realizar esta decodificación, cuya principal ventaja es la disminución de la probabilidad de error, gracias al codificador convolucional (Trellis) empleado en la modulación, que introducía un bit redundante.

En este programa se han implementado dos versiones diferentes del decodificador de Viterbi, y el usuario podrá conmutar entre ellas mediante el menú o la barra de herramientas.

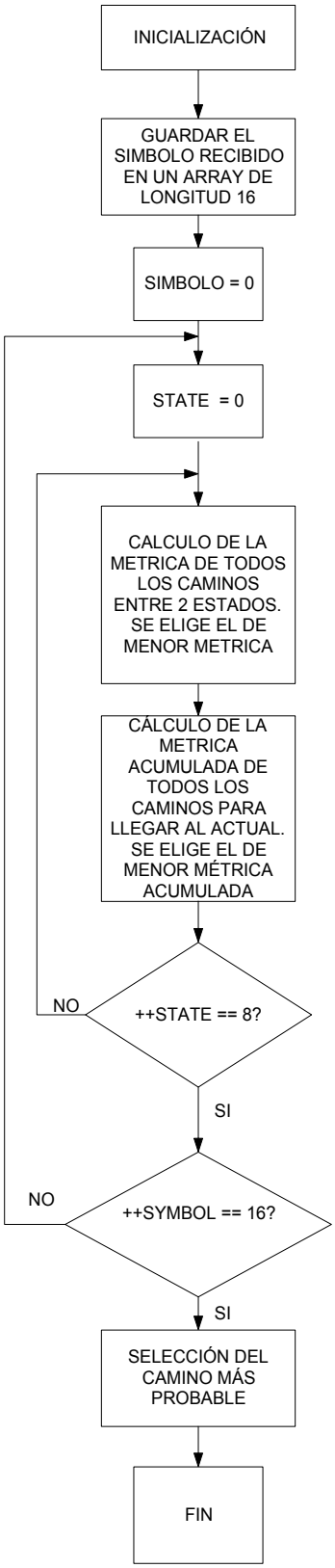
Este algoritmo realiza la decodificación en tramas de longitud 16, y dependiendo de la versión del decodificador que se use se abren dos posibilidades:

- Cada 16 símbolos el decodificador decide cual de los caminos posibles es el más probable y vuelve a empezar con el algoritmo.
- Cada símbolo recibido, el ecualizador decide un símbolo ocurrido 16 intervalos antes.

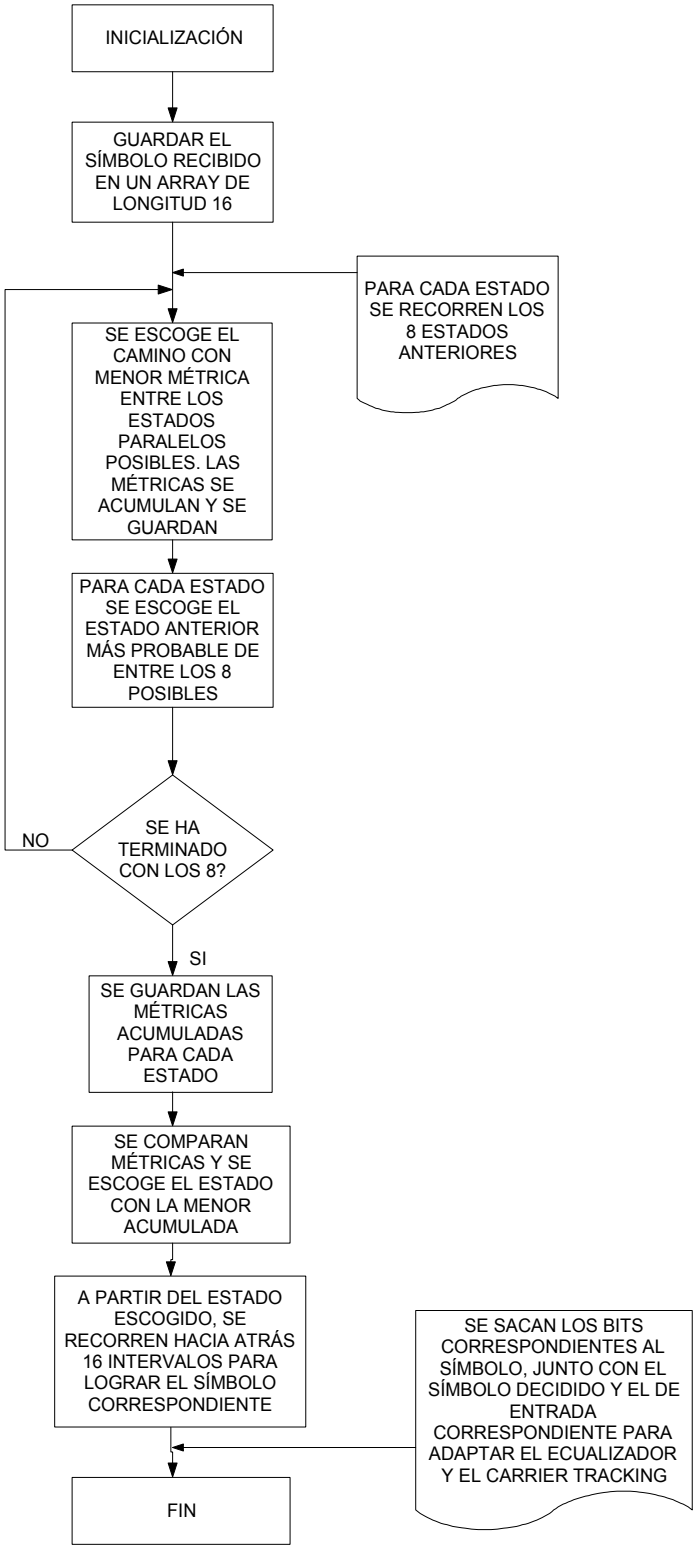
De cualquiera de las dos maneras, esto implica un retardo de 16 símbolos entre la secuencia de salida y a la secuencia de entrada, y habrá que tenerlo en cuenta en el equalizador adaptativo como ya se ha mencionado. La razón de que el retardo sea 16 es porque la teoría dice que la decodificación se puede truncar cada $2^{(L+1)}$ (siendo L el número de iteraciones

hasta alcanzar el estado estacionario y que en nuestro caso es 3 por el tipo de codificador que se está usando) siendo despreciables los efectos negativos.

El primero de los algoritmos implementados sigue el siguiente diagrama de flujo:



Mientras que para el segundo de los algoritmos, el diagrama de flujo será el siguiente:

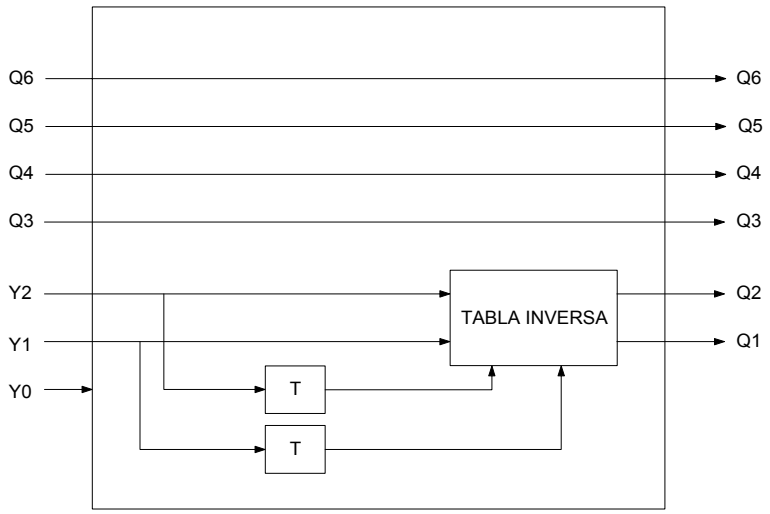


El decodificador de Viterbi y el Slicer devuelven los bits correspondientes al símbolo decidido (es decir, hace la operación inversa al mapeo 2D del modulador) junto con el propio símbolo decidido y el ecualizado correspondiente para el cálculo del error que se tiene que realimentar al ecualizador.

4.2.7 Decodificador diferencial

Se realiza mediante la clase CDiffDecod definida en diffdecod.h e implementada en diffdecod.cpp.

La función de este bloque es deshacer la codificación diferencial que se hace en transmisión. Lo único que hay que hacer es aplicar una de las dos tablas empleadas en la codificación diferencial pero de forma inversa. El esquema del bloque se representa en la siguiente figura.



4.2.8 Conversor paralelo-serie (P/S)

Se realiza mediante la clase CSer_Par que es la que se empleaba en la conversión serie-paralelo y que está definida en serpal.h, y se implementa en serpal.cpp.

Este bloque consiste en transformar los bloques de bits en paralelo generados por el decodificador diferencial en un flujo de bits en serie. Se trata de un registro de desplazamiento del cual se saca un bit cada vez que es llamada la función par_ser_do.

4.2.9 Descrambler (desaleatorizador)

Se realiza mediante la clase CScramb, que es la que se empleaba en el scrambleado del transmisor y que esta definida en scramb.h e implementada en scramb.cpp.

Esta función se encarga de desaleatorizar los datos para recuperar la secuencia original que se introdujo en el scrambler del modulador. Para ello se utiliza el mismo polinomio generador que se empleaba en el scrambler y que viene especificado en la norma V.32bis (PGL). Al igual que en el scrambler, existen dos maneras de hacer el descrambleado, dependiendo de si se define o no la variable SCRAMB_FAST en scramb.h.

4.3 Módem completo

El módem completo se encuentra definido en la clase CModem definida en modem.h e implementada en modem.cpp. La clase contiene los datos miembro necesarios para todas las tareas que ha de realizar el módem:

- El transmisor, que será del tipo CTransmitter.

- El receptor, que será del tipo CReceiver.
- El canal, que será del tipo CChannel.
- La constelación que se esté empleando, que será del tipo CConstel.
- La ventana donde se van a dibujar todos los resultados, que será un puntero al tipo CChildView.
- La estructura donde se guardan todos los datos acerca de la configuración seleccionada por el usuario.

El transmisor y el receptor a su vez, tendrán los datos miembro necesarios para ejecutar sus tareas, y en general tendrán un dato miembro por cada uno de los bloques que ya se han descrito. El transmisor se realiza mediante la clase CTransmitter definida en transmitter.h e implementada en transmitter.cpp. Por su parte, el receptor se realiza mediante la clase CReceiver definida en receiver.h e implementada en receiver.cpp.

5 EXPLICACIÓN DEL SOFTWARE. MANUAL DEL USUARIO

5.1 Puesta en marcha del modem. Configuración

Al cuadro de dialogo "Configuración del módem" se accede bien desde el menú Comandos-Configuración o bien mediante el botón de la barra de herramientas. Mediante este cuadro de dialogo se pueden seleccionar las siguientes características:

- Velocidad de transmisión en bps: Se podrá seleccionar entre los valores posibles de 14.4Kbps, 12Kbps, 9.6Kbps, 7.2Kbps, y 4.8Kbps. Por defecto está seleccionada la opción de 4.8Kbps.
- Desviaciones respecto del funcionamiento óptimo del módem. Se definen 3:
    - Error de fase entre las portadoras de emisión y recepción. Este error se encargará de corregirlo el subsistema de detección de portadora.
    - Error de sincronismo entre el transmisor y el receptor. Los símbolos que se transmiten se componen de 4 muestras cada uno (debido a que hay que transmitir 2400símbolos/s y la frecuencia de muestreo es de 9600muestras/s para soportar el ancho de banda de la señal), y existe un instante óptimo para la recepción que coincide con el instante de máxima apertura del diagrama de ojo. El error de sincronismo se refiere al número de muestras de desviación respecto a ese instante óptimo. El encargado de corregir este error es el ecualizador adaptativo.
    - Translación de frecuencia o efecto Doppler. Consiste en efectuar una translación de frecuencias en la portadora de modulación de tal manera que la nueva frecuencia sea 1800 ± 7Hz (es el valor que se indica en la recomendación como máximo valor a tener en cuenta, pero se pueden emplear valores mayores). El encargado de corregir este error será al igual que en el primer caso, el subsistema de detección de portadora. Para poder hacer esto, no se puede implementar un desplazamiento de frecuencia en el canal ya que acarrea un retardo que hace que sea imposible sincronizar la secuencia de adaptación en el receptor. Por esa razón, en el modulador la portadora se va generando en todo momento con la frecuencia determinada en vez de tenerla guardada en un buffer. Es menos eficiente computacionalmente pero es la manera correcta de hacerlo.
- Opciones de visualización. IMPORTANTE: Antes de abrir los ficheros de salida con un programa de edición (p.ej el Notepad o el CoolEdit, según el caso), hay que cerrar la aplicación. Por otra parte, estos ficheros no pueden estar abiertos si se quiere que la aplicación escriba en ellos.
    - Activación/desactivación de la salida de texto del modulador. Permite visualizar los bits y símbolos correspondientes al sub-bloque de bit del modulador (antes del filtro de transmisión). En el fichero de texto, que se denominará **modfile.txt** aparecen los siguientes campos:

Bits generados → bits scrambleados → bits codificados diferencialmente y convolucionalmente (mediante el codificador 2/3 para las velocidades superiores a 4800bps) → símbolo a transmitir

- Activación/desactivación de la salida de texto del demodulador. Permite visualizar los bits y símbolos correspondientes al sub-bloque de bit del demodulador (a partir del decodificador de Viterbi). En el fichero de texto, que se denominará **demfile.txt** aparecen los siguientes campos:

  Símbolo decidido → bits correspondientes con bit redundante (el que introduce el codificador convolucional para velocidades superiores a 4800bps) → bits correspondientes al símbolo sin el redundante → bits decodificados diferencialmente → bits descrambleados.

- Activación/desactivación de la salida a fichero de audio (formato .wav) de las diferentes señales implicadas:
  - Señal de salida del filtro de transmisión, es decir, la señal QAM en banda base, para los dos canales (en estéreo): Se guarda en **IQSignals.wav**.
  - Señal de salida del modulador (paso banda) en mono. Se guarda en **OutputSignal.wav**.
  - Señal de salida del Phase Splitter para los dos canales (en estéreo). Se guarda en **PSSignals.wav**.
  - Señal pre-envolvente (señal predemodulada a la salida del AGC) para los dos canales (en estéreo). Se guarda en **PDSignals.wav**.

El cuadro de diálogo de configuración se realiza mediante la clase CConfigDlg definida en ConfigDlg.h e implementada en ConfigDlg.cpp.

Cuando se pulsa el botón de configuración, se ejecuta la función OnConfigure implementada en mainfrm.cpp. En esta función, después de recoger los datos que el usuario haya introducido en el cuadro de diálogo, se copian a una variable miembro de la clase CModem para que puedan ser usadas en las inicializaciones de todos los módulos. Esta variable será del tipo struct ConfigData definida en el fichero de cabecera general v32bis.h.

5.2 Comandos de funcionamiento

Se trata de los comandos que controlan el funcionamiento general del módem. Los procedimientos definidos son muy simples:

- Start: Comienza el funcionamiento para unos parámetros de configuración determinados. Por defecto se empieza dibujando el espectro en vez del diagrama Trellis. Antes de empezar a dibujar nada, se borra la pantalla para eliminar rastros de ejecuciones anteriores. A continuación se lanza el Thread que se encargará de ejecutar el módem y representar los resultados de manera independiente al resto de tareas que se estén ejecutando en el sistema.

  Cuando se pulsa el botón de Start, se ejecuta la función OnModemstart implementada en mainfrm.cpp. La función principal del Thread será ModemThreadFunc también implementada en mainfrm.cpp. Esta función define el siguiente proceso:

-Resetear el modem;
-Mientras no se pulse Stop:
-Se generan 4 muestras con el transmisor (1 símbolo);
-Se añaden las distorsiones del canal;
-Se demodulan las 4 muestras;
-Se representan los resultados en la pantalla;
-Finalmente se cierra el módem para liberar los recursos reservados.

- Stop: Se detiene el funcionamiento del módem y se liberan todos los recursos empleados. También se desactivan las distorsiones que pudieran haberse activado durante el funcionamiento. Cuando se pulsa el botón Stop, se ejecuta la función OnModemStop implementada en mainfrm.cpp.
- Pause / Resume: Se detiene el funcionamiento del módem temporalmente y se vuelve a lanzar cuando se pulsa este botón. Lo único que se hace es detener el Thread esté donde esté, es decir, no se tiene en cuenta la función que se pueda estar ejecutando en el momento de la pausa. Esto es útil entre otras cosas para ver paso a paso la evolución del diagrama Trellis. Cuando se pulsa el botón Pause, se ejecuta la función OnModemPause implementada en mainfrm.cpp.
- Trellis Draw: Se emplea para conmutar entre la visualización del espectro de la señal modulada y el diagrama Trellis que genera el decodificador de Viterbi. Es importante tener en cuenta dos cosas:
  - El diagrama Trellis solo se puede dibujar cuando se esté empleando el algoritmo de Viterbi #0 (el primero de los explicados en el apartado 4.2.6).
  - Cuando se dibuja el diagrama Trellis, al ser una tarea que exige muchos cálculos al procesador, la ejecución del módem es mucho más lenta.

  Cuando se pulsa este botón, se ejecuta la función OnTrellisDraw implementada en mainfrm.cpp.
- Viterbi Select: Se emplea para conmutar entre los dos algoritmos de Viterbi implementados en el módem, que ya han sido explicados en el apartado 4.2.6. Por defecto está activada la opción 0, es decir, que se ejecutará el algoritmo que permite la visualización del diagrama Trellis. Cuando se pulsa este botón, se ejecuta la función OnVitAlg0 implementada en mainfrm.cpp.

5.3 Distorsiones a la señal

La RTGC introduce un gran número de distorsiones, y para este módem se han querido implementar las más importantes: El ruido blanco aditivo, la limitación en banda del canal y las translaciones de frecuencia causadas por el efecto Doppler. Ésta última ya ha sido comentada en el apartado dedicado a la configuración. A las otras dos distorsiones se accede mediante el menú Distorsiones o mediante los botones de la barra de herramientas.

- Ruido gaussiano blanco: Se trata de una señal aleatoria que se suma a la señal antes de entrar al receptor con una amplitud determinada. La amplitud se controla mediante el menú o los botones + y -. La ventana situada en la parte inferior derecha del diagrama del espectro nos muestra la relación señal a ruido en cada momento.

- Limitación en banda del canal: Se trata de filtrar la señal modulada con un filtro paso banda que tendrá una banda de paso entre 300 y 3400Hz. No afecta demasiado a la recepción debido a que el filtro de transmisión ya se encarga de conformar la señal de acuerdo con las características del canal.

5.4 Visualización

Cuando se lanza el programa, aparece una ventana con 4 áreas diferenciadas:

- En la ventana superior se dibujará el espectro de la señal modulada o el diagrama Trellis del decodificador de Viterbi cuando el usuario así se lo indique. Cuando se dibuja el espectro de la señal, el eje de abscisas viene en Hz y el de ordenadas en dBs.
- Justo debajo de esa ventana, se pintarán los bits a la salida del demodulador. Puesto que se están modulando todo unos, si lo que aparece en esa área son todo unos significa que la demodulación es correcta.
- Junto a los bits descrambleados, se visualizan la relación señal a ruido en dB a la entrada del receptor y el Bit Error Rate. El cálculo de la relación S/N se realiza en el momento de añadir el ruido a la señal, y el BER no es más que un contador de ceros que se suaviza para que se pueda leer (se filtra paso bajo). Concretamente, se cuentan los ceros por cada 1000 bits demodulados. El valor del BER inicial se debe al régimen transitorio del receptor.
- En la parte inferior de la ventana de la aplicación aparecen tres recuadros. En el de la izquierda se va a dibujar el diagrama 2D de la señal predemodulada o dicho de otra manera, de la preenvolvente de la señal que se recibe en el receptor. En concreto, se dibujará la señal compleja que hay a la salida del AGC (tan solo uno de los dos símbolos). En la parte superior de este recuadro se representa el valor que toma el AGC en cada momento.
- En el recuadro del centro se representa el diagrama 2D de la señal compleja a la salida del ecualizador. Sobre esta representación se pinta el valor del error cuadrático que se tiene en cada momento. El error cuadrático se define como la raíz de la suma de los cuadrados de las partes real e imaginaria del error entre el símbolo decidido y el ecualizado.
- En el recuadro de la derecha se representa el diagrama 2D de la señal compleja a la salida del ecualizador pero con la fase corregida por el sub-bloque de detección de portadora, es decir, la señal sobre la cual el decodificador de Viterbi o el Slicer van a tomar las decisiones. Sobre este recuadro se pinta la corrección en grados que el bloque de detección de portadora está aplicando en cada momento. Hay que tener en cuenta que el giro fijo que tiene la constelación a la salida del ecualizador (cuando no hay desplazamiento de frecuencia) viene determinado por diferentes factores, por lo tanto no tiene por que coincidir el valor de la corrección con el error de fase que el usuario haya definido. Estos factores van a ser tres:
    - El desfase inicial entre portadoras de transmisión y recepción, que puede ser cualquiera.
    - El error de fase que el usuario haya definido.
    - El error de sincronismo que el usuario haya definido.