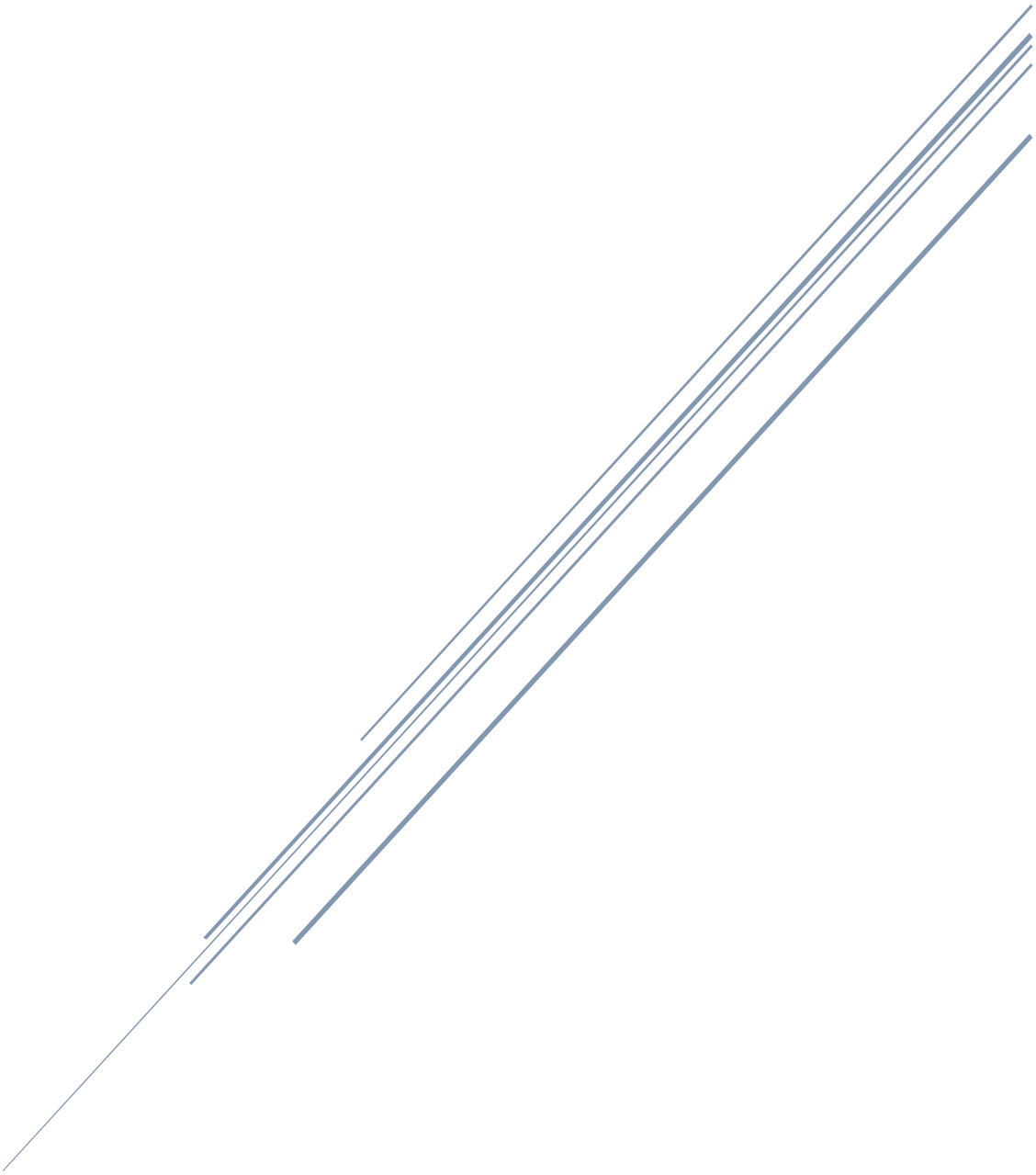
# Receptor OFDM DVBT Simplificado

## Electrónica Digital para Comunicaciones



Pablo Linares Serrano

2019-2020

# Índice

# 1. Introducción

El presente documento describe el modelado e implementación de un receptor OFDM DVBT simplificado realizado para la asignatura Electrónica Digital para Comunicaciones del Máter de Ingeniería de Telecomunicación de la Universidad de Sevilla.

# 2. Modelado del sistema en Matlab

Partiendo del sistema incompleto suministrado en la primera práctica de la asignatura se ha modelado un receptor OFDM DVBT simplificado. Este realiza la estimación del canal que ha atravesado la señal y ecualiza los símbolos. A continuación, se detallan las funcionalidades implementadas en el script 'MLdef.m'.

## Funcionalidades

### Constelaciones

Se permite el uso de las constelaciones BPSK y QPSK. Se puede elegir entre ambas mediante el parámetro 'CONSTEL'.

### Símbolos

Se puede emplear símbolos de longitud 2k o de longitud 8k, fijando el parámetro 'MODE'

### Canales

Se puede elegir entre simular un canal P1 o F1 de los definidos en el estándar DVBT. Esta elección se realiza mediante el parámetro 'CANAL'.

## Bloques del transmisor

En este apartado se describirán de forma sencilla los bloques principales del sistema. Algunos de ellos serán implementados en VHDL. Se puede ver el código completo en 'MLdef.m'

### Secuencia PRBS

Generamos la secuencia PRBS mediante el segmento de código:

```
PRBS = ones(1,NCARRIER);
for j = 12:NCARRIER
    PRBS = [PRBS(1:j-1),xor(PRBS(j-11),PRBS(j-9)),PRBS(j+1:end)];
end
PRBS = PRBS(PILOTOS);
PRBS = (4/3)*(1-2*PRBS);
```

Donde 'PILOTOS' es un vector que almacena las posiciones que ocupan los pilotos dentro del símbolo. De esta forma, almacenamos únicamente los elementos de la secuencia que serán empleados más adelante.

### Conformación de los símbolos a transmitir

Conformamos los símbolos mediante el código:

```
ofdm_freq = zeros(NFFT, NUM_SYMB);
ofdm_freq(ceil((NFFT-NCARRIER)/2)+DATAP,:) = reshape(const_points,
NDATA, NUM_SYMB);
ofdm_freq(ceil((NFFT-NCARRIER)/2)+PILOTOS,:) =
repmat(complex(PRBS).',1,NUM_SYMB);
```

Donde 'DATAP' es un vector similar a 'PILOTOS', pero que contiene las posiciones ocupadas por los datos. Por otra parte, 'const_points' es un vector que contiene los puntos de la constelación

correspondientes a los bits que se van a transmitir. En la Figura 1 se puede ver el primer símbolo a transmitir. Los puntos de mayor energía se corresponden con los pilotos.

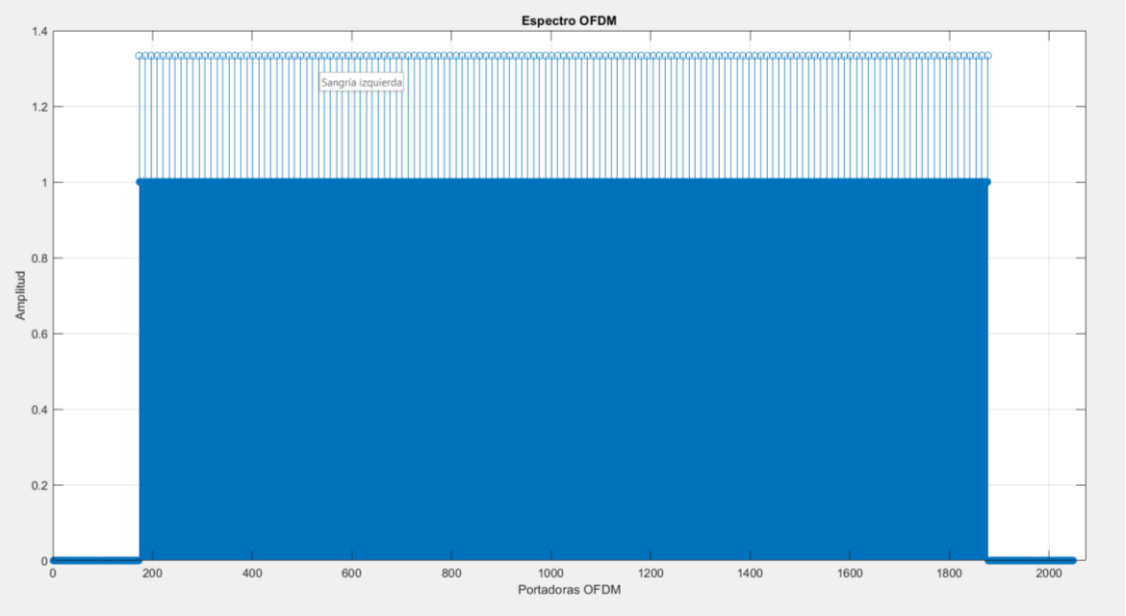



*Figura 1: Representación del primer símbolo antes de atravesar el canal y sumarle ruido.*

## Canal

Obtenemos el canal de la forma:

```
t = (0:NFFT-1)*T;
f = ((0:NFFT-1)-NFFT/2-1).'/(NFFT*T);
ro0 = sqrt(10*sum(ro.^2));
if CANAL == 'F1'
    H = (ro0+(ro.*exp(-1i*fi))*exp(-1i*2*pi.*f.*pos*1e-
6).')/sqrt(ro0^2+sum(ro.^2));
end
if CANAL == 'P1'
    H = (ro.*exp(-1i*fi))*exp(-1i*2*pi.*f.*pos*1e-
6).'/sqrt(ro0^2+sum(ro.^2));
End
```

El vector t contendrá los instantes muestreados de la respuesta impulsiva del canal y f las frecuencias muestreadas. Los vectores 'ro', 'fi' y 'pos' contienen los parámetros que definen los canales F1 y P1 en el estándar DVBT. Se obtienen al cargar el fichero 'MatCanal.mat', en el que se han almacenado estos datos. Tras la obtención del canal se convoluciona el símbolo en tiempo con el canal, para obtener el símbolo recibido. En la Figura 2 y Figura 3 se puede ver el canal obtenido.

## Bloques del receptor

Del símbolo recibido se desecha el prefijo cíclico. Después se realiza la fft para obtener el símbolo en frecuencia. El resultado de esta operación se vuelca en un fichero, para poder emplearlo después en la simulación del VHDL. Esto se realiza mediante las líneas:

```
rx_expR = reshape(real(rx_frec.'),1,[]);
rx_expI = reshape(imag(rx_frec.'),1,[]);
maximo = max(max(abs(rx_expR)),max(abs(rx_expI)));
rx_expR = round(rx_expR*511/maximo);
rx_expI = round(rx_expI*511/maximo);
```

```
dlmwrite('simb8kR_ML.dat', rx_expR, '\n');
dlmwrite('simb8kI_ML.dat', rx_expI, '\n');
```

En primer lugar, se reajustan los elementos para que sean enteros comprendidos entre 511 y -511, ya que los números empleados en la implementación VHDL serán de 10 bits. Después se exportan en dos ficheros mediante 'dlmwrite', uno para la parte imaginaria y otro para la parte real.

## Estimación del canal

Empleamos el código que se muestra a continuación. En él se extraen los pilotos del símbolo, dividiéndolos por los elementos apropiados de la secuencia PRBS. Después, se multiplica la diferencia entre en canal en los pilotos superiores en inferiores en cada segmento por los pesos apropiados. Finalmente, se suma el resultado de la operación al canal en el piloto inferior.

```
pilotosRx = rx_frec(:,ceil((NFFT-NCARRIER)/2)+PILOTOS).';
H1 = pilotosRx./repmat(complex(PRBS).',1,NUM_SYMB);

[a,b] = size(H1(2:end,:));
pesos = (1/12:1/12:11/12)';
H2 = pesos*reshape((H1(2:end,:)-H1(1:end-1,:)).', 1, a*b);
H2 = reshape(H2, length(pesos), b, a)+ repmat(permute(H1(1:end-1,:),
[3, 2, 1]),length(pesos),1,1);
H2 = reshape(permute(H2, [1,3,2]), a*length(pesos), b);

channel = zeros(NCARRIER, NUM_SYMB);
channel(DATAP, :) = H2;
channel(PILOTOS, :) = H1;
```

En la Figura 2 y la Figura 3 se ha representado el canal ideal según el estándar DVBT (azul) y el canal estimado mediante el código citado (naranja).

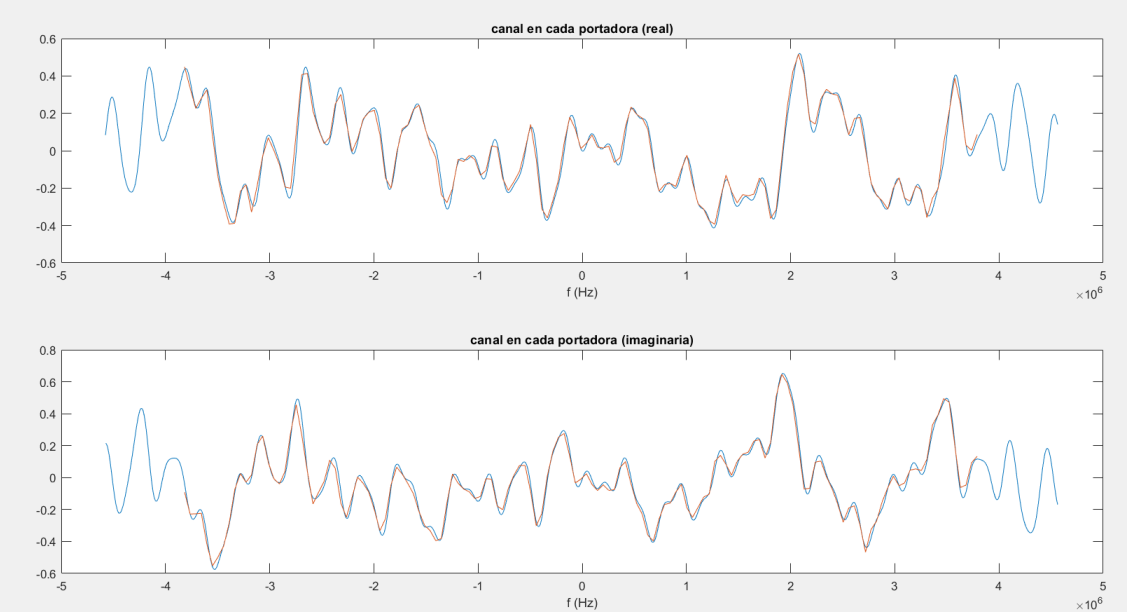



*Figura 2: Parte real y parte imaginaria del canal definido en el estándar DVBT (azul) y el canal estimado (naranja).*

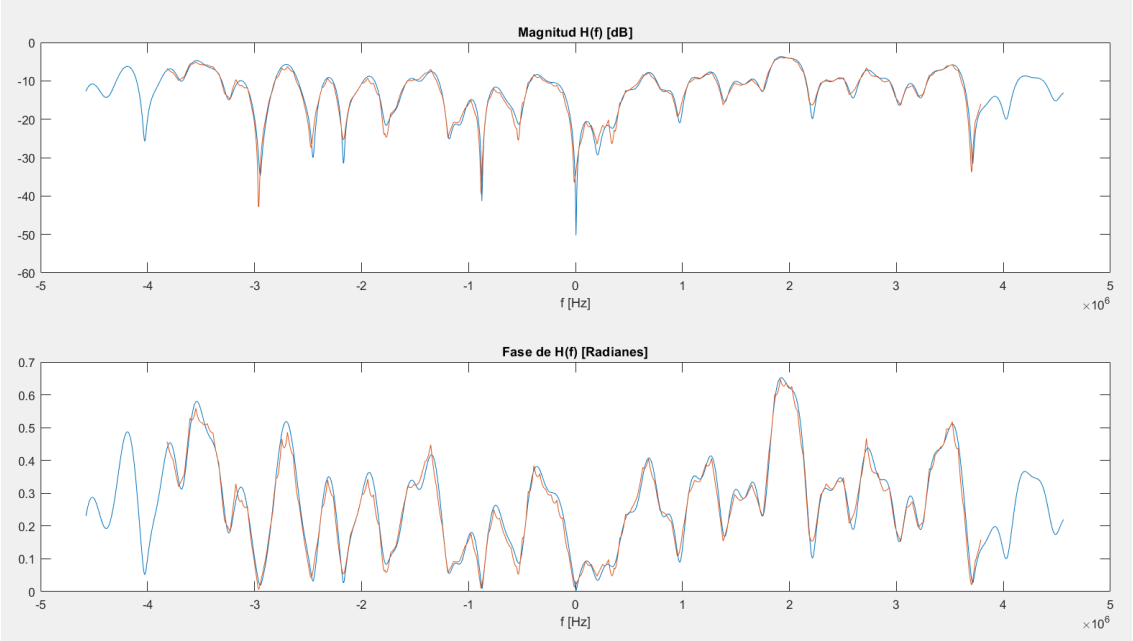



*Figura 3: Magnitud y fase del canal definido en el estándar DVBT (azul) y el canal estimado (naranja).*

## Ecualización

Obtenemos los datos ecualizados dividiendo los datos por el canal punto a punto:

```
x = x./H2;
rx_constel = reshape(x, 1, []);
```

X es un vector que contiene las portadoras correspondientes a las posiciones ocupadas por datos y H2 contendrá en canal en estas posiciones.

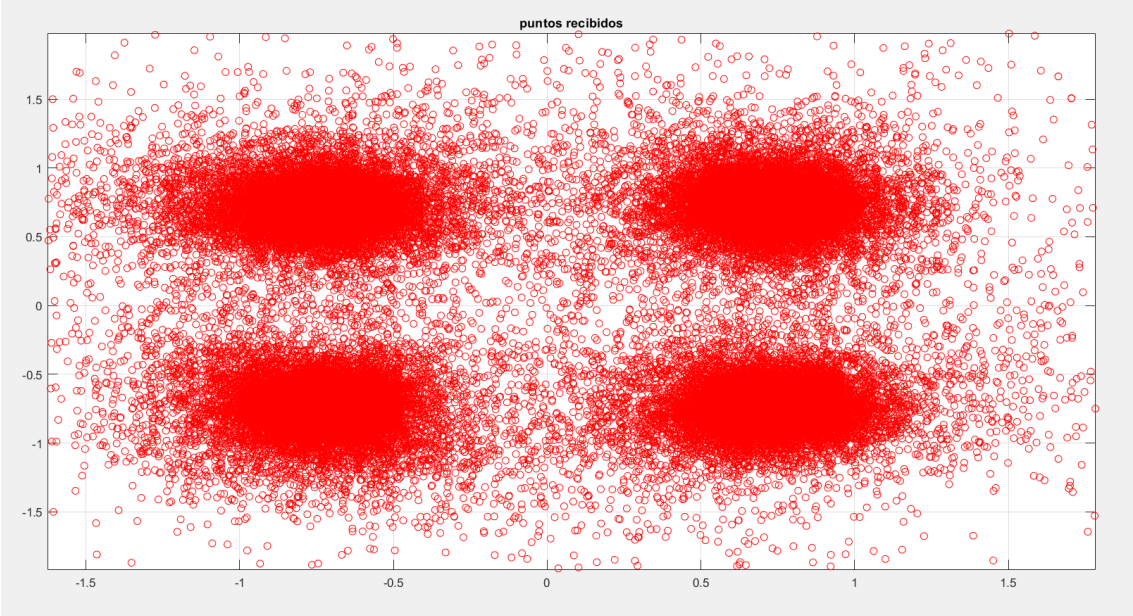



*Figura 4: Constelación obtenida tras ecualizar. Se ha empleado un símbolo 2k con modulación QPSK y canal P1.*

# 3. Implementación en VHDL

En este apartado se describirán los bloques y ficheros empleados para implementar el ecualizador DVBT simplificado. La memoria 'dpram.vhd' y el fichero 'divider.vhd' son los que se han suministrado a los alumnos. En ellos se han variado únicamente el tamaño de los números empleados.

## mypack.vhd

Este fichero contiene la definición del tipo 'complex10' que se empleará en el resto de ficheros y bloques. Este tipo está formado por dos campos. Cada uno de ellos es un número 'signed' de 10 bits. Uno correspondiente a la parte real y otro a la imaginaria de los números complejos que se emplearán.

## prbs.vhd

Implementa el bloque 'prbs'. Tiene un registro de desplazamiento que se inicializa a 1 al activar la señal 'rst'. Éste reset será síncrono, ya que se deberá activar al final de cada símbolo. La secuencia avanzará una posición cada vez que coincida un flanco de subida del reloj con un '1' lógico en la señal 'enable'. Será fundamental mantener el sincronismo de esta señal. En la señal 'salida' se recibirá la secuencia. La entrada al registro de desplazamiento se obtendrá mediante una puerta or de la salida y la octava posición del registro. Todas las entradas son de tipo 'std_logic'.

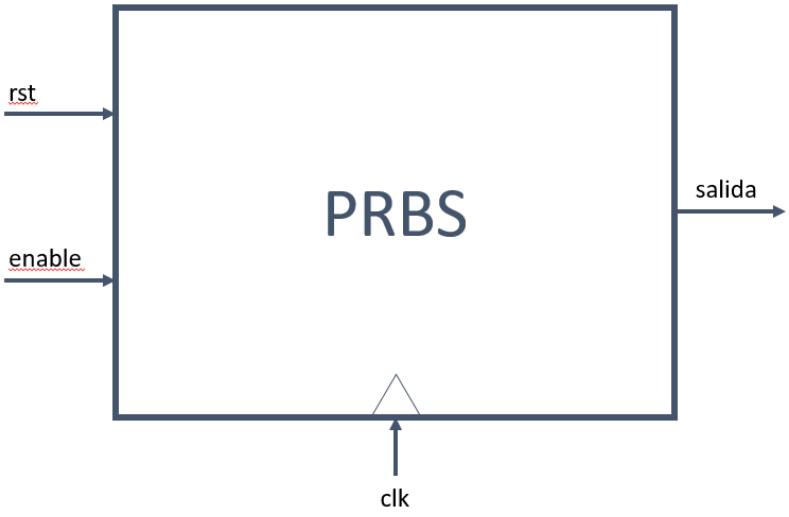



*Figura 5: Bloque prbs.*

## Interpolador.vhd

Contiene el bloque interpolador. Éste será el encargado de hacer la interpolación lineal entre el canal en el piloto superior e inferior. En las entradas 'sup' e 'inf' se recibirá el canal en dichos pilotos. Estas entradas serán de tipo 'complex10'. Las entradas no se registrarán, por lo que deberán mantenerse fijas hasta que se termine la interpolación de un segmento. Para ello, la señal 'okInterpolB' (tipo 'std_logic') dará un pulso cada vez que se obtenga un punto interpolado. Por tanto, las entradas 'sup' e 'inf' deberán actualizarse cada 12 pulsos de dicha señal.

La señal 'valid' se activará cuando se desee que el interpolador comience el proceso de interpolación. Será de tipo 'std_logic'. Las señales okInterpolF y okDato son ambas 'std_logic'. Señalizan cuando la salida del interpolador (estim) es correcta y cuándo el bloque siguiente ya la ha tomado respectivamente. La señal 'estim' será la interpolación correspondiente a la posición indicada por la señal 'pos'. Serán 'complex10' y 'unsigned' respectivamente.

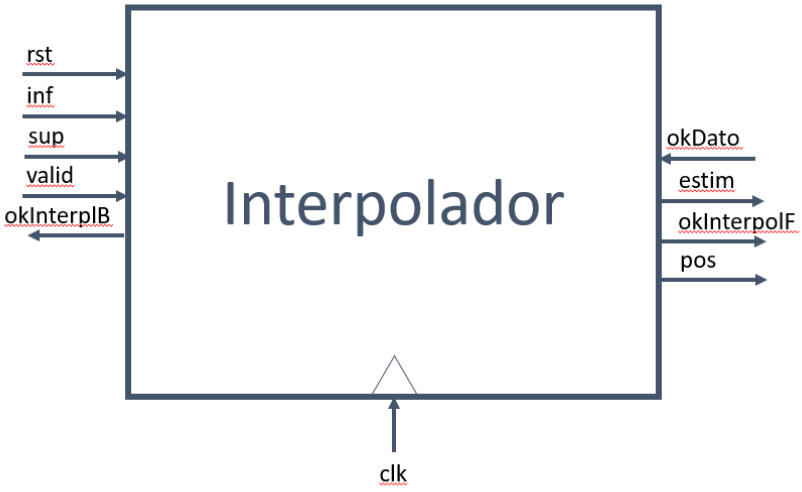



*Figura 6: Bloque interpolador.*

## divisor.vhd

Este fichero contendrá el bloque divisor. Éste bloque recibirá el canal estimado en un punto, la señal recibida en esa posición y devolverá la señal dividida por el canal. Para ellos hará uso del bloque 'divider' suministrado.

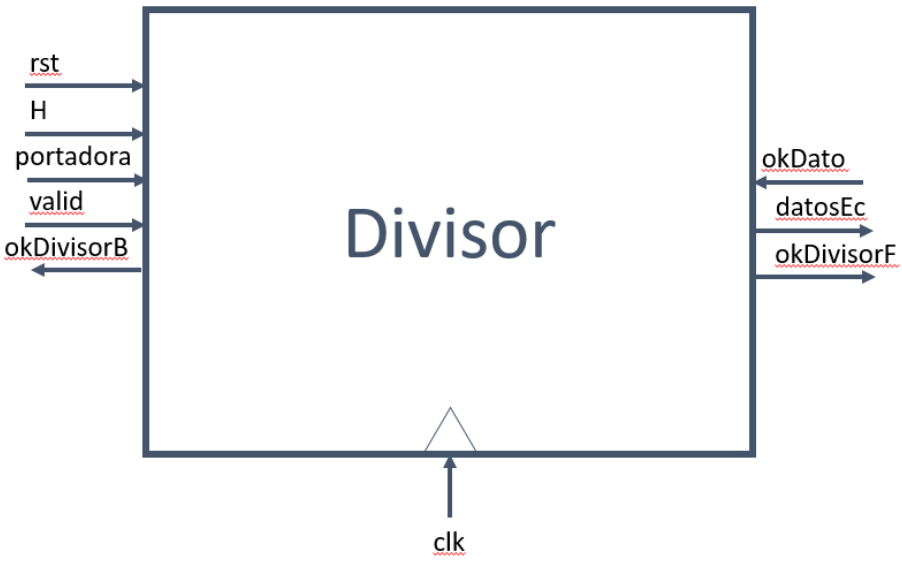



*Figura 7: Bloque divisor.*

El bloque divisor emplea al bloque 'divider' para invertir el denominador del conjugado del canal. Se ha optado por esta configuración frente a la propuesta porque de esta forma se emplea únicamente un divisor. El numerador de la multiplicación de la portadora por el conjugado del canal se obtiene de la forma:

$$denominador = ac + bd + (ad - bc)i$$

Siendo $a$ la parte real del canal, $b$ la parte imaginaria del canal, $c$ la parte real de la portadora y $d$ la parte imaginaria de la portadora.

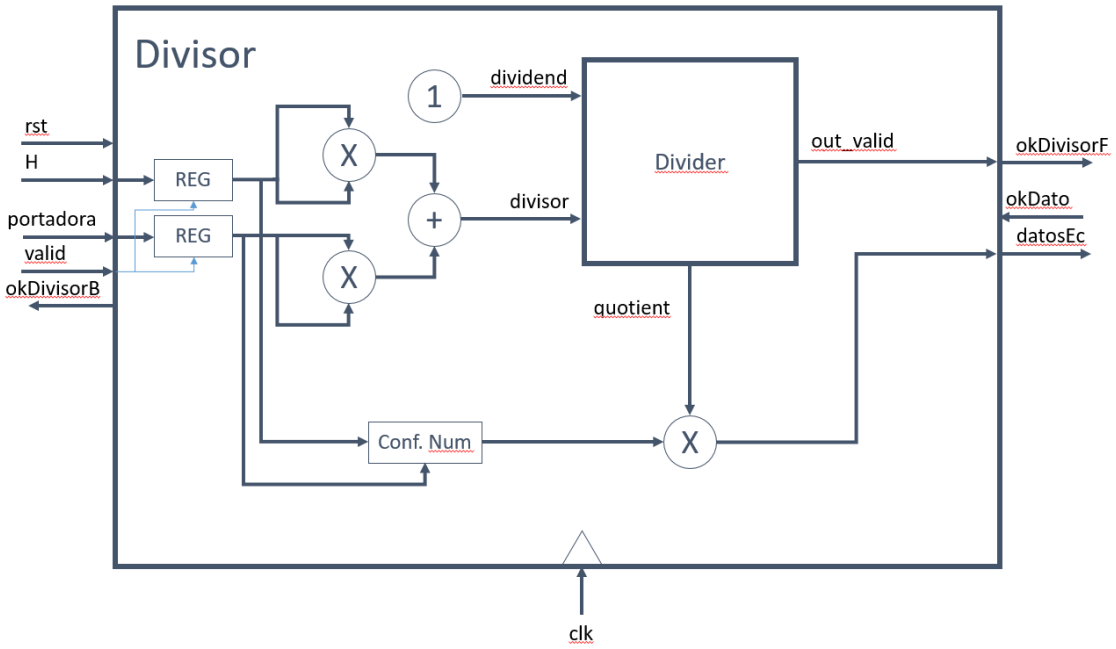



*Figura 8: Funcionamiento simplificado del bloque divisor.*

## top2.vhd

Este fichero contiene el bloque 'top2' que ensamblará los bloques anteriores, implementando los contadores y máquinas de estado necesarios para ello. En la Figura 9 se pueden ver de forma simplificada las conexiones entre los bloques.

El bloque tendrá una máquina de estados de cinco estados. Estos serán: espera, piloto1, piloto2, pilotoNuevo y espera2.

Espera será el estado inicial, se permanece en el hasta que llega un símbolo, señalizándolo mediante la entrada 'datosValid'. De este estado se pasa a piloto1.

El en estado piloto1 se almacenan los datos que se van recibiendo hasta que se llega al primer piloto. Cuando llega el primer piloto, se registra y se pasa al estado piloto2.

En el estado piloto2 se almacenan los datos en la memoria hasta llegar al 2º piloto. Cuando llega el 2º piloto, se registra, se lanza en interpolador y se pasa al estado pilotoNuevo.

En el estado pilotoNuevo se almacenan en la memoria los datos que llegan, lanzando el interpolador cada vez que se alcanza un nuevo piloto. Cuándo se llega al último piloto del símbolo, se registra, se lanza el interpolador y se pasa al estado espera2.

En el estado espera2 se permanece hasta que llega la última muestra del símbolo. Cuando se llega a este punto, se pasa al estado 'espera'.

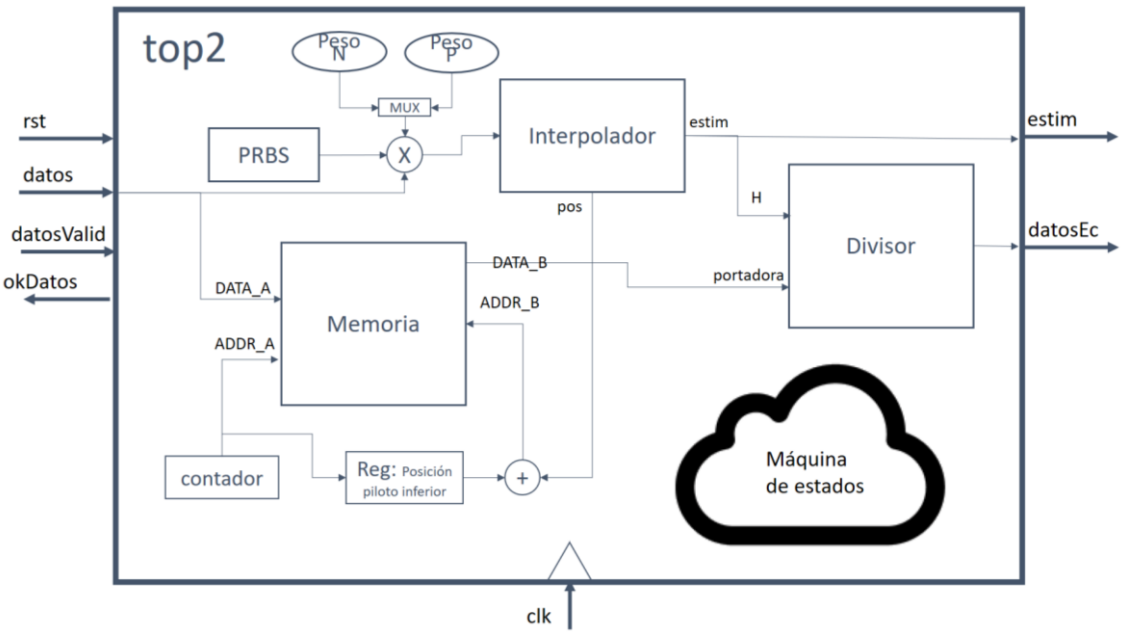



*Figura 9: Diagrama del ensamblado de los bloques simplificado.*

## tb_top.vhd

Este fichero contiene el test bench del sistema. Instancia un componente top2 y le suministra las entradas apropiadas a partir de los ficheros generados con MatLab. Además, registra las salidas, tanto la estimación del canal como los datos ecualizados. Estas salidas las vuelca en cuatro ficheros. Cada fichero contiene una de las partes (real e imaginaria) de una de las salidas (estimación y ecualización.

# 1. Verificación del funcionamiento

Para verificar el funcionamiento del sistema se comparan los resultados obtenidos en MatLab con los de las simulaciones del código VHDL. Para ello se han escrito los ficheros 'test2.m' y 'test3.m'. El primero comprueba la adecuación del canal estimado mediante el sistema desarrollado en VHDL al estimado en MatLab. El segundo recupera los bits a desde los datos ecualizados por el sistema VHDL y computa la BER comparándolos con los 'enviados'.

Mediante la ejecución de estos ficheros se han confeccionado las tablas siguientes.

*Tabla 1: Error absoluto en la estimación del canal (P1) en los primeros 10 símbolos según el tipo de símbolo empleado. Obtenido mediante 'test2.m'.*

| | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2k | 0.011 | 0.011 | 0.011 | 0.011 | 0.011 | 0.011 | 0.011 | 0.011 | 0.011 | 0.011 |
| 8k | 0.011 | 0.011 | 0.011 | 0.011 | 0.011 | 0.011 | 0.011 | 0.011 | 0.011 | 0.011 |

*Tabla 2: Error relativo en la estimación del canal en VHDL con respecto a la de MatLab en los primeros 10 símbolos. Es el valor rms del error sobre el del canal. Obtenido mediante 'test2.m'.*

| (%) | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2k | 0.3674 | 0.3671 | 0.3659 | 0.3821 | 0.3785 | 0.3615 | 0.3623 | 0.3786 | 0.3786 | 0.3701 |
| 8k | 0.3469 | 0.3408 | 0.3426 | 0.3469 | 0.3510 | 0.3327 | 0.3494 | 0.3474 | 0.3450 | 0.3482 |

*Tabla 3: Tasa de error de bit obtenida al ecualizar mediante el sistema VHDL a lo largo de 10 símbolos, según modulación y tipo de símbolo empleado.*

| BER (10 símbolos) VHDL | 2k | 8k |
|---|---|---|
| QPSK | 0.019622 | 0.01553 |
| BPSK | 0.012036 | 0.01142 |

*Tabla 4: Tasa de error de bit obtenida al ecualizar mediante el sistema en MatLab a lo largo de 10 símbolos, según modulación y tipo de símbolo empleado.*

| BER (10 símbolos) VHDL | 2k | 8k |
|---|---|---|
| QPSK | 0.006198 | 0.001448 |
| BPSK | 0.003492 | 0.0008 |

*Tabla 5: BER para una modulación QPSK empleando símbolos 2k, según se emplee el modelo en MatLab o el sistema implementado en VHDL.*

| BER (100 símbolos) | QPSK, 2k |
|---|---|
| VHDL | 0.018956 |
| MatLab | 0.006370 |

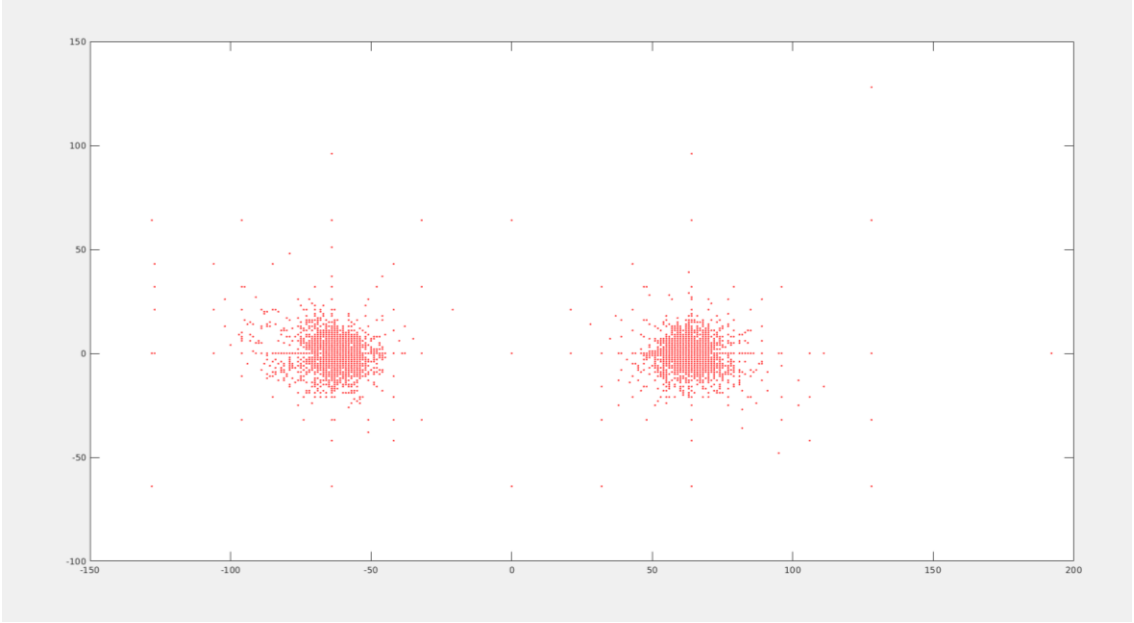



*Figura 10: Constelación BPSK obtenida al ecualizar mediante el sistema implementado en VHDL. Se ha empleado un símbolo 2k.*

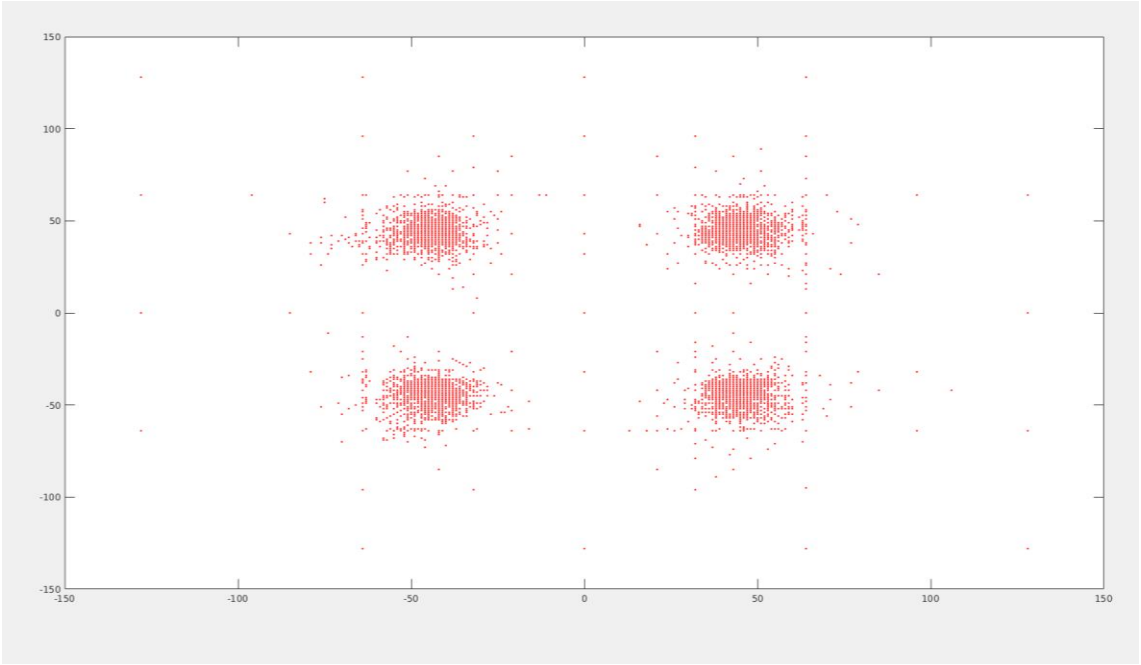



*Figura 11: Constelación QPSK obtenida al ecualizar mediante el sistema implementado en VHDL. Se ha empleado un símbolo 2k.*

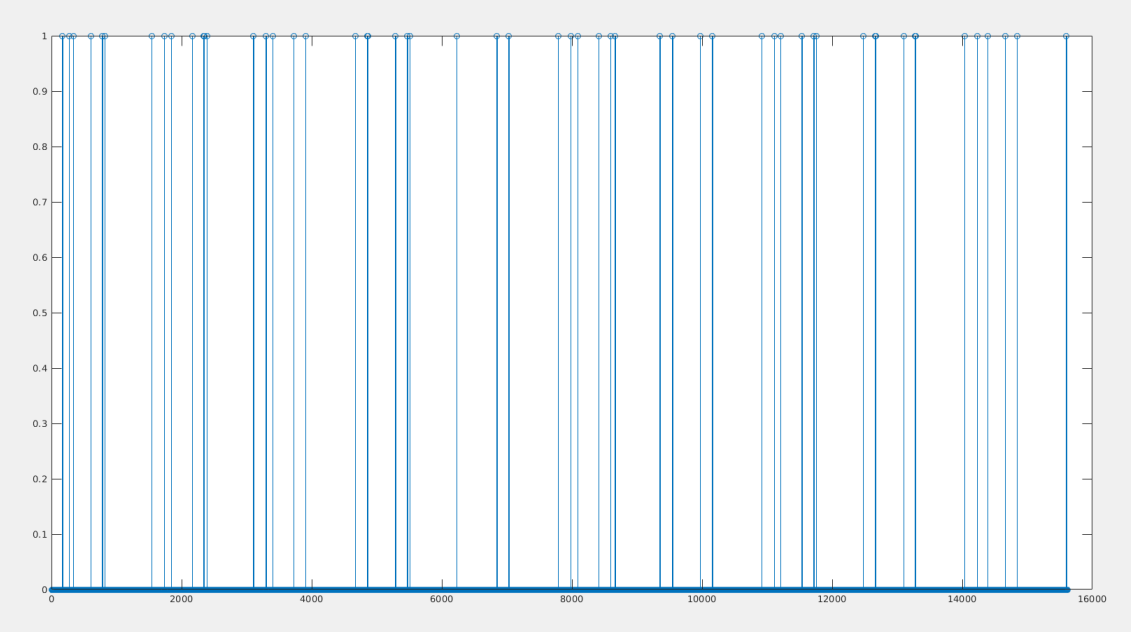



*Figura 12: Bits erróneos en un símbolo 2k empleando una constelación BPSK*

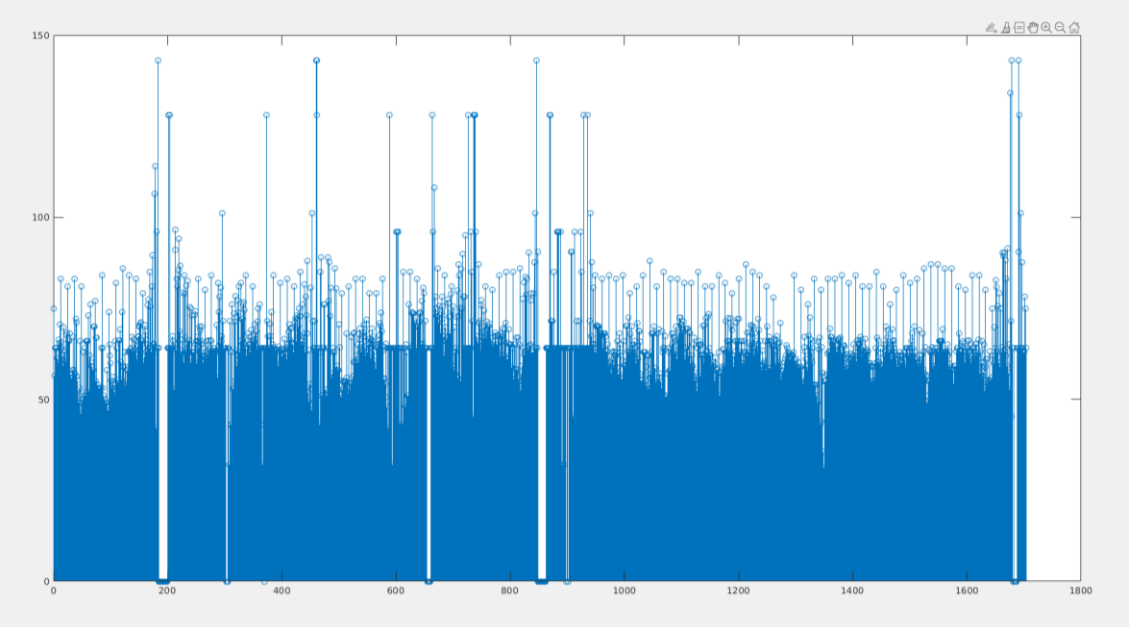



*Figura 13: Símbolo 2k BPSK recibido al ecualizar con el sistema implementado en VHDL. Se puede observar que hay portadoras que se pierden debido a nulos en el canal. se puede comparar con la figura anterior.*