

# Electrónica Digital para Comunicaciones



## PROYECTO DE LA ASIGNATURA.

Estimador de Canal y Ecualizador para DVB-T.

Francisco Javier Ortiz Bonilla

---

### Índice

1. Matlab .....	2
1.1. Parámetros y variables .....	2
1.2. Canal P1 .....	3
1.3. Transmisor .....	4
1.4. Receptor .....	4
1.5. Análisis del código .....	4
1.6. Resultados al ejecutar .....	5
2. VHDL .....	9
2.1. Memoria ROM .....	10
2.2. Interpolador .....	10
2.3. PRBS .....	11
2.4. Divisor .....	12
2.5. FSM .....	13
3. Verificación .....	16
3.1. Script pasarHex.m .....	16
3.2. Script comparador.m .....	16

## 1. Matlab

Para realizar el estimador de canal y el ecualizador en Matlab, se ha partido del código proporcionado para la práctica 1, en el cual venía implementado el transmisor OFDM para modo 2k y modulación BPSK y QPSK.

Además de añadir el canal P1 e implementar el receptor, estimador y ecualizador, se ha añadido lo siguiente:

- Modo 8k.
- Constelaciones 16-QAM y 64-QAM.
- Detector y demapper para dichas constelaciones.

### 1.1. Parámetros y variables

Antes de realizar la transmisión, se da configuración los siguientes parámetros: modo de transmisión, constelación a utilizar, prefijo cíclico, número de portadoras, número de puntos de la FFT...

```
% Parámetros

% Modo 2k ó 8k: {2, 8}.
modo      = 2;
% Constelación: {'BPSK', 'QPSK', '16-QAM', '64-QAM'}
CONSTEL   = 'QPSK';
save('datosMatlab/CONSTEL.mat','CONSTEL')
% Prefijo ciclico: {1/4, 1/8, 1/16, 1/32}
CP        = 1/32;
% SNR
SNR        = 30;
% Espaciado de los pilotos
ESPACIADO_PILOTOS = 12;

%% Ajustar variables según modo.

NPORTADORAS = 1705;
if (modo == 8)
    NPORTADORAS = 6817;
end

NFFT       = modo * 1024;
NCP        = NFFT * CP;
NPILOTOS   = ceil(NPORTADORAS/ESPACIADO_PILOTOS);
NDATA      = NPORTADORAS - NPILOTOS;
NUM_SYMB   = 10;
```

Y se programan las diferentes constelaciones:

```
switch CONSTEL
case 'BPSK'
    M=1;
    C=[1 -1];
case 'QPSK'
    C=[1+1i 1-1i -1+1i -1-1i];
```

```

M=2;
case '16-QAM'
M = 4;
S = de2bi(0:2^M-1, M);
C = [3+3i -3+3i 3-3i -3-3i 1+3i -1+3i 1-3i -1-3i
      3+1i -3+1i 3-1i -3-1i 1+1i -1+1i 1-1i -1-1i];
save('datosMatlab/S.mat', 'S');
case '64-QAM'
M = 6;
S = de2bi(0:2^M-1, M);
C = [7+7i -7+7i 7-7i -7-7i 1+7i -1+7i 1-7i -1-7i
      7+1i -7+1i 7-1i -7-1i 1+1i -1+1i 1-1i -1-1i
      5+7i -5+7i 5-7i -5-7i 3+7i -3+7i 3-7i -3-7i
      5+1i -5+1i 5-1i -5-1i 3+1i -3+1i 3-1i -3-1i
      7+5i -7+5i 7-5i -7-5i 1+5i -1+5i 1-5i -1-5i
      7+3i -7+3i 7-3i -7-3i 1+3i -1+3i 1-3i -1-3i
      5+5i -5+5i 5-5i -5-5i 3+5i -3+5i 3-5i -3-5i
      5+3i -5+3i 5-3i -5-3i 3+3i -3+3i 3-3i -3-3i];
save('datosMatlab/S.mat', 'S');
end
% Energía promedio = 1
C = C./(mean(abs(C)) );

```

Las constelaciones se han programado de acuerdo al estándar (16-QAM Y 64-QAM uniformes) y dividiéndolas por la energía promedio, de forma que esta quede igual a 1.

## 1.2. Canal P1

Se programa el canal P1 siguiendo la descripción que viene en el estándar (Anexo B). A continuación se muestra la gráfica obtenida en matlab del canal P1 en frecuencia:

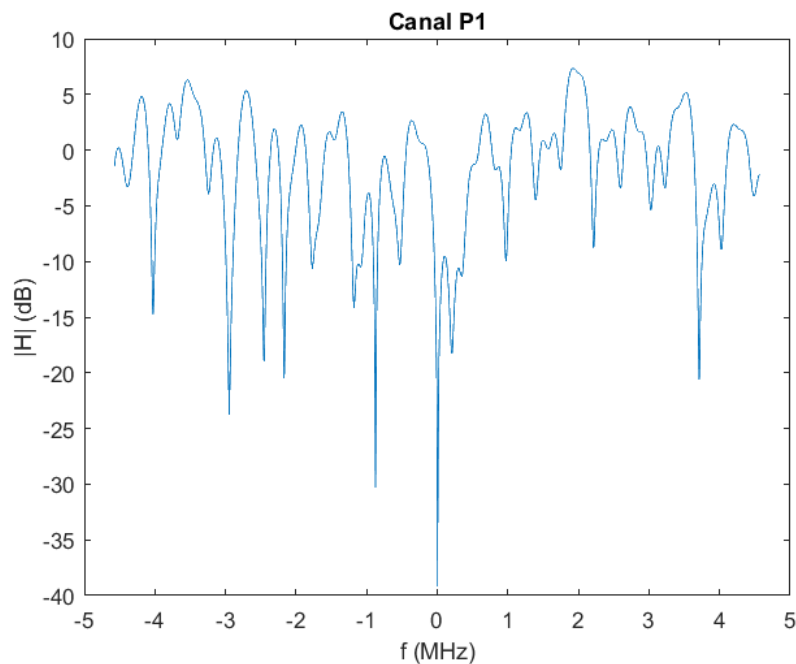


Ilustración 1: Canal P1 real.

### 1.3. Transmisor

Para realizar el transmisor OFDM hay que seguir los siguientes pasos:

- 1) Se generan símbolos a transmitir: Pseudoaleatorios (además, simulamos siempre con la misma semilla). Dependen de la modulación y el modo elegido.
- 2) Se generan los pilotos scattered: A partir de una secuencia pseudoaleatoria (PRBS, apartado 4.5.2 del estándar) que determina el signo de cada uno.
- 3) Se combinan los símbolos generados y los pilotos, y se añaden ceros al principio y al final para completar las 2048 portadoras.
- 4) Se añade el prefijo cíclico.
- 5) La información que acabamos de generar está en frecuencia, así que hacemos la IDFT para pasarla al dominio temporal y poder transmitirla.
- 6) Una vez se tiene la información en el dominio temporal, se convoluciona con el canal P1 y se añade ruido AWGN, de acuerdo a la SNR especificada anteriormente. Estos serán los datos de entrada al receptor.

### 1.4. Receptor

Para obtener la información transmitida, en el receptor:

- 1) Quitamos el prefijo cíclico.
- 2) Hacemos la DFT de lo que recibimos (los datos están en frecuencia, en cada portadora).
- 3) En este momento disponemos de los datos recibidos (datos y pilotos) y podemos representar la constelación recibida, pero dará un resultado muy malo debido al efecto del canal P1 y el ruido. Esto se arregla ecualizando, así que usamos los pilotos para estimar el canal, realizando una interpolación lineal.
- 4) Una vez se ha estimado el canal para cada símbolo enviado, se ecualiza dividiendo los datos recibidos por el canal estimado. Una vez hemos ecualizado, podemos representar la constelación recibida.
- 5) Hacemos el demap: Por cada valor de cada portadora, obtenemos los bits transmitidos usando un detector ML (comparamos el valor de la portadora con los posibles símbolos transmitidos y elegimos aquel cuya distancia sea menor).
- 6) Comparamos los bits recibidos con los transmitidos y calculamos la BER.

### 1.5. Análisis del código

El código no se ha hecho por funciones, está todo en el mismo fichero (**fcoortbon.m**). Adicionalmente se ha hecho otro script, **pasarHex.m**, que es llamado al final de fcoortbon.m, y que sirve para pasar a hexadecimal los datos que se han ido guardando durante la ejecución y para generar un fichero .coe con dichos datos.

El código se ha optimizado lo máximo posible. Sólo se usan los siguientes bucles for:

- Bucle for para calcular el PRBS:

```
for k = 1:NPORTADORAS
    prbs(k) = sec(end);
```

```

p = xor(sec(1,end), sec(1,end-2));
sec(2:end) = sec(1:end-1);
sec(1) = p;
end

```

- Bucle for para programar el canal P1:

```

for k = 1:20
    H = H + p(k)*exp(-1i*theta(k))*exp(-1i*w*tau(k)*10^-6);
end

```

- Tres bucle for anidados para realizar la interpolación lineal y estimar el canal:

```

for s = 1:NUM_SYMB
    for n = 1:NPILOTOS
        p = pos_pilotos(n);
        if (n == NPILOTOS)
            % Si hay más portadoras tras el último piloto...
            portadoras = size(He,1)-p;
            if (portadoras)
                % ... interpolo con la pendiente de la recta anterior.
                a = He(p,s);
                for dp = 1:portadoras
                    He(p+dp,s) = a + m*dp;
                end
            end
        else
            a = He(p,s);
            b = He(p+ESPACIADO_PILOTOS,s);
            m = (b-a)/ESPACIADO_PILOTOS;
            for dp = 1:ESPACIADO_PILOTOS-1
                He(p+dp,s) = a + m*dp;
            end
        end
    end
end
end

```

- Bucle for para detectar los símbolos transmitidos a partir del valor de cada portadora en 16-QAM y 64-QAM:

```

for d = 1:length(rx_datos)
    % Detectamos el símbolo
    [~, ind] = min((C-rx_datos(d)).^2);
    bits_rx(1, M*(d-1)+1:M*d) = S(ind,:);
end

```

En mi ordenador, el tiempo que tarda en hacer una simulación con modo 2k y constelación BPSK es 2.4 segundos. El tiempo con modo 8k y constelación 64-QAM es 3.7 segundos.

## 1.6. Resultados al ejecutar

Si ejecutamos una simulación usando modo 8k y una constelación 64-QAM, obtenemos los siguientes resultados:

```

>> fcoortbon
Modo: 8k · Constel: 64-QAM · SNR: 40.0dB
BER = 0.008496

```

Elapsed time is 3.683635 seconds.

Constelación transmitida

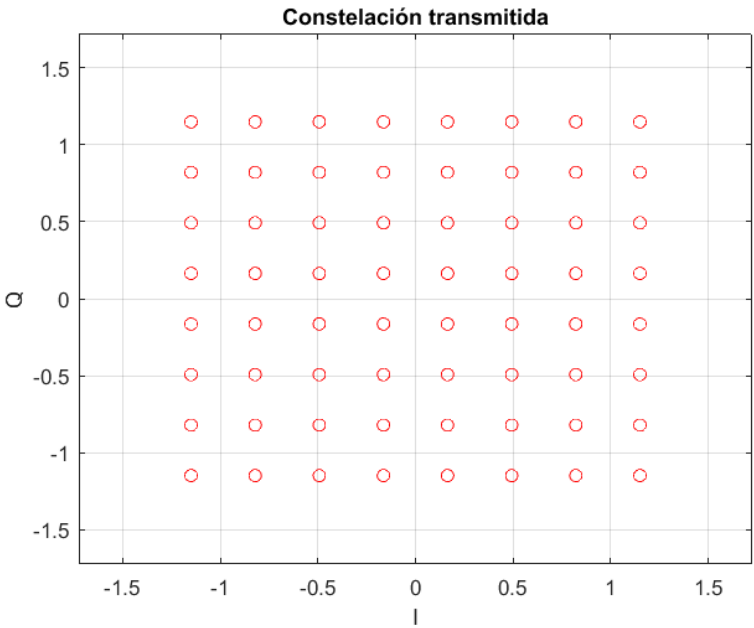


Ilustración 2: Constelación 64-QAM.

Espectro OFDM enviado

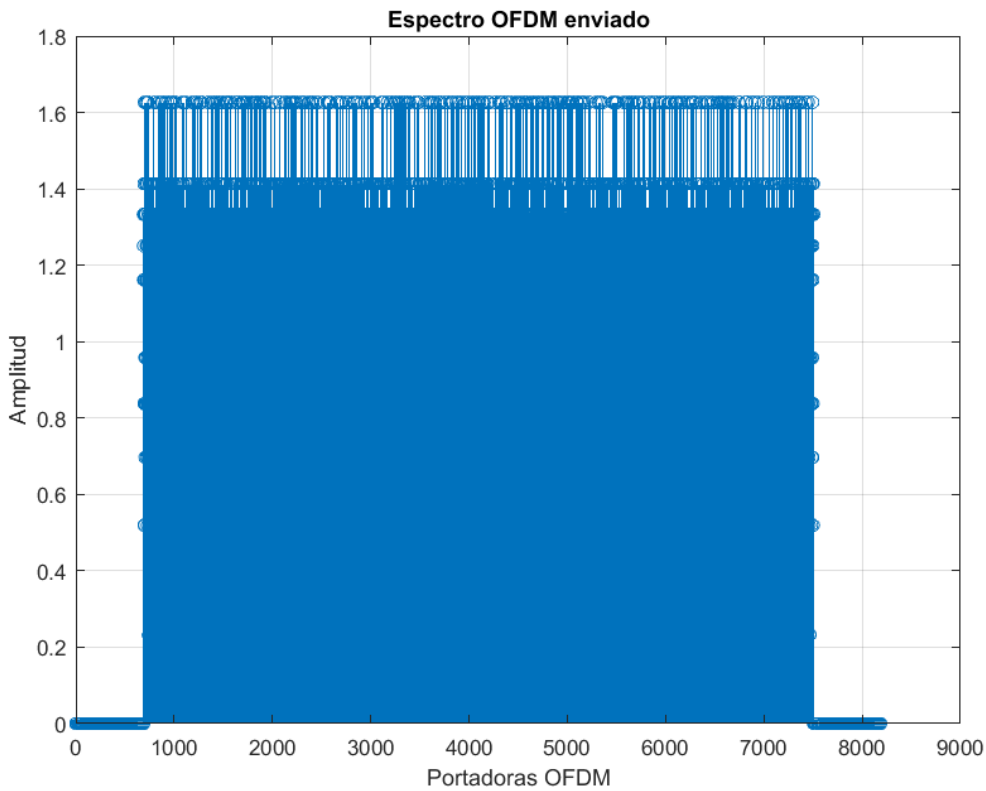


Ilustración 3: Espectro OFDM enviado. Modo 8k y constelación 64-QAM.

**Espectro OFDM recibido**

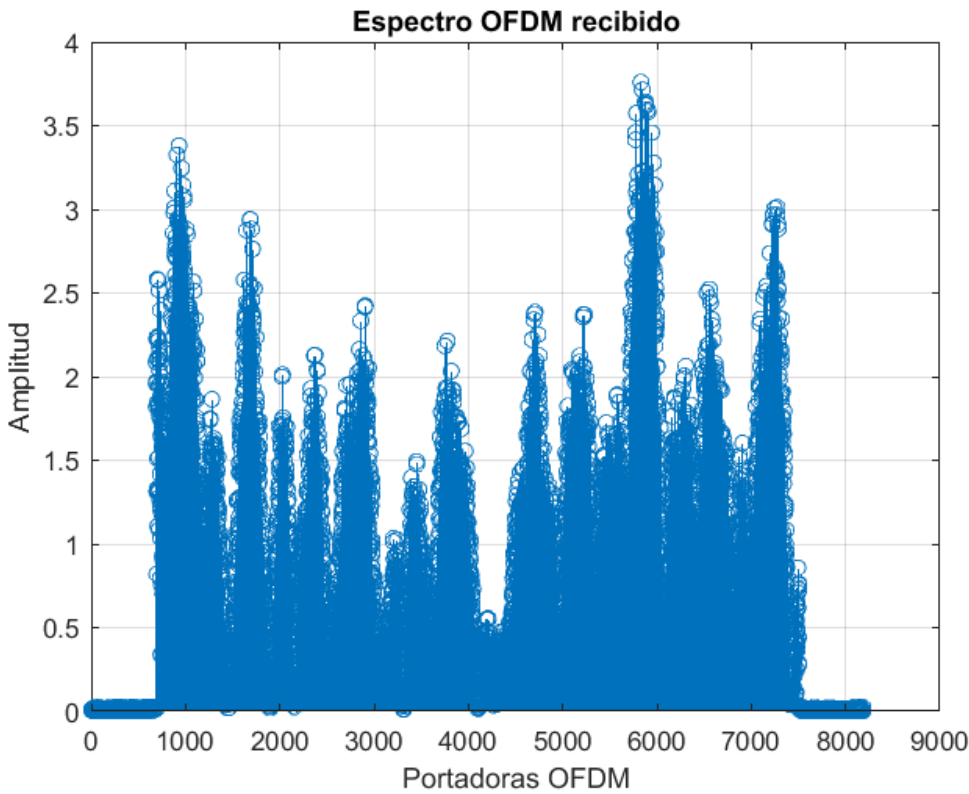


Ilustración 4: Espectro OFDM enviado. Modo 8k y constelación 64-QAM.

**Constelación recibida antes de ecualizar**

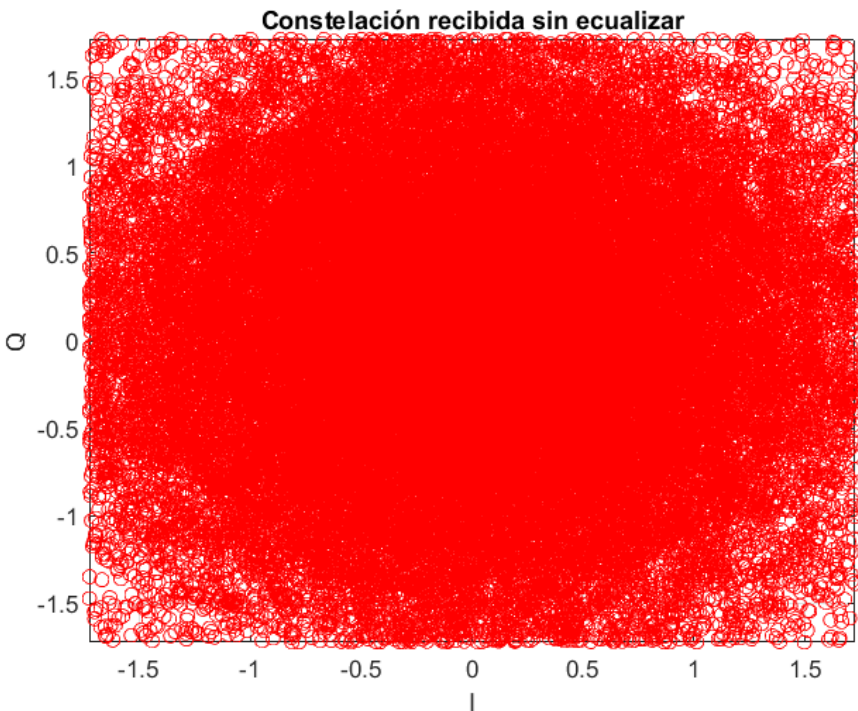


Ilustración 5: Constelación recibida antes de ecualizar. Modo 8k y constelación 64-QAM.

**Canal P1 real y estimado**

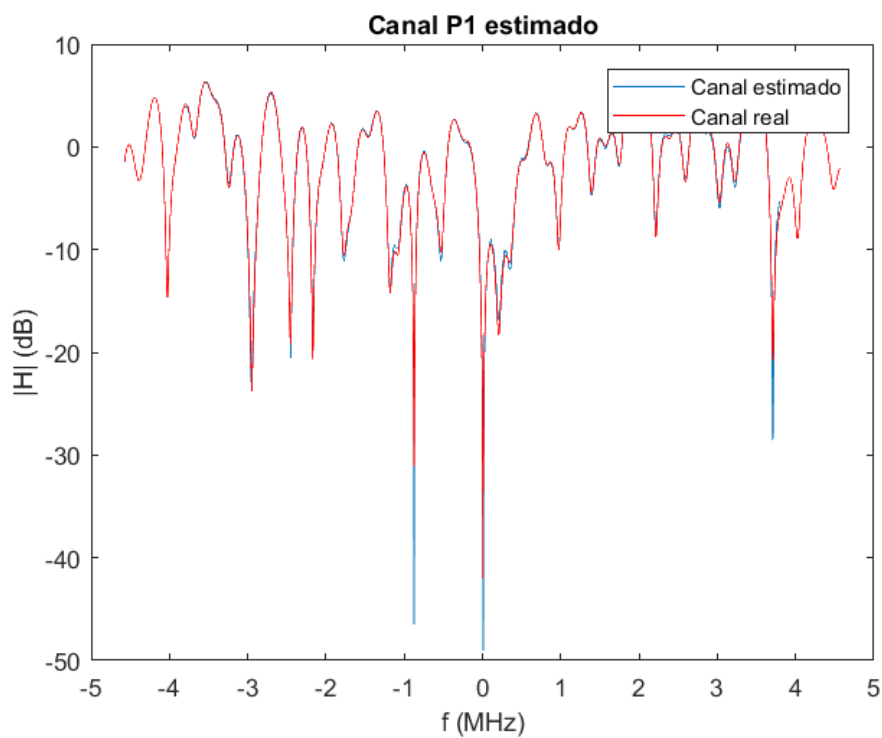


Ilustración 6: Comparación canal P1 real y estimado. Modo 8k.

**Constelación recibida tras ecualizar**

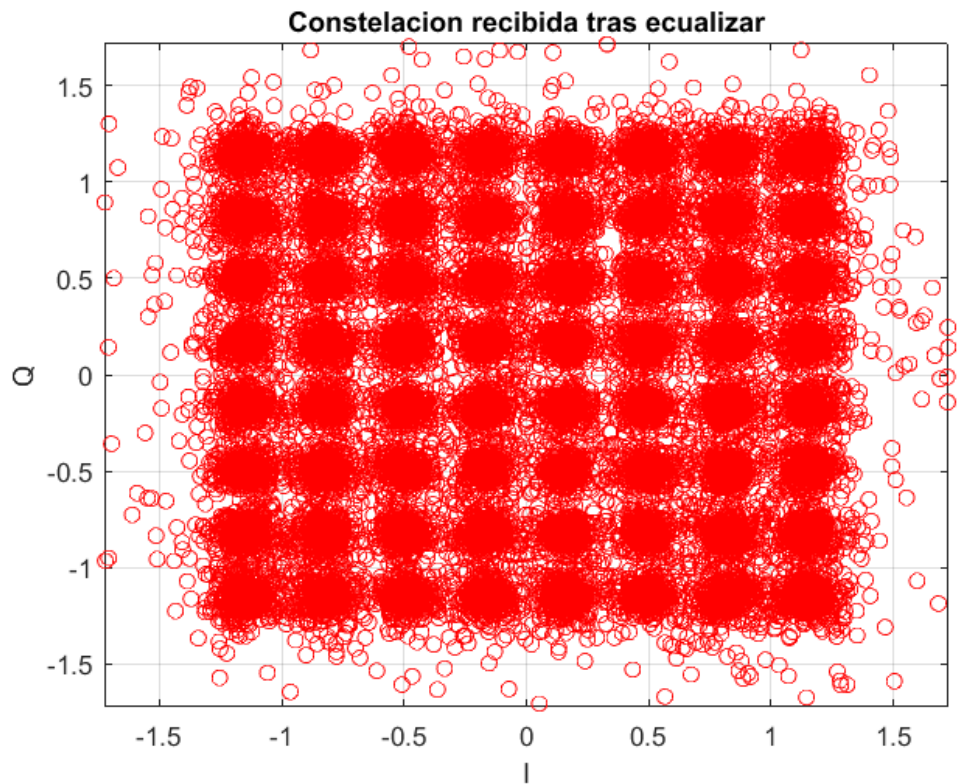


Ilustración 7: Constelación recibida tras ecualizar. Modo 8k y constelación 64-QAM.



## 2. VHDL

En VHDL se ha diseñado el siguiente sistema:

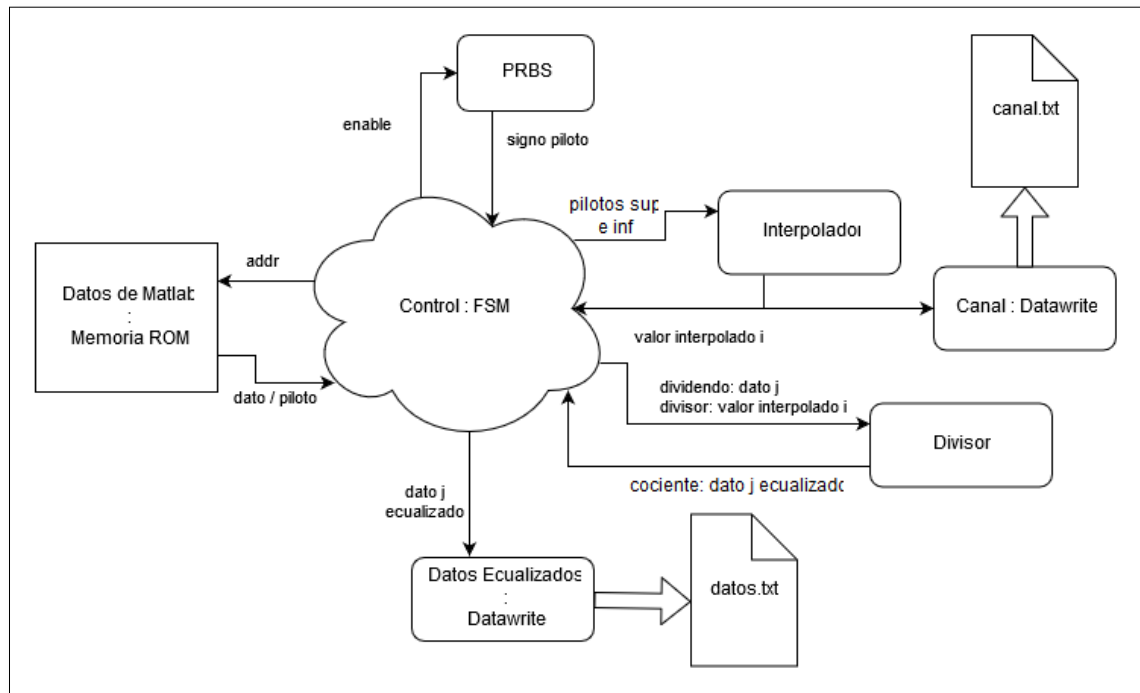


Ilustración 8: Diagrama de bloques del circuito diseñado en VHDL.

El bloque FSM contiene la máquina de estados que gobierna el funcionamiento del ecualizador. Va leyendo de la memoria ROM los pilotos (posiciones de memoria 0, 12, 24, 36...) y va pasando parejas de pilotos inferior y superior al interpolador. Cada vez que se interpola un valor (de los 11 que hay entre piloto y piloto), el interpolador queda a la espera y el bloque FSM realiza la ecualización del dato correspondiente, el cual lee de la memoria ROM y lo pasa al divisor para que lo divida entre el valor que se acaba de interpolar (se realizan dos divisiones, una para la parte real y otra para la parte imaginaria). Tras hacer estas divisiones y obtener el dato ecualizado, se da como salida del bloque, poniéndose `dato_valid = 1` durante un ciclo de reloj, y vuelve a activarse el interpolador para que interpole el valor siguiente. Este proceso se repite 11 veces, tras las cuales vuelve a leerse de la memoria un nuevo piloto, que pasa a ser el piloto superior, y el piloto superior anterior es ahora el inferior, y se repite el proceso anterior.

Además, las salidas del bloque interpolador `estim` y `estim_valid` son entradas de un bloque Datawrite, que irá escribiendo cada valor estimado válido en hexadecimal en un fichero de texto, `canal.txt`, el cuál luego usaremos para comparar con el canal estimado en matlab.

Lo mismo se hace con las salidas `dato_ecualizado` y `dato_valid`, de forma que en el fichero `datos.txt` se hallan los datos ecualizados en VHDL.

## 2.1. Memoria ROM

La memoria ROM se ha hecho pensando en el modo 2k, por lo que tiene una longitud de 1705, y cada dato es de 24 bits, de los cuales los 12 primeros corresponden a la parte real y los 12 últimos a la imaginaria. Los dos bits más significativos de ambas partes sobran, ya que ambas tienen longitud de 10 bits, pero en al representarlos en hexadecimal se usan 12.

La memoria se llena a partir de un fichero .coe que se genera en la carpeta **datosMatlab** tras realizar una simulación en matlab (usando el script **fcoortbon.m**). Y los datos son los datos recibidos para el primer símbolo OFDM transmitido sin haber sido ecualizados (tanto datos como pilotos).

## 2.2. Interpolador

Este bloque consta de los siguientes puertos:

Puerto	Dirección	Tipo de Dato	Descripción
clk	in	std_logic	Señal de reloj. Activa por flanco de subida
rst	in	std_logic	Reset global asíncrono. Activo a nivel alto.
inf	in	complex10	Piloto inferior.
sup	in	complex10	Piloto superior.
enable	in	std_logic	Señal que habilita el interpolador.
estim	out	complex10	Valor interpolado entre sup e inf.
estim_valid	out	std_logic	Señal que indica que la salida estim es válida.

Además dispone de un generic, `ESPACIADO_PILOTOS`, que indicará el número de interpolaciones que se hacen entre sup e inf, el cual es el valor mismo de este generic, puesto que además de interpolarse los valores intermedios, también se devuelve el primer piloto (el inferior).

El siguiente diagrama de bolas ilustra el funcionamiento de este bloque.

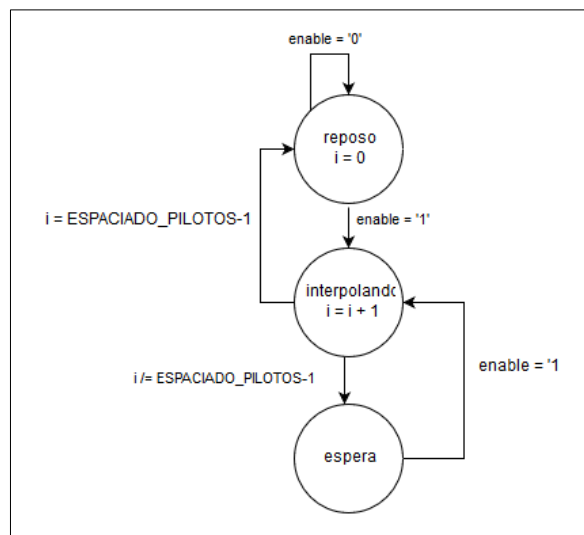
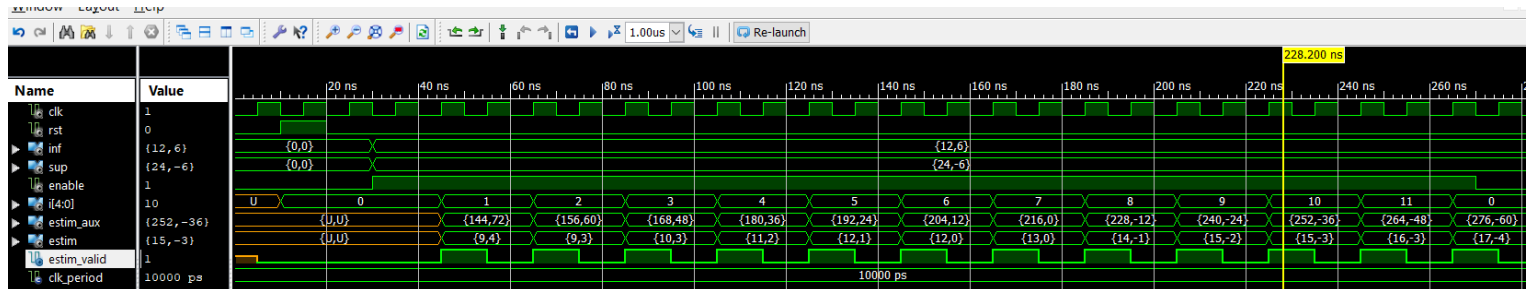


Ilustración 9: Diagrama de bolas del bloque Interpolador.

En el estado interpolando sólo se mantiene un ciclo de reloj, y asigna el nuevo valor interpolado a la salida `estim`, poniendo durante un ciclo de reloj `estim_valid = 1`.

Este bloque se probó de forma aislada mediante un test bench que le pasaba como entradas `inf` y `sup` un par de valores imaginarios sencillos ( $12 + 6j$  y  $24 - 6j$  respectivamente). Podemos observar en las formas de onda que el bloque funciona tal y como se espera.



En la señal `estim_aux` observamos cada valor interpolado multiplicado por 12:

- $i = 0: 12 * (12 + 6i) = 144 + 72i$
- $i = 1: 12 * (13 + 5i) = 156 + 60i$
- ...
- $i = 11: 12 * (23 - 5i) = 276 - 60i$

Y la señal `estim` es el valor de la estimación tras haber sido truncada (eliminando los 4 bits menos significativos). Tenemos que tener en cuenta que el valor de los pilotos es  $\pm 4/3$ . Eso junto a que tenemos la interpolación multiplicada por 12 nos da un factor de escala de 16, que precisamente podemos arreglar quitando los 4 últimos bits: esto es equivalente a dividir por 16, aunque perdemos el resto, y, por lo tanto, precisión.

### 2.3. PRBS

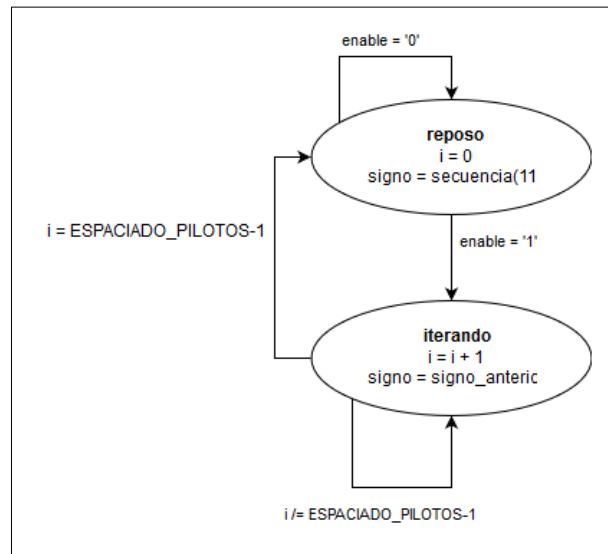
El bloque PRBS consiste en un registro de desplazamiento. Se usa para averiguar el signo de los pilotos. Como 1 de cada `ESPACIADO_PILOTOS` datos es un piloto, es necesario desplazar el registro `ESPACIADO_PILOTOS` veces para ir obteniendo el signo de cada piloto.

Al igual que en el bloque interpolador, este bloque también dispone del generico `ESPACIADO_PILOTOS`. Los puertos de este bloque son los siguientes:

Puerto	Dirección	Tipo de Dato	Descripción
clk	in	std_logic	Señal de reloj. Activa por flanco de subida
rst	in	std_logic	Reset global asíncrono. Activo a nivel alto.
enable	in	std_logic	Señal que que habilita el registro para que este se desplace <code>ESPACIADO_PILOTO</code> veces.
ready	out	complex10	Señal que indica que el registro no está desplazándose actualmente. Además, esto quiere decir que el signo que se da como salida es bueno.
signo	out	std_logic	Señal que indica el signo del piloto: 1 (negativo), 0 (positivo).

Este bloque funciona de forma que la salida `signo` es siempre el signo del piloto de interés, y cuando este es leído, aquel bloque que lo lee, si pretende en el futuro consultar el signo del piloto siguiente, debe mandar un pulso con `enable = 1` para que el registro comience a desplazarse de forma que se prepare el signo del próximo piloto.

El siguiente diagrama de bolas ilustra la lógica que se acaba de comentar.



**Ilustración 10: Diagrama de bolas del bloque PRBS.**

## 2.4. Divisor

El divisor es un bloque Divider Generator 3.0 generado mediante la herramienta Core enerator de ISE de la siguiente forma:

**Ilustración 11: Generación del bloque divisor.**

En teoría, dividiendo y divisor son números de 21 bits (suma de dos números de 20 bits que son el producto de 2 números de 10 bits), sin embargo si intento utilizar un divisor de 21 bits, ISE da un error diciendo que la expresión es de 20 bits y que se esperaban 21, así que he optado por usar 20 bits.

```

307      p_dividendo_aux <= std_logic_vector(signed(dato_re)*signed(h_re)+signed(dato_im)*signed(h_im));
308      n_divisor_aux   <= std_logic_vector(signed(h_re)*signed(h_re)+signed(h_im)*signed(h_im));

```

Design Summary (Synthesized) | Ecuizador.vhd | FSM.vhd

'C:\Users\Javi\Desktop\Ecuizador\FSM.vhd' Line 307: Expression has 20 elements ; expected 21

Ilustración 12: Error al utilizar divisor de 21 bits.

El bloque consta de los siguientes puertos:

Puerto	Dirección	Tipo de Dato	Descripción
clk	in	std_logic	Señal de reloj. Activa por flanco de subida
nd	in	std_logic	Indica al divisor que hay datos nuevos para dividir.
rdy	out	std_logic	El divisor indica que ha terminado la división y que las salidas quotient y fractional son correctas.
rfd	out	std_logic	El divisor indica que está listo para recibir datos nuevos para una nueva división.
dividend	in	std_logic_vector(20 downto 0)	Dividendo.
divisor	in	std_logic_vector(20 downto 0)	Divisor.
quotient	out	std_logic_vector(20 downto 0)	Cociente resultante de la división.
fractional	out	std_logic_vector(20 downto 0)	Resto resultante de la división.

El funcionamiento de este bloque es simple, avisamos al divisor de que queremos que realice una nueva división poniendo `nd = 1` y el dividendo y divisor en `dividend` y `divisor` respectivamente, y esperamos hasta que haya acabado, cuando `rdy = 1`. Antes de darle nuevos datos, debemos cerciorarnos de que `rdy = 1`, o el divisor nos ignorará.

### 2.5. FSM

Como se comentó anteriormente, este bloque contiene la máquina de estados que gobierna el funcionamiento del ecualizador. Lee parejas de pilotos de la memoria, obtiene valor a valor el canal estimado a través del interpolador, y cada vez que obtiene un valor de canal estimado ecualiza el dato correspondiente utilizando el divisor.

Este bloque consta de los siguientes puertos:

Puerto	Direcc.	Tipo de Dato	Descripción
clk	in	std_logic	Señal de reloj. Activa por flanco de subida
rst	in	std_logic	Indica al divisor que hay datos nuevos para dividir.
start	in	std_logic	Señal que indica al bloque que comience a funcionar.
datoMemoria	in	std_logic_vector(23 downto 0)	Puerto por donde se reciben los datos de la memoria ROM.
signo	in	std_logic	Puerto por donde se recibe el signo proporcionado por el PRBS.
prbs_ok	in	std_logic	Puerto que indica que el bloque PRBS está listo y que su salida es válida.
int_valid	in	std_logic	Puerto que indica que la salida del interpolador es válida.
int_dato	in	complex10	Puerto por donde se reciben los valores estimados por el interpolador.
div_rfd	in	std_logic	Puerto por donde se recibe la señal rfd del divisor.
div_rdy	in	std_logic	Puerto por donde se recibe la señal rdy del divisor.
div_q	in	std_logic_vector(19 downto 0)	Puerto por donde se recibe la señal quotient del divisor.
div_f	in	std_logic_vector(19 downto 0)	Puerto por donde se recibe la señal fractional del divisor.
div_nd	out	std_logic	Puerto a través del cual se asigna la entrada nd del divisor.
div_dividend	out	std_logic_vector(19 downto 0)	Puerto a través del cual se asigna la entrada dividend del divisor.
div_divisor	out	std_logic_vector(19 downto 0)	Puerto a través del cual se asigna la entrada divisor del divisor.
enablePRBS	out	std_logic	Puerto a través del cual se asigna la entrada enable del PRBS.
dir	out	std_logic_vector(10 downto 0)	Puerto a través del cual se asigna la dirección de la memoria ROM.
interpola	out	std_logic	Puerto a través del cual se asigna la entrada enable del bloque interpolador.
piloto_sup	out	complex10	Puerto a través del cual se asigna la entrada sup del bloque interpolador.
piloto_inf	out	complex10	Puerto a través del cual se asigna la entrada inf del bloque interpolador.
dato_valid	out	std_logic	Salida que indica que el valor de la salida dato_ecualizado es bueno.
dato_ecualizado	out	std_logic_vector(15 downto 0)	Dato ecualizado.

A continuación se muestra un diagrama de bolas que ilustra el funcionamiento de este bloque.

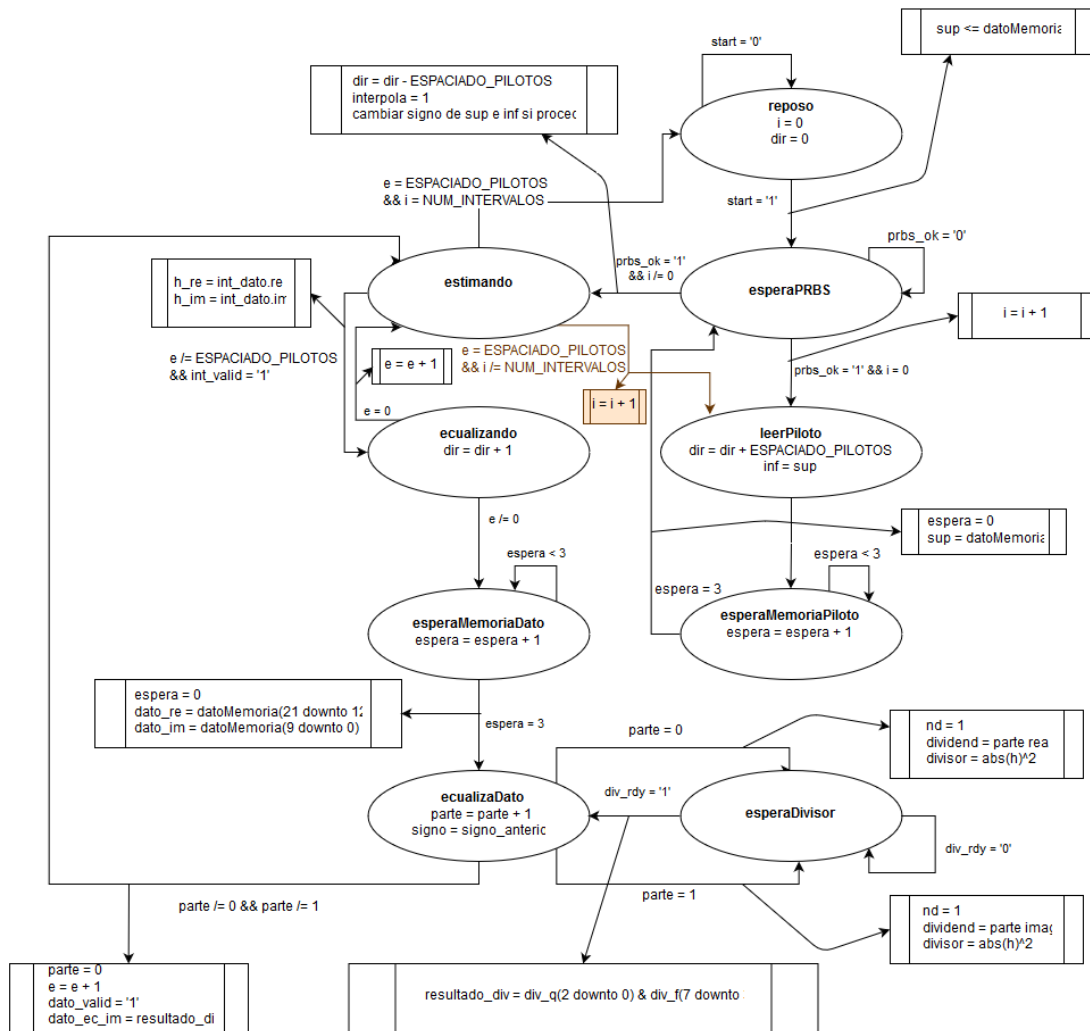


Ilustración 13: Diagrama de bolas del bloque FSM.

Destacar que en el estado `ecualizaDato`, la variable `parte` indica qué división se va a realizar o si ya se han realizado ambas divisiones. Se hace una división que corresponde a la parte real del dato ecualizado y otra que corresponde a la parte imaginaria.

La variable '`e`' corresponde al índice del dato que se está ecualizando entre piloto y piloto, y va de 0 a 11 (cuando es 0 no se ecualiza porque el resultado de la interpolación es el piloto inferior).

La variable '`i`' corresponde al trozo que se está interpolando y estimando, como en modo 2k hay 143 pilotos, habría 142 trozos. Este es el valor por defecto del generico `NUM_TROZOS`.

### 3. Verificación

Para verificar el correcto funcionamiento del circuito, se han creado scripts de matlab que procesan los datos para pasarlos a hexadecimal y viceversa, pudiendo comparar las salidas de ambas soluciones cuando los datos de entrada son los mismos.

A partir de una ejecución en matlab, como se comentó anteriormente, se genera un fichero .coe con los datos recibidos en las 1705 (datos y pilotos). Este fichero .coe se usa posteriormente al simular el circuito VHDL, siendo el contenido de la memoria ROM.

También, como se ha comentado más arriba, en la simulación existen dos bloques datawrite que guardan el canal estimado y los datos ecualizados en hexadecimal en ficheros .txt.

#### 3.1. Script pasarHex.m

Es llamado desde el script de simulación en matlab. Crear ficheros con los datos recibidos en hexadecimal, el más importante es el .coe que servirá para crear la ROM en VHDL.

Este script espera que los datos de matlab estén en el directorio 'datosMatlab', y los carga con la orden load. El fichero de simulación fcoortbon.m es el que guarda los datos en ese directorio.

#### 3.2. Script comparador.m

Este script procesa los datos en hexadecimal resultantes de la simulación en VHDL (ficheros canal.txt y datos.txt), pasándolos a decimal y deshaciendo el factor de escala realizado previamente.

Este script espera que los ficheros resultantes de la simulación en VHDL, canal.txt y datos.txt, estén en el directorio 'datosVHDL', y los abre con fopen. Al simular con VHDL, los ficheros quedan guardados en la carpeta output del proyecto, así que es necesario copiarlos a la carpeta 'datosVHDL' que se encuentra dentro del directorio 'Matlab'.

Una vez están los datos en el formato correcto, se comparan ambas estimaciones del canal, las constelaciones ecualizadas y la BER.

Se ha simulado usando modo 2k, constelación QPSK y SNR = 30dB y se han obtenido los siguientes resultados:

```
>> comparador
Error entre estimaciones del canal de 0.719%
BER Matlab = 0.004994
BER VHDL = 0.481754
```



## Comparación de las estimaciones del canal

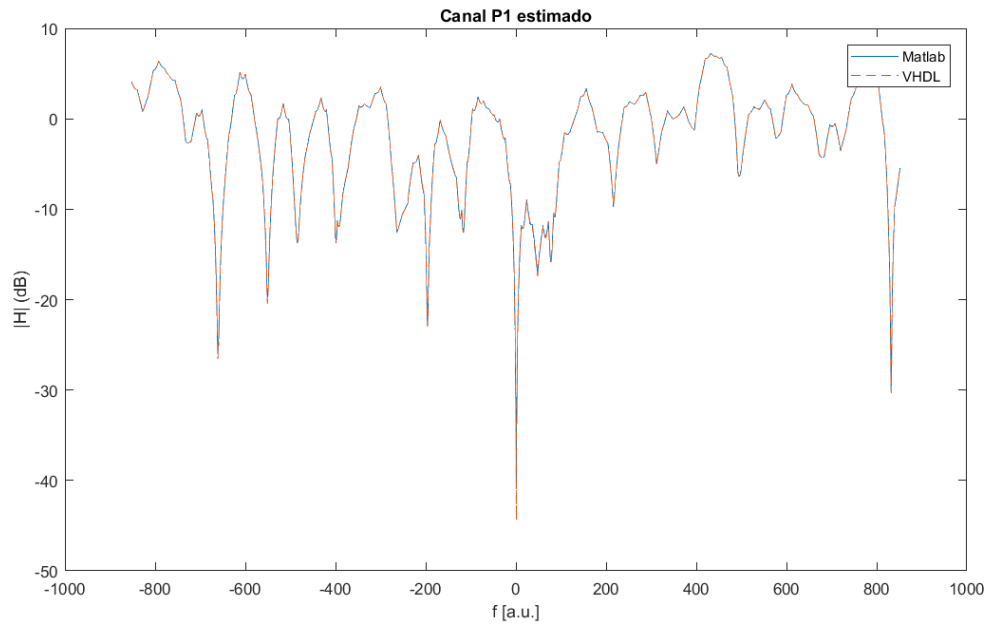


Ilustración 14: Comparación de las estimaciones del canal en Matlab y VHDL.

Podemos observar que las estimaciones son prácticamente idénticas.

## Comparación de las constelaciones obtenidas tras ecualizar

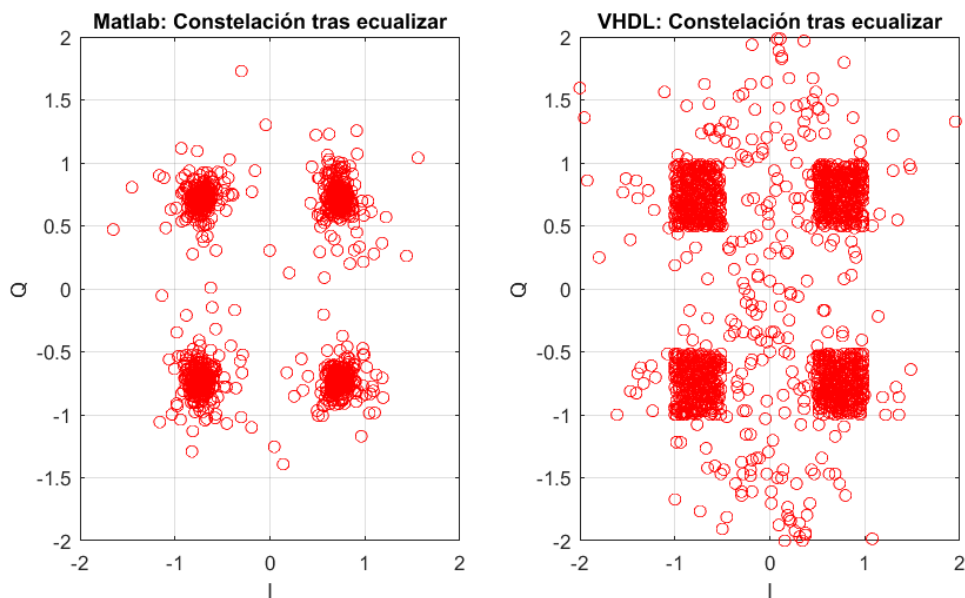


Ilustración 15: Comparación de las constelaciones obtenidas tras ecualizar en Matlab y VHDL.

Observamos que la constelación obtenida en VHDL es buena, sin embargo el resultado obtenido de la BER es desastroso. Si comparamos los datos ecualizados en matlab y en VHDL observamos lo siguiente:

### Matlab

```
>> datos_ec(1:10)

ans =

    0.7144 + 0.7163i
   -0.7292 + 0.7367i
   -0.7005 + 0.7548i
   -0.7343 - 0.7160i
    0.7285 - 0.7673i
    0.7675 + 0.7107i
   -0.7361 + 0.7694i
    0.7680 + 0.7347i
   -0.7587 - 0.7397i
    0.7094 - 0.7350i
```

### VHDL

```
>> datos(1:10)

ans =


   -0.8281 - 0.5156i
   -0.6250 - 0.9531i
   -0.6563 + 0.5781i
    0.6094 + 0.5313i
    0.6094 - 0.5156i
   -1.0000 - 0.6563i
    0.7813 - 0.7813i
   -0.5469 + 0.5625i
    0.8906 + 0.5313i
   -0.5938 - 0.5938i
```

Observamos que los datos ecualizados en VHDL tienen una fase aleatoria, lo que causa que la BER sea casi de 0.5. Esto es debido a un error que cometo al hacer las divisiones, pero no lo he encontrado y no he tenido el tiempo suficiente para debugear y arreglarlo.

































### 3.3. Uso de recursos de la FPGA

Ecualizador Project Status			
Project File:	Proyecto.xise	Parser Errors:	No Errors
Module Name:	Ecualizador	Implementation State:	Synthesized
Target Device:	xc6sxb9-3csg324	• Errors:	No Errors
Product Version:	ISE 14.7	• Warnings:	<a href="#">32 Warnings (30 new)</a>
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	<a href="#">Xilinx Default (unlocked)</a>	• Timing Constraints:	
Environment:	<a href="#">System Settings</a>	• Final Timing Score:	












Device Utilization Summary (estimated values)				
Logic Utilization	Used	Available	Utilization	
Number of Slice Registers	1080	11440	9%	
Number of Slice LUTs	993	5720	17%	
Number of fully used LUT-FF pairs	918	1155	79%	
Number of bonded IOBs	21	200	10%	
Number of Block RAM/FIFO	4	32	12%	
Number of BUFG/BUFGCTRLs	1	16	6%	
Number of DSP48A1s	20	16	125%	

Detailed Reports						
Report Name	Status	Generated	Errors	Warnings	Infos	
<a href="#">Synthesis Report</a>	Current	mi. 6. jun. 23:03:20 2018	0	<a href="#">32 Warnings (30 new)</a>	<a href="#">11 Infos (11 new)</a>	
Translation Report						
Map Report						
Place and Route Report						
Power Report						
Post-PAR Static Timing Report						
Bitgen Report						

### 3.4. Lista de warnings

Program	All Implementation Messages - Errors, Warnings, and Infos
xst	 HDLCompiler:89 - "C:\Users\Javi\Dropbox\Master\Electronica\Proyecto\Proyecto\Ecualizador.vhd" Line 46: <datawrite> remains a black-box since it has no binding entity.
xst	 HDLCompiler:89 - "C:\Users\Javi\Dropbox\Master\Electronica\Proyecto\Proyecto\Ecualizador.vhd" Line 70: <divisor20> remains a black-box since it has no binding entity.
xst	 Xst:647 - Input <div_q<19:3>> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.
xst	 Xst:647 - Input <div_f<14:0>> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.
xst	 Xst:2677 - Node <estim_aux_re_0> of sequential type is unconnected in block <interpolador>.
xst	 Xst:2677 - Node <estim_aux_re_1> of sequential type is unconnected in block <interpolador>.
xst	 Xst:2677 - Node <estim_aux_re_2> of sequential type is unconnected in block <interpolador>.
xst	 Xst:2677 - Node <estim_aux_re_3> of sequential type is unconnected in block <interpolador>.
xst	 Xst:2677 - Node <estim_aux_re_14> of sequential type is unconnected in block <interpolador>.
xst	 Xst:2677 - Node <estim_aux_im_0> of sequential type is unconnected in block <interpolador>.
xst	 Xst:2677 - Node <estim_aux_im_1> of sequential type is unconnected in block <interpolador>.
xst	 Xst:2677 - Node <estim_aux_im_2> of sequential type is unconnected in block <interpolador>.
xst	 Xst:2677 - Node <estim_aux_im_3> of sequential type is unconnected in block <interpolador>.
xst	 Xst:2677 - Node <estim_aux_im_14> of sequential type is unconnected in block <interpolador>.
xst	 Xst:2677 - Node <sup_10> of sequential type is unconnected in block <control>.
xst	 Xst:2677 - Node <sup_11> of sequential type is unconnected in block <control>.
xst	 Xst:2677 - Node <sup_22> of sequential type is unconnected in block <control>.
xst	 Xst:2677 - Node <sup_23> of sequential type is unconnected in block <control>.
xst	 Xst:2677 - Node <inf_10> of sequential type is unconnected in block <control>.
xst	 Xst:2677 - Node <inf_11> of sequential type is unconnected in block <control>.
xst	 Xst:2677 - Node <inf_22> of sequential type is unconnected in block <control>.
xst	 Xst:2677 - Node <inf_23> of sequential type is unconnected in block <control>.
xst	 Xst:2677 - Node <sup_10> of sequential type is unconnected in block <FSM>.
xst	 Xst:2677 - Node <sup_11> of sequential type is unconnected in block <FSM>.
xst	 Xst:2677 - Node <sup_22> of sequential type is unconnected in block <FSM>.
xst	 Xst:2677 - Node <sup_23> of sequential type is unconnected in block <FSM>.
xst	 Xst:2677 - Node <inf_10> of sequential type is unconnected in block <FSM>.
xst	 Xst:2677 - Node <inf_11> of sequential type is unconnected in block <FSM>.
xst	 Xst:2677 - Node <inf_22> of sequential type is unconnected in block <FSM>.
xst	 Xst:2677 - Node <inf_23> of sequential type is unconnected in block <FSM>.
xst	 Xst:1710 - FF/Latch <interpolador/l_4> (without init value) has a constant value of 0 in block <Ecualizador>. This FF/Latch will be trimmed during the optimization process.
xst	 Xst:1336 - (*) More than 100% of Device resources are used

xst	 Xst:1767 - HDL ADVISOR - Resource sharing has identified that some arithmetic operations in this design can share the same physical resources for reduced device utilization. For improved clock frequency you may try to disable resource sharing.
xst	 Xst:2260 - The FF/Latch <blk0000000c> in Unit <blk00000003> is equivalent to the following 5 FFs/Latches : <blk000000555> <blk00000556> <blk00000557> <blk00000558> <blk00000559>
xst	 Xst:2260 - The FF/Latch <blk000000a3> in Unit <blk00000003> is equivalent to the following FF/Latch : <blk00000550>
xst	 Xst:2260 - The FF/Latch <blk0000009f> in Unit <blk00000003> is equivalent to the following FF/Latch : <blk0000055a>
xst	 Xst:2260 - The FF/Latch <blk0000000c> in Unit <blk00000003> is equivalent to the following 5 FFs/Latches : <blk00000555> <blk00000556> <blk00000557> <blk00000558> <blk00000559>
xst	 Xst:2260 - The FF/Latch <blk000000a3> in Unit <blk00000003> is equivalent to the following FF/Latch : <blk00000550>
xst	 Xst:2260 - The FF/Latch <blk0000009f> in Unit <blk00000003> is equivalent to the following FF/Latch : <blk0000055a>
xst	 Xst:2260 - The FF/Latch <blk0000000c> in Unit <blk00000003> is equivalent to the following 5 FFs/Latches : <blk00000555> <blk00000556> <blk00000557> <blk00000558> <blk00000559>
xst	 Xst:2260 - The FF/Latch <blk000000a3> in Unit <blk00000003> is equivalent to the following FF/Latch : <blk00000550>
xst	 Xst:2260 - The FF/Latch <blk0000009f> in Unit <blk00000003> is equivalent to the following FF/Latch : <blk0000055a>
xst	 Xst:2169 - HDL ADVISOR - Some clock signals were not automatically buffered by XST with BUFG/BUFR resources. Please use the buffer_type constraint in order to insert these buffers to the clock signals to help prevent skew problems.

Vemos que los warnings que salen son por tener algunas salidas sin conectar debido a que son bits que se desprecian porque no aportan información o que se truncan, y de bits que tienen un valor constante y que se omitirán al implementar el circuito.