UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN EN FPGA DE LA ETAPA DE FILTRADO FRACCIONAL PARA RECEPTORES QAM DE ANCHO DE BANDA GRANDE

Autor: Jose Carlos Castro Díaz

Tutor: Javier Valls Coquillat

Cotutor: Vicente Torres Carot

Trabajo Fin de Máster presentado en el Departamento de Ingeniería Electrónica de la Universitat Politècnica de València para la obtención del Título de Máster Universitario en Ingeniería de Sistemas Electrónicos

Curso 2019-20

Valencia, Enero de 2020

Resumen

La transmisión a alta velocidad, por ejemplo, mediante fibra óptica, plantea importantes problemas de diseño con las tecnologías actuales. Dado que las tasas requeridas no son alcanzables directamente con un único flujo de datos es necesario paralelizar los diseños, motivo por el cual se ha decidido utilizar un dispositivo FPGA para la implementación de uno de estos sistemas. En concreto en este proyecto se ha diseñado e implementado la etapa de recepción requerida para la demodulación de señales multinivel QAM con ancho de banda del orden de 2 GHz. Se han diseñado las arquitecturas (mediante el uso de transformaciones polifásicas), simulado en Simulink, codificado en HDL y verificado las siguientes etapas: mezclado, filtro racional de cambio de tasa y filtro adaptado (preparado para el sincronismo de tiempo).

Resum

La transmissió a alta velocitat, per exemple, mitjançant fibra òptica, planteja importants problemes de disseny amb les tecnologies actuals. Atés que les taxes requerides no són assolibles directament amb un únic flux de dades és necessari paralelizar els dissenys, motiu pel qual s'ha decidit utilitzar un dispositiu FPGA per a la implementació d'un d'aquests sistemes. En concret en aquest projecte s'ha dissenyat i implementat l'etapa de recepció requerida per a la demodulación de senyals multinivell QAM amb amplada de banda de l'ordre de 2 GHz. S'han dissenyat les arquitectures (mitjançant l'ús de transformacions polifàsiques), simulat en Simulink, codificat en HDL i verificat les següents etapes: barrejat, filtre racional de canvi de taxa i filtre adaptat (preparat per al sincronisme de temps).

Abstract

High-speed transmission, used in optical fiber for example, poses significant design problems with current technologies. Due required rates are not directly achievable with a single data flow, it is necessary to parallelize the designs. That is why it has been decided to use an FPGA device for the implementation of one of these systems. In this project the receiver stage required for the demodulation of multilevel QAM signals with bandwidth in the order of 2GHz has been designed and implemented. For the stages of the mixing, rational rate change filter and adapted filter (prepared for time synchronism), the architectures have been designed (through the use of polyphaser transformations), simulated in Simulink, coded in HDL and verified.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Introducción

En las últimas décadas, los sistemas de telecomunicaciones han experimentado un salto tecnológico sin precedentes, con la aparición del teléfono, y más tarde los teléfonos móviles e internet, estos han tenido que adaptarse rápidamente a un mercado de constante cambio.

Hoy en día tanto usuarios como empresas demandan velocidades cada vez mayores, las cuales son posibles gracias a los métodos actuales de procesado digital.

Una de las técnicas más empleadas en aquellas comunicaciones que requieren alta tasa de bits, es la modulación QAM. Esto es debido a su alta eficiencia espectral, que permite la transmisión de un mayor número de bits por Hercio.

Tradicionalmente, este tipo de procesado digital se ha realizado mediante procesadores y/o DSPs los cuales son capaces de proveer velocidades altísimas capaces de satisfacer las características del sistema. Sin embargo, en los últimos años, las FPGAs se han vuelto cada vez más populares gracias a su gran carga computacional. Esto se debe a su gran capacidad de paralelización, la cual permite que, aplicando la arquitectura adecuada, se puedan lograr mejores resultados que con procesadores o DSPs.

A lo largo de esta memoria nos centraremos en la creación de un receptor QAM de alta velocidad adaptado a las características propias de las FPGA.

1.2. Objetivos

El objetivo del presente proyecto es el de desarrollar un receptor QAM e implementarlo en una FPGA que satisfaga las siguientes especificaciones:

- Los datos proceden de un convertidor A/D de 5 Gsps (estructurados como 16 canales a 312.5 MHz).
- La señal QAM esta modulada con una frecuencia portadora $f_o = f_s/4$, siendo $f_s = 5$ GHz.
- La digitalización se realiza con una tasa de 16/7 muestras por símbolo.

Este receptor estará formado por tres elementos fundamentales, el mezclador, el filtro de ajuste de tasa racional y por último el filtro adaptado, que será de coseno alzado, o en adelante filtro RC.

Dada las especificaciones anteriores, es necesario paralelizar el diseño de todos los componentes del receptor. Para ello se aplicará la transformación polifásica en el diseño de la arquitectura hardware. Los diseños se implementarán en una FPGA mediante HDL y, por último, se verificará el modelo implementado tomando como referencia un modelo paralelizado y correctamente cuantificado que previamente se habrá de desarrollar en Matlab/Simulink.

Por tanto, nuestros objetivos serán:

- Modelizar en Matlab/Simulink versiones no paralelas de los componentes del receptor (i.e. el mezclador, el filtro diezmador y el filtro RC).
- Aplicar la transformación polifásica sobre los elementos anteriormente nombrados y derivar de ese análisis las arquitecturas hardware a implementar.
- Modelizar en Matlab/Simulink versiones paralelas de los componentes del receptor, comprobando que su comportamiento es idéntico al de las versiones no paralelas.
- Codificar e implementar el modelo de todos los componentes y del receptor completo en el lenguaje Verilog HDL.
- Verificar el diseño realizado en el paso anterior contrastándolo con el modelo de Matlab/Simulink.

1.3. Metodología

A continuación, se explicará la metodología seguida durante todo el proyecto.

En primer lugar, se ha modelado el receptor con una arquitectura no paralela, la cual hará de modelo ideal a lo largo de todo el proyecto.

Una vez modelado el receptor, se ha implementado la arquitectura polifásica sobre el mismo y se ha comparado con nuestro modelo ideal.

Una vez realizado el modelo con arquitectura paralela, se ha implementado en un lenguaje HDL, en este caso Verilog.

Por último, se ha verificado el modelo implementado, haciendo uso del modelo de Matlab/Simulink y ModelSim.

Por tanto, podemos decir que el proyecto consta de tres etapas bien diferenciadas:

- Diseño.
- Implementación.
- Verificación.

1.4. Estructura de la memoria

Se ha estructurado la presente memoria en cuatro capítulos tal y como se muestra a continuación:

- **Capítulo 1: Introducción**
  En este capítulo se pretende dar una idea general del proyecto mediante una breve introducción, los objetivos del proyecto, la metodología y la estructura de la memoria.

- **Capítulo 2: Fundamentos teóricos**
  El segundo capítulo pretende profundizar un poco más en los conceptos teóricos más básicos en los que se fundamenta el proyecto. En él se explica de forma general el modelo implementado y poco a poco se ahonda en los elementos más básicos que lo componen.

- **Capítulo 3: Resultados y discusión**
  El capítulo tres describe de forma detallada el proceso seguido durante la implementación del proyecto. En primer lugar, nos centraremos en el modelo de Matlab/Simulink y posteriormente en la implementación en lenguaje HDL.

- **Capítulo 4: Conclusiones**
  Por último, en el capítulo final se hablará de las conclusiones finales, así como las posibles mejoras futuras del proyecto.

Capítulo 2

Fundamentos teóricos

A lo largo del presente capítulo se hará una pequeña introducción sobre la modulación QAM. Nos centraremos específicamente en el receptor QAM y en los componentes que lo integran, es decir, el mezclador, el filtro diezmador y el filtro de coseno alzado.

Una vez introducidos dichos conceptos fundamentales, se explicará de manera genérica la transformación polifásica.

2.1. Modulación QAM

La modulación QAM permite transmitir información a partir de la variación de una señal tanto en amplitud como en fase consiguiendo en el proceso un alto nivel de eficiencia espectral.

En la Figura 1 podemos ver un ejemplo de señal QAM.

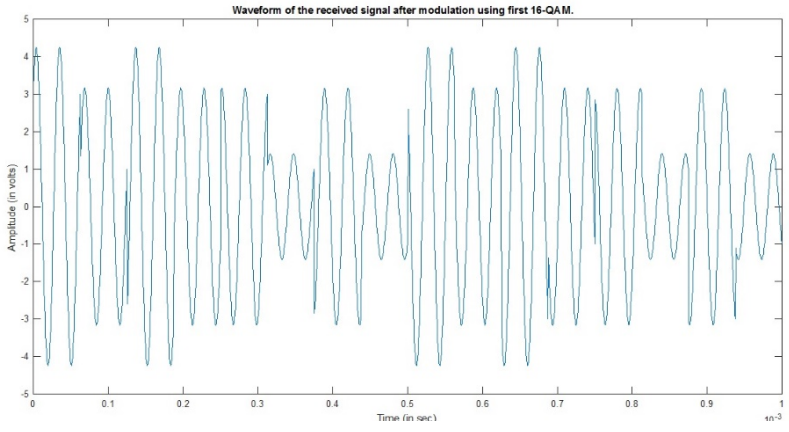

Figura 1: Forma de onda de una modulación 16-QAM

De forma resumida podemos decir que la modulación QAM consiste en modular dos señales independientes que están desfasadas entre si 90º, mediante ASK (Modulación por desplazamiento en Amplitud). La señal modulada QAM es el resultado de sumar dichas señales. El hecho de estar

desfasadas hace que estén en cuadratura.

A menudo una de las señales es conocida como señal “I” (fase) y la otra como señal “Q” (cuadratura). Estas pueden ser representadas mediante las siguientes ecuaciones:

$$I = A_1 \cos(2 f_0 t)$$
$$Q = A_2 \sin(2 f_0 t)$$

En la modulación QAM entran en juego dos elementos fundamentales, un emisor y un receptor.

El emisor (Figura 2) se encarga de agrupar unos y ceros en grupos, cuyo tamaño dependerá del tipo de QAM implementado. Cada grupo se corresponde con un fragmento de la señal, también conocido como símbolo, el cual se transmite a través de un medio físico.

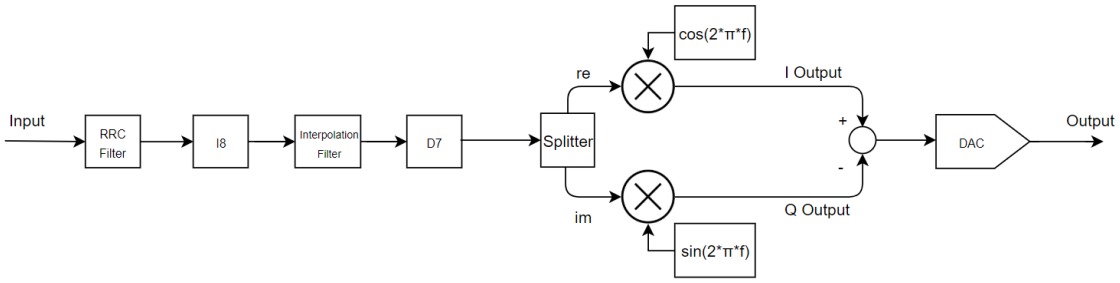

Figura 2: Emisor QAM

El receptor (Figura 3) se encarga de recibir dicha señal y de demodularla, con el fin de obtener la señal original.

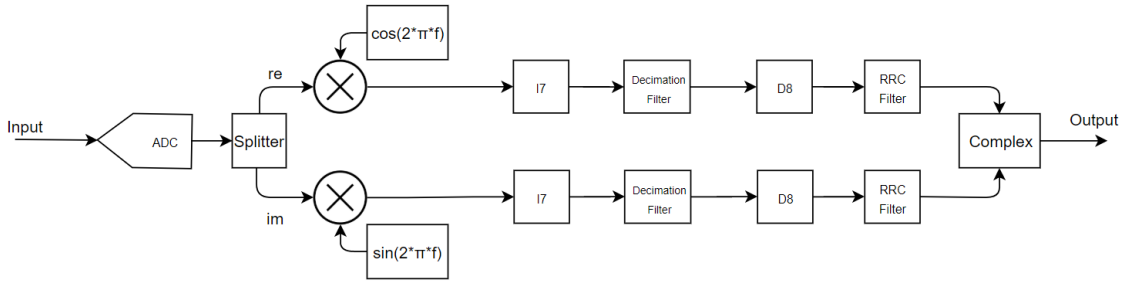

Figura 3: Receptor QAM

Para poder representar los posibles estados QAM, se utiliza el diagrama de constelación. Esta constelación está dividida en una serie de rejillas donde la amplitud y la fase de la señal hacen referencia a una celda en concreto.

En la Figura 4 tenemos un ejemplo de constelación donde A1 y A2 se corresponden con la amplitud y P1 y P2 con la fase.

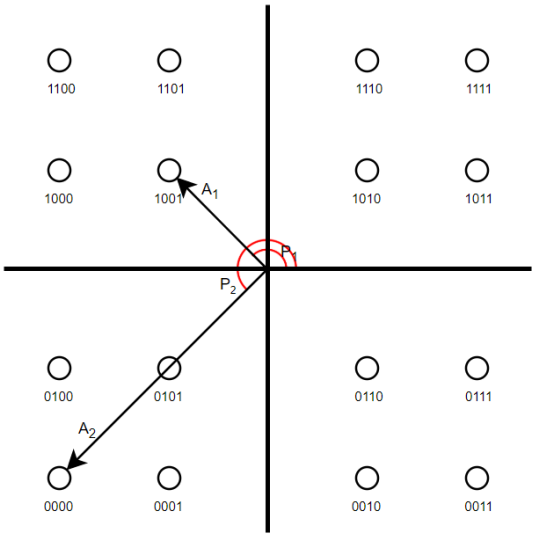

Figura 4: Constelación 16-QAM

Esta imagen se corresponde con la forma 16-QAM, es decir cada símbolo se corresponde con 4 bits. Sin embargo existen muchos más tipos (Figura 5).

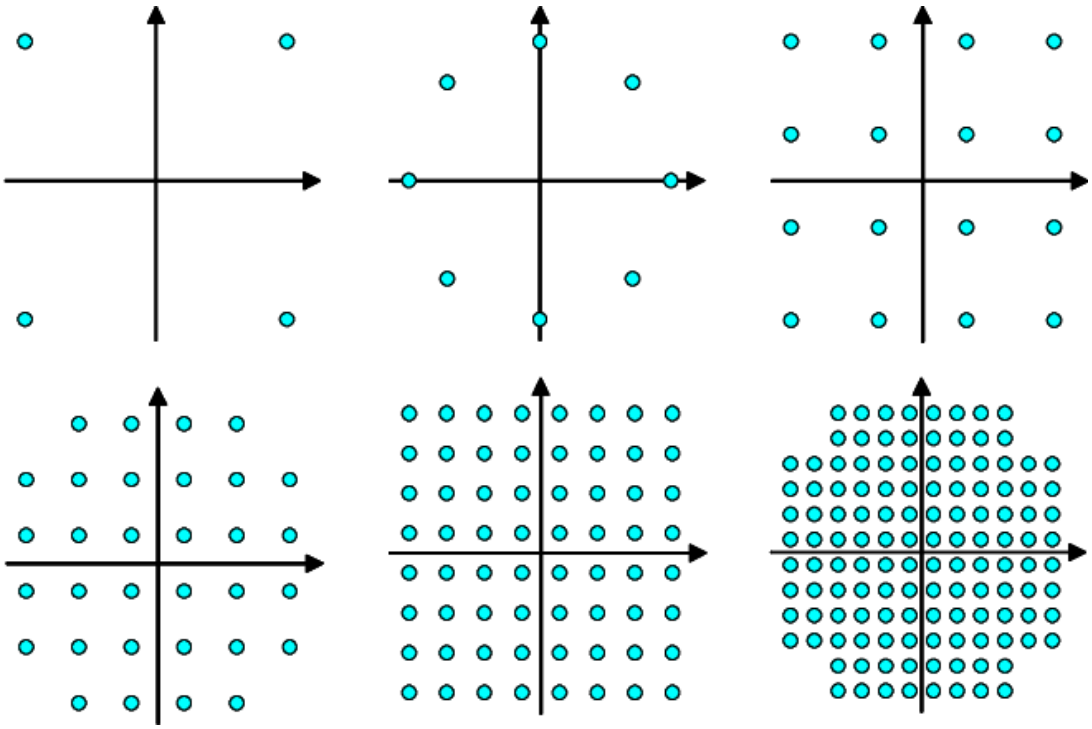

Figura 5: Tipos de QAM

El receptor deberá ser capaz de muestrear correctamente la señal recibida, de manera que no haya pérdida de información. Si esto ocurre, aparecerá en la constelación una nube de puntos tal y como se muestra en la Figura 6.

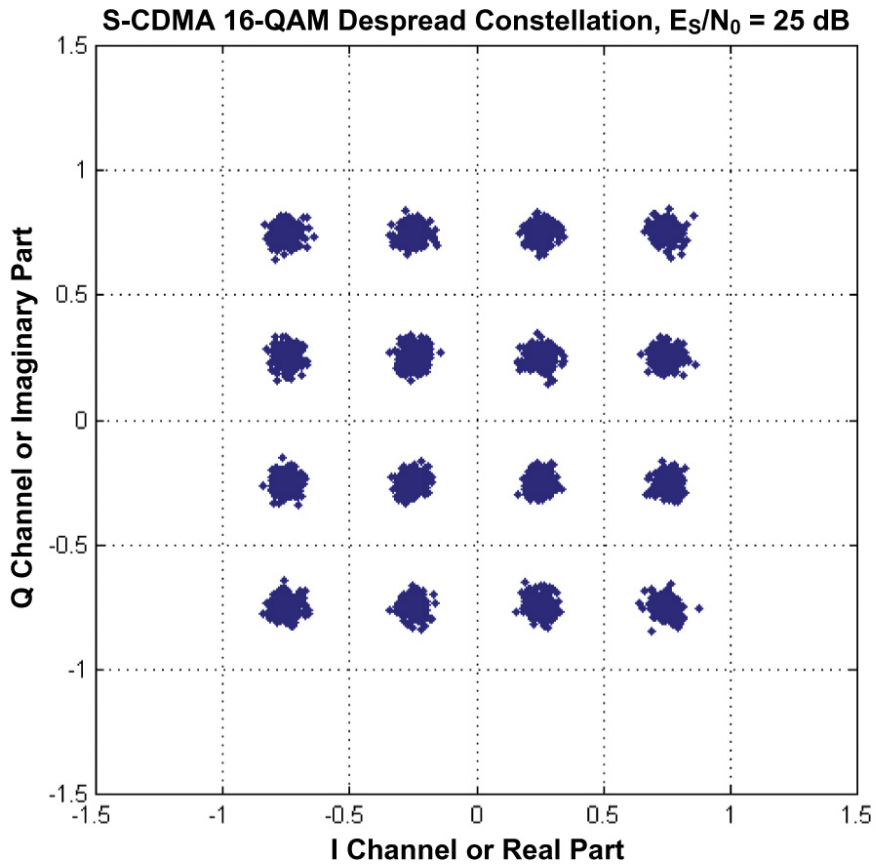

Figura 6: Constelación 16-QAM con ruido

Esto implica que, para modulaciones más complejas, el margen de ruido es mucho menor, ya que los puntos están más cerca entre sí. Por tanto, podemos decir que una de las mayores ventajas de la modulación QAM es su alta velocidad, sin embargo, su mayor defecto es que es susceptible al ruido [1].

2.2. Filtro coseno alzado (RC)

El filtro de coseno alzado es un filtro paso bajo cuyo principal cometido es el de recuperar la información con la mayor precisión posible. Su principal ventaja es la de reducir la interferencia entre símbolos. Esto es posible gracias a que cumple el primer criterio de Nyquist, el cual describe las condiciones que se tienen que dar para que no haya interferencia entre símbolos.

Para explicar este criterio imaginemos una entrada definida por:

$$x\left(t\right) = \sum_{k \in Z} d\left[k\right] g\left(t - kT\right)$$

Donde:

- $d[k]$: Son los datos a transmitir.
- $g(t - kT)$: Es el impulso del filtro.
- T: Distancia entre símbolos.

Si generamos ahora una señal aleatoria de acuerdo a este modelo, donde la entrada dependa solo del símbolo transmitido en un tiempo kT. Matemáticamente se describe de la siguiente forma:

$$x\left(kT\right) = d\left[k\right]$$

Esto queda reflejado en la Figura 7.

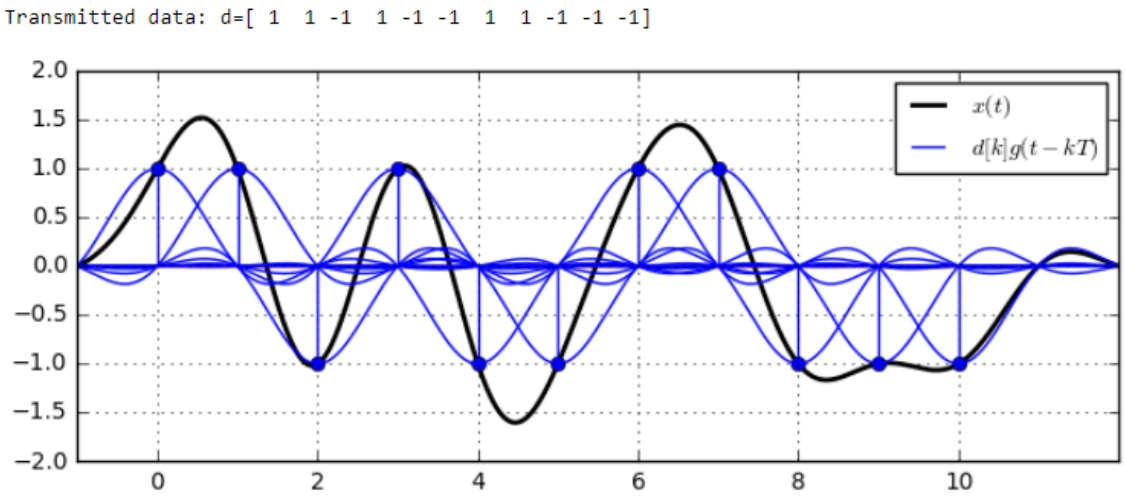

Figura 7: Primer criterio de Nyquist

Tal y como se observa, mediante este criterio se captan todos los símbolos correctamente [2].

A pesar de esto y dada la naturaleza del sistema, es posible que se produzcan retardos o adelantos en las muestras recibidas, esto derivaría en la perdida de información por parte del receptor, es por ello que es posible mejorar este filtro mediante un filtro de retardo fraccionario, de manera que podamos reducir aún más la pérdida de información.

2.3. Filtro diezmador

El filtro diezmador nos permite filtrar y cambiar la frecuencia de muestreo de entrada a otra que deseemos. En este caso, tal y como su propio nombre indica, la frecuencia de salida será menor a la de la entrada.

En la Figura 8 se muestra un ejemplo de un filtro interpolador/diezmador donde en primer lugar se interpola para evitar la pérdida de información y posteriormente se filtra y se diezma.

Figura 8: Filtro de cambio de tasa L/M

La frecuencia de muestreo objetivo se puede expresar como el cociente entre el orden de interpolación L y el orden de diezmado M, es decir L/M. Siendo esta la fracción irreducible.

A continuación, se explica el proceso seguido a lo largo de este filtrado:

1. Imaginemos que partimos de una señal de entrada donde el espectro de frecuencia es el mostrado en la Figura 9.

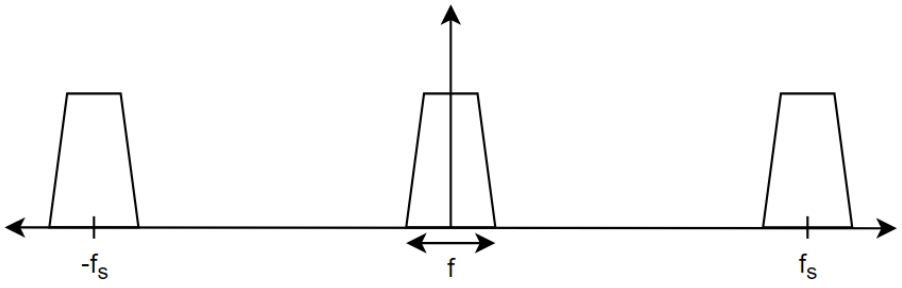

Figura 9: Espectro frecuencial de una señal

2. Si la interpolamos por L aparecerán L copias espectrales (Figura 10).

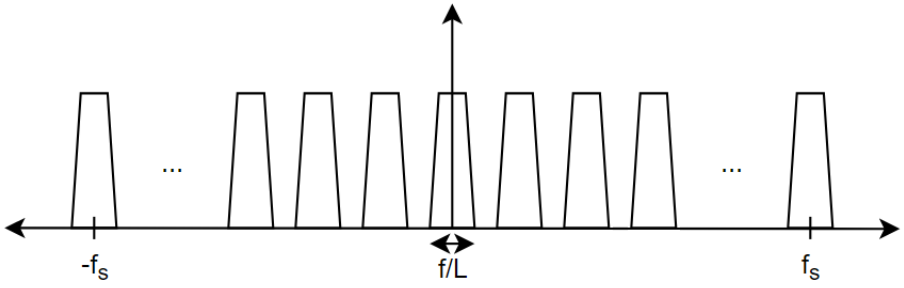

Figura 10: Espectro frecuencial de una señal interpolada por L

3. Durante el filtrado eliminamos las copias creadas (Figura 11).

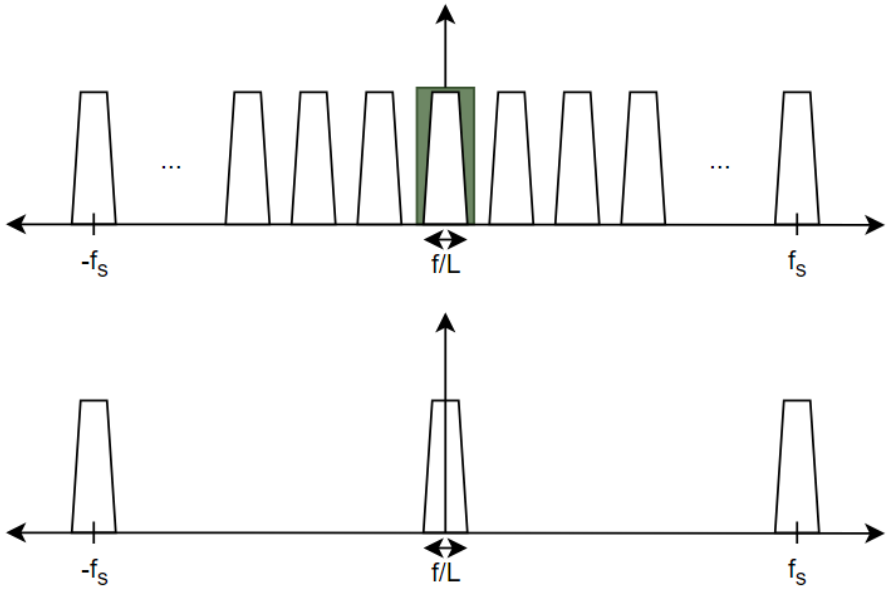

Figura 11: Espectro frecuencial de una señal interpolada por L y filtrada

4. Por último, una vez diezmamos la señal, el espectro frecuencial de esta se expande (Figura 12)

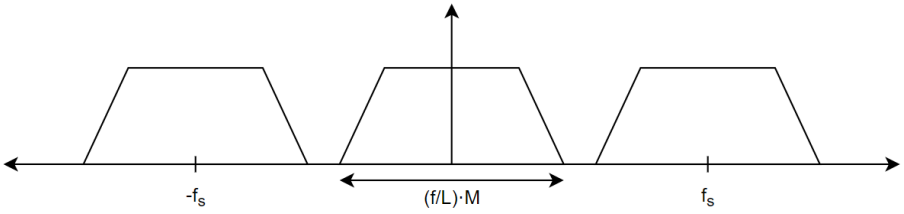

Figura 12: Espectro frecuencial de una señal interpolada por L, filtrada y diezmada por M

Es posible reducir los recursos al mínimo usando identidades nobles (Figura 13).

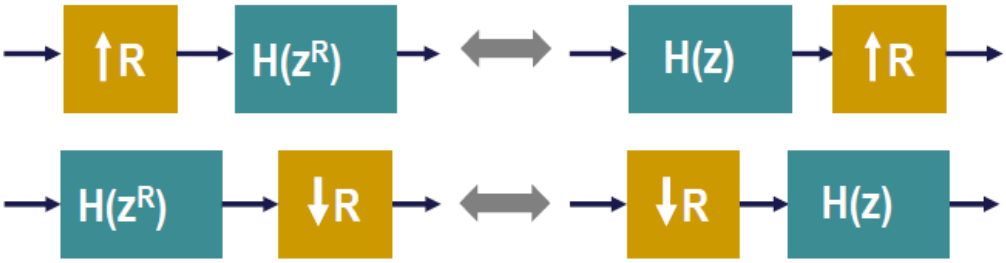

Figura 13: Identidades nobles

Esto se debe a que cuanto menor sea la frecuencia de entrada del filtro menos coeficientes serán necesarios para lograr un correcto filtrado, por ello mediante esta transformación se evita interpolar antes del filtro y se intenta diezmar antes de este.

2.4. Mezclador

De acuerdo a la Figura 3, el mezclador se encarga de separar las muestras que procesarán el canal I y el canal Q. La señal recibida se divide entre las dos ramas de la siguiente forma:

- Rama en fase:
$$I = r \cdot \cos\left(2 \cdot \pi \cdot f_o \cdot \frac{n}{f_s}\right)$$
- Rama en cuadratura:
$$Q = -r \cdot \sin\left(2 \cdot \pi \cdot f_o \cdot \frac{n}{f_s}\right)$$

Dado que el mezclador tiene una frecuencia de $f_o = f_s/4$, tenemos que:

$$I = r \cdot \cos\left(2 \cdot \pi \cdot \frac{f_s}{4} \cdot \frac{n}{f_s}\right) = r \cdot \cos\left(\frac{\pi}{2} \cdot n\right)$$
$$Q = -r \cdot \sin\left(2 \cdot \pi \cdot \frac{f_s}{4} \cdot \frac{n}{f_s}\right) = -r \cdot \sin\left(\frac{\pi}{2} \cdot n\right)$$

A partir de estas ecuaciones obtenemos la distribución de la señal entrante entre ambos canales (Tabla 1).

Muestra (n)	Salida	Canal I	Canal Q
0	1 + 0i	1	0
1	0 + 1i	0	1
2	-1 + 0i	-1	0
3	0 – 1i	0	-1
4	1 + 0i	1	0
5	0 +1i	0	1
6	-1 + 0i	-1	0
7	0 – 1i	0	-1

Tabla 1: Distribución de muestras por canal

2.5. Transformación polifásica

Debido a las altas frecuencias a la que debe operar el sistema propuesto (5 GHz) y debido a que la FPGA no puede alcanzar velocidades tan altas, se deberá aplicar la transformación polifásica.

Esta nueva arquitectura nos permitirá procesar los datos de entrada de forma paralela, permitiéndonos reducir la frecuencia en cada canal. En concreto, nuestra FPGA deberá trabajar con 16 canales (m) y cada uno a 312.5 MHz, si multiplicamos la frecuencia de los canales por el número de estos, tenemos que:

$$f_s = f_{\text{canal}} \cdot m = 312{,}5\,\text{MHz} \cdot 16 = 5000\,\text{MHz} \tag{1}$$

La expresión de la descomposición polifásica en dos subfiltros, es la siguiente:

$$H(z) = \sum_{i=0}^{M} h_i z^{-i} = \sum_{i=0}^{M/2} h_{2i} z^{-2i} + \sum_{i=0}^{M/2} h_{2i+1} z^{-2i} = H_0(z^2) + z^{-1} H_1(z^2) \tag{2}$$

Con el fin de lograr alcanzar la velocidad propuesta, se ha aplicado la transformación polifásica al mezclador, al filtro diezmador y al filtro RC. A continuación, se mostrará el proceso seguido.

2.5.1. Implementación en el filtro RC

El primer elemento sobre el que se ha realizado la arquitectura polifásica ha sido el filtro RC. Para realizar la paralelización deberemos tener claro dos factores:

- Número de entradas (m).
- Número de coeficientes del filtro (n).

La filosofía que sigue la arquitectura polifásica es la de dividir el trabajo en una serie de subfiltros. Si asumimos que queremos aplicar una transformación de m entradas, en lugar de usar un único filtro a una frecuencia (f_{clk}), utilizaremos m filtros que trabajarán a una frecuencia (f_{canal}) m veces más baja. Por tanto, tenemos que:

$$f_{\text{canal}} = \frac{f_{\text{clk}}}{m} \tag{3}$$

En la Figura 14 podemos ver un ejemplo del sistema con el que tendremos que trabajar.

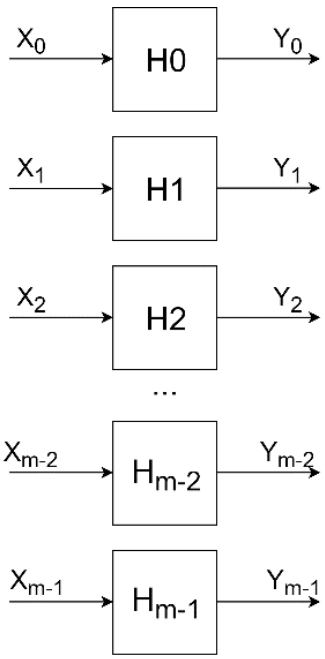

Figura 14: Arquitectura polifásica de m entradas

Tal y como se observa, el número de subfiltros depende directamente del número de entradas que queramos.

En la Tabla 2 y Tabla 3 se puede observar el análisis temporal de la transformación a una arquitectura polifásica de un sistema de m entradas (y m salidas) y n coeficientes, desde el punto de vista de las salidas.

	Z^{0} (t_0)	Z^{-1} (t_1)	Z^{-2} (t_2)	Z^{-3} (t_3)	Z^{-4} (t_4)	Z^{-5} (t_5)	Z^{-6} (t_6)	…	Z^{-n+2} (t_{n-2})	Z^{-n+1} (t_{n-1})	Z^{-n} (t_n)	Z^{-n-1} (t_{n+1})	Z^{-n-2} (t_{n+2})	…	$Z^{-(n+m)+2}$ (t_{n+m-2})
H_0	$h_0 x_0$	$h_0 x_1$	$h_0 x_2$	$h_0 x_3$	$h_0 x_4$	$h_0 x_5$	$h_0 x_6$	…							
H_1		$h_1 x_0$	$h_1 x_1$	$h_1 x_2$	$h_1 x_3$	$h_1 x_4$	$h_1 x_5$	…							
H_2			$h_2 x_0$	$h_2 x_1$	$h_2 x_2$	$h_2 x_3$	$h_2 x_4$	…							
H_3				$h_3 x_0$	$h_3 x_1$	$h_3 x_2$	$h_3 x_3$	…							
...					…	…	…	…	…	…	…	…	…	…	
H_{n-2}									$h_{n-2} x_0$	$h_{n-2} x_1$	$h_{n-2} x_2$	$h_{n-2} x_3$	…	…	
H_{n-1}										$h_{n-1} x_0$	$h_{n-1} x_1$	$h_{n-1} x_2$	$h_{n-1} x_3$	…	$h_{n-1} x_{m-1}$

Tabla 2: Análisis temporal de una arquitectura polifásica de m entradas y n coeficientes

	t_0	t_1	t_2	t_3	…	t_{n+m-3}	t_{n+m-2}
Y_0 (m*t + 0)	0	m	2m	3m	…	t_{n+m-3}m	t_{n+m-2}m
Y_1 (m*t + 1)	1	m+1	2m+1	3m+1	…	t_{n+m-3}m+1	t_{n+m-2}m+1
Y_2 (m*t + 2)	2	m+2	2m+2	3m+2	…	t_{n+m-3}m+2	t_{n+m-2}m+2
Y_3 (m*t + 3)	3	m+3	2m+3	3m+3	…	t_{n+m-3}m+3	t_{n+m-2}m+3
...	…	…	…	…	…	…	…
Y_{m-2} (m*t + (m-2))	m-2	2m-2	3m-2	4m-2	…	t_{n+m-3}2m-2	t_{n+m-2}3m-2
Y_{m-1} (m*t + (m-1))	m-1	2m-1	3m-1	4m-1	…	t_{n+m-3}2m-1	t_{n+m-2}3m-1

Tabla 3: Análisis temporal de una arquitectura polifásica de m salidas y n coeficientes

A partir de ambas tablas podemos obtener las ecuaciones del sistema. La Tabla 2 nos indica el subfiltro afectado, mientras que la Tabla 3 nos indica a que salida afecta dicha operación de filtrado.

El número de salidas dependerá del número de entradas. Cada una estará definida por una ecuación característica:

$$
\begin{aligned}
Y_0 &= H_0X_0 + (H_1X_{m-1} + H_2X_{m-2} + H_3X_{m-3} + \cdots)\cdot Z^{-1} \\
&\quad + (H_{m+1}X_{m-1} + H_{m+2}X_{m-2} + H_{m+3}X_{m-3} + \cdots)\cdot Z^{-1} \\
&\quad + (H_{2m+1}X_{m-1} + H_{2m+2}X_{m-2} + H_{2m+3}X_{m-3} + \cdots)\cdot Z^{-1} \\
&\quad + \cdots \\
Y_1 &= H_0X_1 + H_1X_0 + (H_2X_{m-1} + H_3X_{m-2} + H_4X_{m-3} + \cdots)\cdot Z^{-1} \\
&\quad + (H_{m+2}X_{m-1} + H_{m+3}X_{m-2} + H_{m+4}X_{m-3} + \cdots)\cdot Z^{-1} \\
&\quad + (H_{2m+2}X_{m-1} + H_{2m+3}X_{m-2} + H_{2m+4}X_{m-3} + \cdots)\cdot Z^{-1} \\
&\quad + \cdots \\
Y_2 &= H_0X2 + H_1X_1 + H_2X_0 + (H_3X_{m-1} + H_4X_{m-2} + H_5X_{m-3} + \cdots)\cdot Z^{-1} \\
&\quad + (H_{m+3}X_{m-1} + H_{m+4}X_{m-2} + H_{m+5}X_{m-3} + \cdots)\cdot Z^{-1} \\
&\quad + (H_{2m+3}X_{m-1} + H_{2m+4}X_{m-2} + H_{2m+5}X_{m-3} + \cdots)\cdot Z^{-1} \\
&\quad + \cdots \\
Y_3 &= H_0X3 + H_1X2 + H_2X_1 + H_3X_0 + (H_4X_{m-1} + H_5X_{m-2} + H_6X_{m-3} + \cdots)\cdot Z^{-1} \\
&\quad + (H_{m+4}X_{m-1} + H_{m+5}X_{m-2} + H_{m+6}X_{m-3} + \cdots)\cdot Z^{-1} \\
&\quad + (H_{2m+4}X_{m-1} + H_{2m+5}X_{m-2} + H_{2m+6}X_{m-3} + \cdots)\cdot Z^{-1} \\
&\quad + \cdots \\
&\text{etc.}
\end{aligned}
\tag{4}
$$

A partir de estas ecuaciones es posible construir el modelo equivalente. En el apartado 3.2.2 podemos observar un ejemplo práctico donde se sigue este mismo procedimiento.

2.5.2. Implementación en el filtro diezmador

Partiendo del filtro de cambio de tasa de la Figura 15, donde L equivale al factor de interpolación y M al factor de diezmado, podemos identificar el cambio de las frecuencias de muestreo a lo largo del sistema.

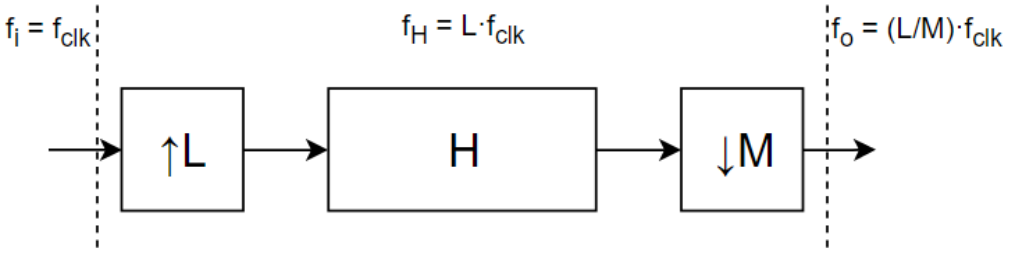

Figura 15: Filtro de cambio de tasa

Así pues, partiendo de una frecuencia de entrada (f_i) dada por el reloj del sistema f_{clk}, una vez se interpole la entrada, la frecuencia en el filtro (f_H) será la del factor de interpolación multiplicado por la frecuencia de reloj y tras el posterior diezmado, la frecuencia de salida será la frecuencia en el filtro dividido por el factor de diezmado, es decir:

$$\begin{aligned} f_i &= f_{\text{clk}} \\ f_H &= L \cdot f_i \\ f_0 &= \frac{f_H}{M} = \frac{L}{M} f_{\text{clk}} \end{aligned} \tag{5}$$

Por tanto, según los valores de L y M, la frecuencia de salida será mayor o menor que la de entrada. En caso de que $L > M$, hablamos de un filtro interpolador y en caso de que $L < M$, hablamos de un filtro diezmador.

Antes de hablar de la implementación de la arquitectura polifásica, debemos hablar de las identidades nobles. Las identidades nobles es el nombre que se le da a las equivalencias entre filtros donde la interpolación o el diezmado se hace antes o después de filtrar (Figura 16 y Figura 17) [3].

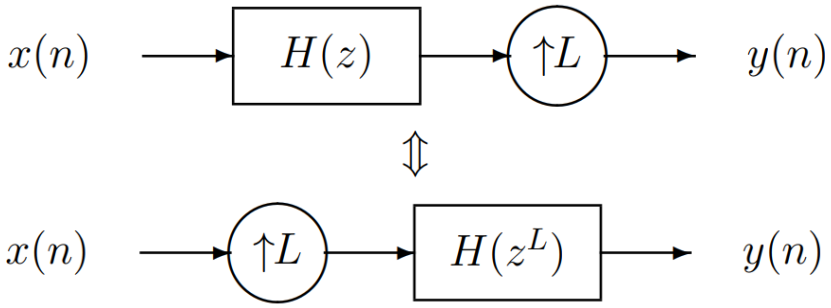

Figura 16: Identidad noble de la interpolación

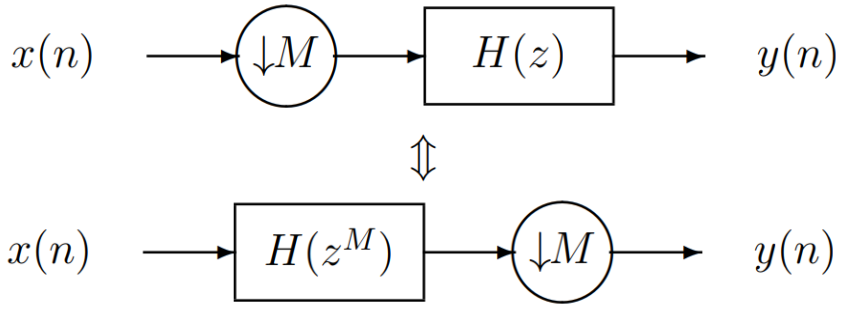

Figura 17: Identidad noble del diezmado

Nuestro propósito, a fin de ahorrar recursos, es la de evitar trabajar con frecuencias altas en el filtro, por tanto, deberemos intentar interpolar después de filtrar o diezmar antes de filtrar. Para explicar de la manera más sencilla la arquitectura polifásica de un filtro de cambio de tasa, vamos a tomar como ejemplo un caso general (Figura 18).

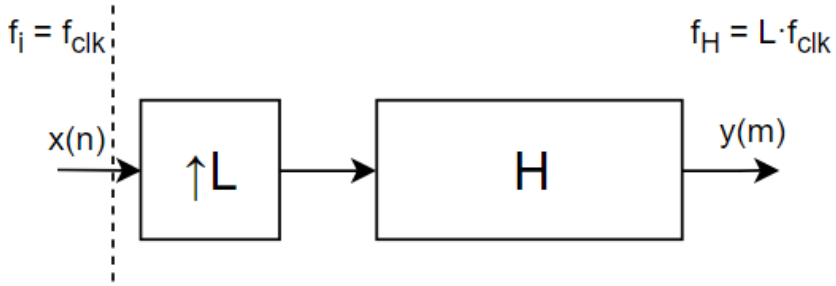

Figura 18: Filtro con un interpolador

En esta figura podemos ver que se trata de un filtro interpolador con un factor de interpolación de orden L donde la frecuencia de entrada es la del reloj del sistema y la frecuencia en el filtro es de L veces la frecuencia de entrada.

Partiendo de este caso, podemos elaborar un modelo equivalente (Figura 19) donde la entrada ha sido paralelizada por un factor P_i, disponemos de P_o salidas y el número de subfiltros coincide con el factor de interpolación (L). La paralelización total del sistema será $L \cdot P_i$.

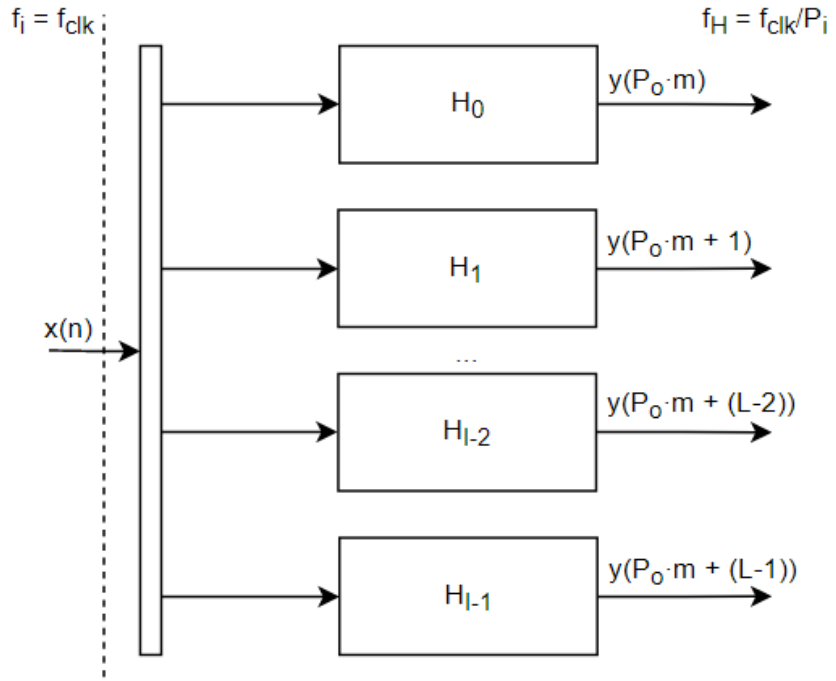

Figura 19: Filtro paralelizado para un factor de interpolación de I

Una vez llegado a este punto habrá que agrupar los coeficientes de cada subfiltro. A partir de un filtro con n coeficientes:

$$H = \{0, 1, 2, 3, 4, 5, 6, ..., n{-}1\}$$

Deberán ser agrupados de la siguiente manera:

$$\begin{aligned} H_0 &= \{0, L, 2L, 3L, \text{etc.}\} \\ H_1 &= \{1, 1+L, 1+2L, 1+3L, \text{etc.}\} \\ &\dots \\ H_{I-2} &= \{0, (L-2)+L, (L-2)+2L, (L-2)+3L, \text{etc.}\} \\ H_{I-1} &= \{0, (L-1)+L, (L-1)+2L, (L-1)+3L, \text{etc.}\} \end{aligned} \tag{6}$$

Una vez agrupados los coeficientes debemos aplicar sobre cada subfiltro el procedimiento visto en el apartado 2.5.1. De esta manera obtendremos las ecuaciones de salida de cada subfiltro. En este caso, cada subfiltro tendrá P_i salidas.

A continuación se va a mostrar un diagrama temporal para el sistema de la Figura 20, correspondiente a un filtro diezmador 3/4 paralelizado por 16. El número de salidas de dicho sistema es de 12.

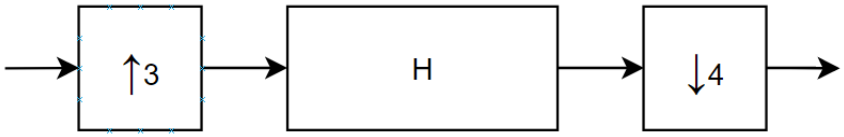

Figura 20: Filtro diezmador por 3 y paralelizado por 16

Si observamos la Figura 21, podemos ver la selección de datos con respecto al tiempo.

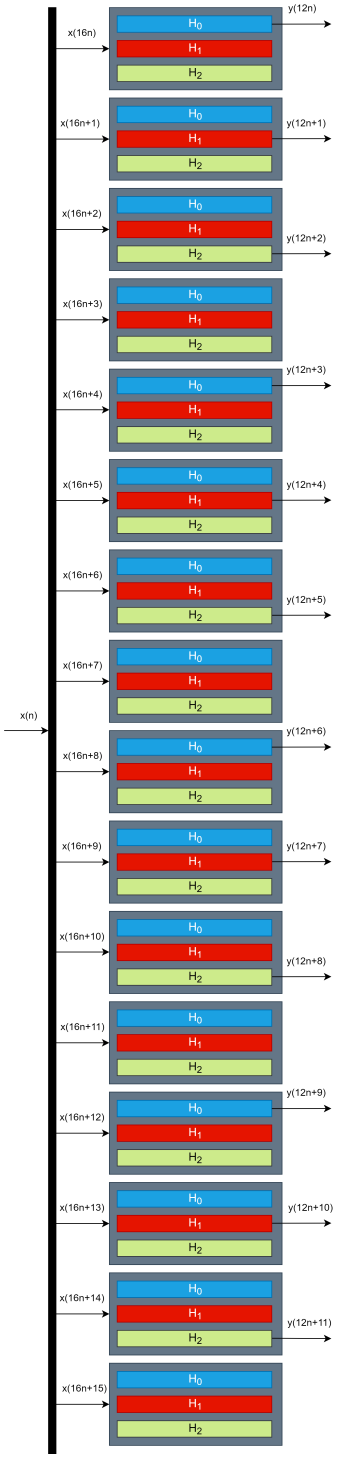

Figura 21: Esquema temporal del filtro diezmador implementado con una arquitectura polifásica

Tal y como se aprecia, en el instante 0 la salida se corresponde a la primera salida del filtro H0, en el instante 2 se corresponde a la segunda salida del filtro H1, en el 3 a la tercera salida del filtro H3, etc. Cabe mencionar que en los instantes 4, 8, 12 y 16 no se saca ninguna muestra.

Si queremos que la Figura 21 sea totalmente equivalente a la Figura 20, a la salida deberemos interpolar por 12, de forma que la relación entre interpolación y diezmado sea igual:

$$\frac{I}{D} = \frac{3}{4} = \frac{12}{16} = 0{,}75 \tag{7}$$

En este caso la entrada esta paralelizada por 16 y la salida final es de 12 para que coincidan las frecuencias. Para un caso general donde el factor de interpolado y diezmado son L y M, el número de entradas es P_i, y el número de salidas es P_o, aplicamos la ecuación anterior para poder calcular el número de salidas a partir de P, L y M:

$$P_o = \frac{L}{M} \cdot P_i \tag{8}$$

Capítulo 3

Resultados y discusión

A lo largo de este capítulo se explicará el proceso llevado a cabo a la hora de modelar el receptor tanto en Matlab/Simulink como en Verilog.

3.1. Diseño y modelado del receptor en Matlab/Simulink

Para la realización del receptor, realizaremos en primer lugar un modelo no paralelizado que nos servirá como modelo de referencia.

A partir de ahí, deberemos realizar el modelo con arquitectura polifásica sin cuantificación, es decir, trabajando con puntos flotantes. Posteriormente deberemos cuantificar las entradas, salidas y coeficientes a punto fijo.

El esquema del receptor que se quiere implementar es el visto en la Figura 3.

La implementación de cada elemento del receptor se ha realizado en dos pasos. En primer lugar, se ha creado un modelo no paralelizado y en segundo lugar, una vez comprobado que funciona correctamente, se ha aplicado la transformación polifásica para paralelizar dicho elemento.

Como ya se ha comentado previamente, dichos elementos son el filtro RC, el filtro diezmador y el mezclador. A continuación, se explicará la implementación de cada uno más detalladamente.

3.1.1. Filtro RC

Para modelizar el filtro RC lo primero que se deberá hacer es obtener los coeficientes necesarios de acuerdo a nuestras necesidades.

El cálculo de los coeficientes se ha realizado mediante el comando de Matlab:

rcosine (Fd,Fs,Type,Rolloff,Delay)

Donde:

- Fd: Frecuencia de muestreo de entrada
- Fs: Frecuencia de muestreo del filtro

- Type: Tipo de filtrado
- Rolloff: Roll off del filtro
- Delay: Retardo del filtro

En la Figura 22 puede verse la configuración de dichos coeficientes.

```
%%Coeficientes filtro coseno alzado

Rolloff = 0.14;
Delay = 10;
Irrc = 2;
fracs = 16; % numero fracciones
mu_rx = 0; % retardo fraccional

hrrc_ri_full=rcosine(1,Irrc*fracs,'sqrt',Rolloff,Delay); % Filtro adaptado

hrrc_ri_id = hrrc_ri_full(mu_rx+1:fracs:end);
```

Figura 22: Configuración del filtro RC

Para su implementación en Simulink, tan solo debemos utilizar un bloque de filtro digital (Digital Filter), donde podamos especificar los coeficientes que queremos utilizar (Figura 23).

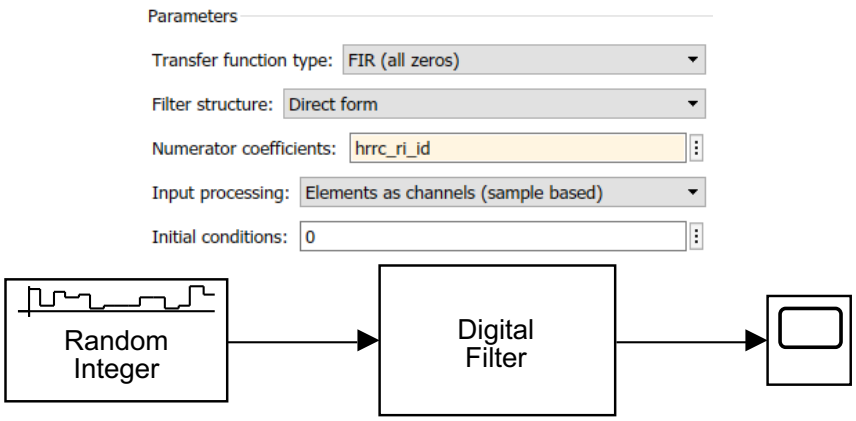

Figura 23: Filtro RC no paralelizado y configuración

Si aplicamos la transformación polifásica, tal y como se ha visto en el capítulo 2, tenemos que tener en cuenta, que este filtro dispondrá de 14 entradas y un total de 41 coeficientes.

Ahora bien, para este caso específico, realizamos el análisis de la Tabla 2.

	z^0	z^1	z^2	z^3	z^4	z^5	z^6	z^7	z^8	z^9	z^{10}	z^{11}	z^{12}	z^{13}	z^{14}	z^{15}	z^{16}	z^{17}	z^{18}	z^{19}	z^{20}	z^{21}	z^{22}	z^{23}	z^{24}	z^{25}	z^{26}	z^{27}	z^{28}	z^{29}	z^{30}	z^{31}	z^{32}	z^{33}
H_0	$h_0{\cdot}x_0$	$h_0{\cdot}x_1$	$h_0{\cdot}x_2$	$h_0{\cdot}x_3$	$h_0{\cdot}x_4$	$h_0{\cdot}x_5$	$h_0{\cdot}x_6$	$h_0{\cdot}x_7$	$h_0{\cdot}x_8$	$h_0{\cdot}x_9$	$h_0{\cdot}x_{10}$	$h_0{\cdot}x_{11}$	$h_0{\cdot}x_{12}$	$h_0{\cdot}x_{13}$																					
H_1		$h_1{\cdot}x_0$	$h_1{\cdot}x_1$	$h_1{\cdot}x_2$	$h_1{\cdot}x_3$	$h_1{\cdot}x_4$	$h_1{\cdot}x_5$	$h_1{\cdot}x_6$	$h_1{\cdot}x_7$	$h_1{\cdot}x_8$	$h_1{\cdot}x_9$	$h_1{\cdot}x_{10}$	$h_1{\cdot}x_{11}$	$h_1{\cdot}x_{12}$	$h_1{\cdot}x_{13}$																				
H_2			$h_2{\cdot}x_0$	$h_2{\cdot}x_1$	$h_2{\cdot}x_2$	$h_2{\cdot}x_3$	$h_2{\cdot}x_4$	$h_2{\cdot}x_5$	$h_2{\cdot}x_6$	$h_2{\cdot}x_7$	$h_2{\cdot}x_8$	$h_2{\cdot}x_9$	$h_2{\cdot}x_{10}$	$h_2{\cdot}x_{11}$	$h_2{\cdot}x_{12}$	$h_2{\cdot}x_{13}$																			
H_3				$h_3{\cdot}x_0$	$h_3{\cdot}x_1$	$h_3{\cdot}x_2$	$h_3{\cdot}x_3$	$h_3{\cdot}x_4$	$h_3{\cdot}x_5$	$h_3{\cdot}x_6$	$h_3{\cdot}x_7$	$h_3{\cdot}x_8$	$h_3{\cdot}x_9$	$h_3{\cdot}x_{10}$	$h_3{\cdot}x_{11}$	$h_3{\cdot}x_{12}$	$h_3{\cdot}x_{13}$																		
H_4					$h_4{\cdot}x_0$	$h_4{\cdot}x_1$	$h_4{\cdot}x_2$	$h_4{\cdot}x_3$	$h_4{\cdot}x_4$	$h_4{\cdot}x_5$	$h_4{\cdot}x_6$	$h_4{\cdot}x_7$	$h_4{\cdot}x_8$	$h_4{\cdot}x_9$	$h_4{\cdot}x_{10}$	$h_4{\cdot}x_{11}$	$h_4{\cdot}x_{12}$	$h_4{\cdot}x_{13}$																	
H_5						$h_5{\cdot}x_0$	$h_5{\cdot}x_1$	$h_5{\cdot}x_2$	$h_5{\cdot}x_3$	$h_5{\cdot}x_4$	$h_5{\cdot}x_5$	$h_5{\cdot}x_6$	$h_5{\cdot}x_7$	$h_5{\cdot}x_8$	$h_5{\cdot}x_9$	$h_5{\cdot}x_{10}$	$h_5{\cdot}x_{11}$	$h_5{\cdot}x_{12}$	$h_5{\cdot}x_{13}$															
H_6							$h_6{\cdot}x_0$	$h_6{\cdot}x_1$	$h_6{\cdot}x_2$	$h_6{\cdot}x_3$	$h_6{\cdot}x_4$	$h_6{\cdot}x_5$	$h_6{\cdot}x_6$	$h_6{\cdot}x_7$	$h_6{\cdot}x_8$	$h_6{\cdot}x_9$	$h_6{\cdot}x_{10}$	$h_6{\cdot}x_{11}$	$h_6{\cdot}x_{12}$	$h_6{\cdot}x_{13}$													
H_7								$h_7{\cdot}x_0$	$h_7{\cdot}x_1$	$h_7{\cdot}x_2$	$h_7{\cdot}x_3$	$h_7{\cdot}x_4$	$h_7{\cdot}x_5$	$h_7{\cdot}x_6$	$h_7{\cdot}x_7$	$h_7{\cdot}x_8$	$h_7{\cdot}x_9$	$h_7{\cdot}x_{10}$	$h_7{\cdot}x_{11}$	$h_7{\cdot}x_{12}$	$h_7{\cdot}x_{13}$										
H_8									$h_8{\cdot}x_0$	$h_8{\cdot}x_1$	$h_8{\cdot}x_2$	$h_8{\cdot}x_3$	$h_8{\cdot}x_4$	$h_8{\cdot}x_5$	$h_8{\cdot}x_6$	$h_8{\cdot}x_7$	$h_8{\cdot}x_8$	$h_8{\cdot}x_9$	$h_8{\cdot}x_{10}$	$h_8{\cdot}x_{11}$	$h_8{\cdot}x_{12}$	$h_8{\cdot}x_{13}$									
H_9										$h_9{\cdot}x_0$	$h_9{\cdot}x_1$	$h_9{\cdot}x_2$	$h_9{\cdot}x_3$	$h_9{\cdot}x_4$	$h_9{\cdot}x_5$	$h_9{\cdot}x_6$	$h_9{\cdot}x_7$	$h_9{\cdot}x_8$	$h_9{\cdot}x_9$	$h_9{\cdot}x_{10}$	$h_9{\cdot}x_{11}$	$h_9{\cdot}x_{12}$	$h_9{\cdot}x_{13}$								
H_{10}											$h_{10}{\cdot}x_0$	$h_{10}{\cdot}x_1$	$h_{10}{\cdot}x_2$	$h_{10}{\cdot}x_3$	$h_{10}{\cdot}x_4$	$h_{10}{\cdot}x_5$	$h_{10}{\cdot}x_6$	$h_{10}{\cdot}x_7$	$h_{10}{\cdot}x_8$	$h_{10}{\cdot}x_9$	$h_{10}{\cdot}x_{10}$	$h_{10}{\cdot}x_{11}$	$h_{10}{\cdot}x_{12}$	$h_{10}{\cdot}x_{13}$							
H_{11}												$h_{11}{\cdot}x_0$	$h_{11}{\cdot}x_1$	$h_{11}{\cdot}x_2$	$h_{11}{\cdot}x_3$	$h_{11}{\cdot}x_4$	$h_{11}{\cdot}x_5$	$h_{11}{\cdot}x_6$	$h_{11}{\cdot}x_7$	$h_{11}{\cdot}x_8$	$h_{11}{\cdot}x_9$	$h_{11}{\cdot}x_{10}$	$h_{11}{\cdot}x_{11}$	$h_{11}{\cdot}x_{12}$	$h_{11}{\cdot}x_{13}$						
H_{12}													$h_{12}{\cdot}x_0$	$h_{12}{\cdot}x_1$	$h_{12}{\cdot}x_2$	$h_{12}{\cdot}x_3$	$h_{12}{\cdot}x_4$	$h_{12}{\cdot}x_5$	$h_{12}{\cdot}x_6$	$h_{12}{\cdot}x_7$	$h_{12}{\cdot}x_8$	$h_{12}{\cdot}x_9$	$h_{12}{\cdot}x_{10}$	$h_{12}{\cdot}x_{11}$	$h_{12}{\cdot}x_{12}$	$h_{12}{\cdot}x_{13}$						
H_{13}														$h_{13}{\cdot}x_0$	$h_{13}{\cdot}x_1$	$h_{13}{\cdot}x_2$	$h_{13}{\cdot}x_3$	$h_{13}{\cdot}x_4$	$h_{13}{\cdot}x_5$	$h_{13}{\cdot}x_6$	$h_{13}{\cdot}x_7$	$h_{13}{\cdot}x_8$	$h_{13}{\cdot}x_9$	$h_{13}{\cdot}x_{10}$	$h_{13}{\cdot}x_{11}$	$h_{13}{\cdot}x_{12}$	$h_{13}{\cdot}x_{13}$						
H_{14}															$h_{14}{\cdot}x_0$	$h_{14}{\cdot}x_1$	$h_{14}{\cdot}x_2$	$h_{14}{\cdot}x_3$	$h_{14}{\cdot}x_4$	$h_{14}{\cdot}x_5$	$h_{14}{\cdot}x_6$	$h_{14}{\cdot}x_7$	$h_{14}{\cdot}x_8$	$h_{14}{\cdot}x_9$	$h_{14}{\cdot}x_{10}$	$h_{14}{\cdot}x_{11}$	$h_{14}{\cdot}x_{12}$	$h_{14}{\cdot}x_{13}$						
H_{15}																$h_{15}{\cdot}x_0$	$h_{15}{\cdot}x_1$	$h_{15}{\cdot}x_2$	$h_{15}{\cdot}x_3$	$h_{15}{\cdot}x_4$	$h_{15}{\cdot}x_5$	$h_{15}{\cdot}x_6$	$h_{15}{\cdot}x_7$	$h_{15}{\cdot}x_8$	$h_{15}{\cdot}x_9$	$h_{15}{\cdot}x_{10}$	$h_{15}{\cdot}x_{11}$	$h_{15}{\cdot}x_{12}$	$h_{15}{\cdot}x_{13}$					
H_{16}																	$h_{16}{\cdot}x_0$	$h_{16}{\cdot}x_1$	$h_{16}{\cdot}x_2$	$h_{16}{\cdot}x_3$	$h_{16}{\cdot}x_4$	$h_{16}{\cdot}x_5$	$h_{16}{\cdot}x_6$	$h_{16}{\cdot}x_7$	$h_{16}{\cdot}x_8$	$h_{16}{\cdot}x_9$	$h_{16}{\cdot}x_{10}$	$h_{16}{\cdot}x_{11}$	$h_{16}{\cdot}x_{12}$	$h_{16}{\cdot}x_{13}$				
H_{17}																		$h_{17}{\cdot}x_0$	$h_{17}{\cdot}x_1$	$h_{17}{\cdot}x_2$	$h_{17}{\cdot}x_3$	$h_{17}{\cdot}x_4$	$h_{17}{\cdot}x_5$	$h_{17}{\cdot}x_6$	$h_{17}{\cdot}x_7$	$h_{17}{\cdot}x_8$	$h_{17}{\cdot}x_9$	$h_{17}{\cdot}x_{10}$	$h_{17}{\cdot}x_{11}$	$h_{17}{\cdot}x_{12}$	$h_{17}{\cdot}x_{13}$			
H_{18}																			$h_{18}{\cdot}x_0$	$h_{18}{\cdot}x_1$	$h_{18}{\cdot}x_2$	$h_{18}{\cdot}x_3$	$h_{18}{\cdot}x_4$	$h_{18}{\cdot}x_5$	$h_{18}{\cdot}x_6$	$h_{18}{\cdot}x_7$	$h_{18}{\cdot}x_8$	$h_{18}{\cdot}x_9$	$h_{18}{\cdot}x_{10}$	$h_{18}{\cdot}x_{11}$	$h_{18}{\cdot}x_{12}$	$h_{18}{\cdot}x_{13}$		
H_{19}																				$h_{19}{\cdot}x_0$	$h_{19}{\cdot}x_1$	$h_{19}{\cdot}x_2$	$h_{19}{\cdot}x_3$	$h_{19}{\cdot}x_4$	$h_{19}{\cdot}x_5$	$h_{19}{\cdot}x_6$	$h_{19}{\cdot}x_7$	$h_{19}{\cdot}x_8$	$h_{19}{\cdot}x_9$	$h_{19}{\cdot}x_{10}$	$h_{19}{\cdot}x_{11}$	$h_{19}{\cdot}x_{12}$	$h_{19}{\cdot}x_{13}$		
H_{20}																					$h_{20}{\cdot}x_0$	$h_{20}{\cdot}x_1$	$h_{20}{\cdot}x_2$	$h_{20}{\cdot}x_3$	$h_{20}{\cdot}x_4$	$h_{20}{\cdot}x_5$	$h_{20}{\cdot}x_6$	$h_{20}{\cdot}x_7$	$h_{20}{\cdot}x_8$	$h_{20}{\cdot}x_9$	$h_{20}{\cdot}x_{10}$	$h_{20}{\cdot}x_{11}$	$h_{20}{\cdot}x_{12}$	$h_{20}{\cdot}x_{13}$	

Tabla 4: Análisis temporal de la arquitectura polifásica de un filtro RC desde H0 hasta H20

	z^{21}	z^{22}	z^{23}	z^{24}	z^{25}	z^{26}	z^{27}	z^{28}	z^{29}	z^{30}	z^{31}	z^{32}	z^{33}	z^{34}	z^{35}	z^{36}	z^{37}	z^{38}	z^{39}	z^{40}	z^{41}	z^{42}	z^{43}	z^{44}	z^{45}	z^{46}	z^{47}	z^{48}	z^{49}	z^{50}	z^{51}	z^{52}	z^{53}
H_{21}	$h_{21}{\cdot}x_0$	$h_{21}{\cdot}x_1$	$h_{21}{\cdot}x_2$	$h_{21}{\cdot}x_3$	$h_{21}{\cdot}x_4$	$h_{21}{\cdot}x_5$	$h_{21}{\cdot}x_6$	$h_{21}{\cdot}x_7$	$h_{21}{\cdot}x_8$	$h_{21}{\cdot}x_9$	$h_{21}{\cdot}x_{10}$	$h_{21}{\cdot}x_{11}$	$h_{21}{\cdot}x_{12}$	$h_{21}{\cdot}x_{13}$																			
H_{22}		$h_{22}{\cdot}x_0$	$h_{22}{\cdot}x_1$	$h_{22}{\cdot}x_2$	$h_{22}{\cdot}x_3$	$h_{22}{\cdot}x_4$	$h_{22}{\cdot}x_5$	$h_{22}{\cdot}x_6$	$h_{22}{\cdot}x_7$	$h_{22}{\cdot}x_8$	$h_{22}{\cdot}x_9$	$h_{22}{\cdot}x_{10}$	$h_{22}{\cdot}x_{11}$	$h_{22}{\cdot}x_{12}$	$h_{22}{\cdot}x_{13}$																			
H_{23}			$h_{23}{\cdot}x_0$	$h_{23}{\cdot}x_1$	$h_{23}{\cdot}x_2$	$h_{23}{\cdot}x_3$	$h_{23}{\cdot}x_4$	$h_{23}{\cdot}x_5$	$h_{23}{\cdot}x_6$	$h_{23}{\cdot}x_7$	$h_{23}{\cdot}x_8$	$h_{23}{\cdot}x_9$	$h_{23}{\cdot}x_{10}$	$h_{23}{\cdot}x_{11}$	$h_{23}{\cdot}x_{12}$	$h_{23}{\cdot}x_{13}$																		
H_{24}				$h_{24}{\cdot}x_0$	$h_{24}{\cdot}x_1$	$h_{24}{\cdot}x_2$	$h_{24}{\cdot}x_3$	$h_{24}{\cdot}x_4$	$h_{24}{\cdot}x_5$	$h_{24}{\cdot}x_6$	$h_{24}{\cdot}x_7$	$h_{24}{\cdot}x_8$	$h_{24}{\cdot}x_9$	$h_{24}{\cdot}x_{10}$	$h_{24}{\cdot}x_{11}$	$h_{24}{\cdot}x_{12}$	$h_{24}{\cdot}x_{13}$																	
H_{25}					$h_{25}{\cdot}x_0$	$h_{25}{\cdot}x_1$	$h_{25}{\cdot}x_2$	$h_{25}{\cdot}x_3$	$h_{25}{\cdot}x_4$	$h_{25}{\cdot}x_5$	$h_{25}{\cdot}x_6$	$h_{25}{\cdot}x_7$	$h_{25}{\cdot}x_8$	$h_{25}{\cdot}x_9$	$h_{25}{\cdot}x_{10}$	$h_{25}{\cdot}x_{11}$	$h_{25}{\cdot}x_{12}$	$h_{25}{\cdot}x_{13}$																
H_{26}						$h_{26}{\cdot}x_0$	$h_{26}{\cdot}x_1$	$h_{26}{\cdot}x_2$	$h_{26}{\cdot}x_3$	$h_{26}{\cdot}x_4$	$h_{26}{\cdot}x_5$	$h_{26}{\cdot}x_6$	$h_{26}{\cdot}x_7$	$h_{26}{\cdot}x_8$	$h_{26}{\cdot}x_9$	$h_{26}{\cdot}x_{10}$	$h_{26}{\cdot}x_{11}$	$h_{26}{\cdot}x_{12}$	$h_{26}{\cdot}x_{13}$															
H_{27}							$h_{27}{\cdot}x_0$	$h_{27}{\cdot}x_1$	$h_{27}{\cdot}x_2$	$h_{27}{\cdot}x_3$	$h_{27}{\cdot}x_4$	$h_{27}{\cdot}x_5$	$h_{27}{\cdot}x_6$	$h_{27}{\cdot}x_7$	$h_{27}{\cdot}x_8$	$h_{27}{\cdot}x_9$	$h_{27}{\cdot}x_{10}$	$h_{27}{\cdot}x_{11}$	$h_{27}{\cdot}x_{12}$	$h_{27}{\cdot}x_{13}$													
H_{28}								$h_{28}{\cdot}x_0$	$h_{28}{\cdot}x_1$	$h_{28}{\cdot}x_2$	$h_{28}{\cdot}x_3$	$h_{28}{\cdot}x_4$	$h_{28}{\cdot}x_5$	$h_{28}{\cdot}x_6$	$h_{28}{\cdot}x_7$	$h_{28}{\cdot}x_8$	$h_{28}{\cdot}x_9$	$h_{28}{\cdot}x_{10}$	$h_{28}{\cdot}x_{11}$	$h_{28}{\cdot}x_{12}$	$h_{28}{\cdot}x_{13}$												
H_{29}									$h_{29}{\cdot}x_0$	$h_{29}{\cdot}x_1$	$h_{29}{\cdot}x_2$	$h_{29}{\cdot}x_3$	$h_{29}{\cdot}x_4$	$h_{29}{\cdot}x_5$	$h_{29}{\cdot}x_6$	$h_{29}{\cdot}x_7$	$h_{29}{\cdot}x_8$	$h_{29}{\cdot}x_9$	$h_{29}{\cdot}x_{10}$	$h_{29}{\cdot}x_{11}$	$h_{29}{\cdot}x_{12}$	$h_{29}{\cdot}x_{13}$										
H_{30}										$h_{30}{\cdot}x_0$	$h_{30}{\cdot}x_1$	$h_{30}{\cdot}x_2$	$h_{30}{\cdot}x_3$	$h_{30}{\cdot}x_4$	$h_{30}{\cdot}x_5$	$h_{30}{\cdot}x_6$	$h_{30}{\cdot}x_7$	$h_{30}{\cdot}x_8$	$h_{30}{\cdot}x_9$	$h_{30}{\cdot}x_{10}$	$h_{30}{\cdot}x_{11}$	$h_{30}{\cdot}x_{12}$	$h_{30}{\cdot}x_{13}$								
H_{31}											$h_{31}{\cdot}x_0$	$h_{31}{\cdot}x_1$	$h_{31}{\cdot}x_2$	$h_{31}{\cdot}x_3$	$h_{31}{\cdot}x_4$	$h_{31}{\cdot}x_5$	$h_{31}{\cdot}x_6$	$h_{31}{\cdot}x_7$	$h_{31}{\cdot}x_8$	$h_{31}{\cdot}x_9$	$h_{31}{\cdot}x_{10}$	$h_{31}{\cdot}x_{11}$	$h_{31}{\cdot}x_{12}$	$h_{31}{\cdot}x_{13}$							
H_{32}												$h_{32}{\cdot}x_0$	$h_{32}{\cdot}x_1$	$h_{32}{\cdot}x_2$	$h_{32}{\cdot}x_3$	$h_{32}{\cdot}x_4$	$h_{32}{\cdot}x_5$	$h_{32}{\cdot}x_6$	$h_{32}{\cdot}x_7$	$h_{32}{\cdot}x_8$	$h_{32}{\cdot}x_9$	$h_{32}{\cdot}x_{10}$	$h_{32}{\cdot}x_{11}$	$h_{32}{\cdot}x_{12}$	$h_{32}{\cdot}x_{13}$						
H_{33}													$h_{33}{\cdot}x_0$	$h_{33}{\cdot}x_1$	$h_{33}{\cdot}x_2$	$h_{33}{\cdot}x_3$	$h_{33}{\cdot}x_4$	$h_{33}{\cdot}x_5$	$h_{33}{\cdot}x_6$	$h_{33}{\cdot}x_7$	$h_{33}{\cdot}x_8$	$h_{33}{\cdot}x_9$	$h_{33}{\cdot}x_{10}$	$h_{33}{\cdot}x_{11}$	$h_{33}{\cdot}x_{12}$	$h_{33}{\cdot}x_{13}$						
H_{34}														$h_{34}{\cdot}x_0$	$h_{34}{\cdot}x_1$	$h_{34}{\cdot}x_2$	$h_{34}{\cdot}x_3$	$h_{34}{\cdot}x_4$	$h_{34}{\cdot}x_5$	$h_{34}{\cdot}x_6$	$h_{34}{\cdot}x_7$	$h_{34}{\cdot}x_8$	$h_{34}{\cdot}x_9$	$h_{34}{\cdot}x_{10}$	$h_{34}{\cdot}x_{11}$	$h_{34}{\cdot}x_{12}$	$h_{34}{\cdot}x_{13}$					
H_{35}															$h_{35}{\cdot}x_0$	$h_{35}{\cdot}x_1$	$h_{35}{\cdot}x_2$	$h_{35}{\cdot}x_3$	$h_{35}{\cdot}x_4$	$h_{35}{\cdot}x_5$	$h_{35}{\cdot}x_6$	$h_{35}{\cdot}x_7$	$h_{35}{\cdot}x_8$	$h_{35}{\cdot}x_9$	$h_{35}{\cdot}x_{10}$	$h_{35}{\cdot}x_{11}$	$h_{35}{\cdot}x_{12}$	$h_{35}{\cdot}x_{13}$				
H_{36}																$h_{36}{\cdot}x_0$	$h_{36}{\cdot}x_1$	$h_{36}{\cdot}x_2$	$h_{36}{\cdot}x_3$	$h_{36}{\cdot}x_4$	$h_{36}{\cdot}x_5$	$h_{36}{\cdot}x_6$	$h_{36}{\cdot}x_7$	$h_{36}{\cdot}x_8$	$h_{36}{\cdot}x_9$	$h_{36}{\cdot}x_{10}$	$h_{36}{\cdot}x_{11}$	$h_{36}{\cdot}x_{12}$	$h_{36}{\cdot}x_{13}$			
H_{37}																	$h_{37}{\cdot}x_0$	$h_{37}{\cdot}x_1$	$h_{37}{\cdot}x_2$	$h_{37}{\cdot}x_3$	$h_{37}{\cdot}x_4$	$h_{37}{\cdot}x_5$	$h_{37}{\cdot}x_6$	$h_{37}{\cdot}x_7$	$h_{37}{\cdot}x_8$	$h_{37}{\cdot}x_9$	$h_{37}{\cdot}x_{10}$	$h_{37}{\cdot}x_{11}$	$h_{37}{\cdot}x_{12}$	$h_{37}{\cdot}x_{13}$		
H_{38}																		$h_{38}{\cdot}x_0$	$h_{38}{\cdot}x_1$	$h_{38}{\cdot}x_2$	$h_{38}{\cdot}x_3$	$h_{38}{\cdot}x_4$	$h_{38}{\cdot}x_5$	$h_{38}{\cdot}x_6$	$h_{38}{\cdot}x_7$	$h_{38}{\cdot}x_8$	$h_{38}{\cdot}x_9$	$h_{38}{\cdot}x_{10}$	$h_{38}{\cdot}x_{11}$	$h_{38}{\cdot}x_{12}$	$h_{38}{\cdot}x_{13}$		
H_{39}																			$h_{39}{\cdot}x_0$	$h_{39}{\cdot}x_1$	$h_{39}{\cdot}x_2$	$h_{39}{\cdot}x_3$	$h_{39}{\cdot}x_4$	$h_{39}{\cdot}x_5$	$h_{39}{\cdot}x_6$	$h_{39}{\cdot}x_7$	$h_{39}{\cdot}x_8$	$h_{39}{\cdot}x_9$	$h_{39}{\cdot}x_{10}$	$h_{39}{\cdot}x_{11}$	$h_{39}{\cdot}x_{12}$	$h_{39}{\cdot}x_{13}$		
H_{40}																				$h_{40}{\cdot}x_0$	$h_{40}{\cdot}x_1$	$h_{40}{\cdot}x_2$	$h_{40}{\cdot}x_3$	$h_{40}{\cdot}x_4$	$h_{40}{\cdot}x_5$	$h_{40}{\cdot}x_6$	$h_{40}{\cdot}x_7$	$h_{40}{\cdot}x_8$	$h_{40}{\cdot}x_9$	$h_{40}{\cdot}x_{10}$	$h_{40}{\cdot}x_{11}$	$h_{40}{\cdot}x_{12}$	$h_{40}{\cdot}x_{13}$	

Tabla 5: Análisis temporal de la arquitectura polifásica de un filtro RC desde H0 hasta H20

Por tanto, las ecuaciones de salida, obtenidas a partir de la Tabla 4 y de la Tabla 5, son las siguientes:

y0 = H0· x0 + (H1· x13 + H2· x12 + H3· x11 + H4· x10 + H5· x9 + H6· x8 + H7· x7 + H8· x6 + H9· x5 + H10· x4 + H11· x3 + H12· x2 + H13· x1 + H14· x0)· z^−1 + (H15· x13 + H16· x12 + H17· x11 + H18· x10 + H19· x9 + H20· x8 + H21· x7 + H22· x6 + H23· x5 + H24· x4 + H25· x3 + H26· x2 + H27· x1 + H28· x0)· z^−2 + (H29· x13 + H30· x12 + H31· x11 + H32· x10 + H33· x9 + H34· x8 + H35· x7 + H36· x6 + H37· x5 + H38· x4 + H39· x3 + H40· x2)· z^−3

y1 = H0· x1 + H1· x0 + (H2· x13 + H3· x12 + H4· x11 + H5· x10 + H6· x9 + H7· x8 + H8· x7 + H9· x6 + H10· x5 + H11· x4 + H12· x3 + H13· x2 + H14· x1 + H15· x0)· z^−1 + (H16· x13 + H17· x12 + H18· x11 + H19· x10 + H20· x9 + H21· x8 + H22· x7 + H23· x6 + H24· x5 + H25· x4 + H26· x3 + H27· x2 + H28· x1 + H29· x0)· z^−2 + (H30· x13 + H31· x12 + H32· x11 + H33· x10 + H34· x9 + H35· x8 + H36· x7 + H37· x6 + H38· x5 + H39· x4 + H40· x3)· z^−3

y2 = H0· x2 + H1· x1 + H2· x0 + (H3· x13 + H4· x12 + H5· x11 + H6· x10 + H7· x9 + H8· x8 + H9· x7 + H10· x6 + H11· x5 + H12· x4 + H13· x3 + H14· x2 + H15· x1 + H16· x0)· z^−1 + (H17· x13 + H18· x12 + H19· x11 + H20· x10 + H21· x9 + H22· x8 + H23· x7 + H24· x6 + H25· x5 + H26· x4 + H27· x3 + H28· x2 + H29· x1 + H30· x0)· z^−2 + (H31· x13 + H32· x12 + H33· x11 + H34· x10 + H35· x9 + H36· x8 + H37· x7 + H38· x6 + H39· x5 + H40· x4)· z^−3

y3 = H0· x3 + H1· x2 + H2· x1 + H3· x0 + (H4· x13 + H5· x12 + H6· x11 + H7· x10 + H8· x9 + H9· x8 + H10· x7 + H11· x6 + H12· x5 + H13· x4 + H14· x3 + H15· x2 + H16· x1 + H17· x0)· z^−1 + (H18· x13 + H19· x12 + H20· x11 + H21· x10 + H22· x9 + H23· x8 + H24· x7 + H25· x6 + H26· x5 + H27· x4 + H28· x3 + H29· x2 + H30· x1 + H31· x0)· z^−2 + (H32· x13 + H33· x12 + H34· x11 + H35· x10 + H36· x9 + H37· x8 + H38· x7 + H39· x6 + H40· x5)· z^−3

y4 = H0· x4 + H1· x3 + H2· x2 + H3· x1 + H4· x0 + (H5· x13 + H6· x12 + H7· x11 + H8· x10 + H9· x9 + H10· x8 + H11· x7 + H12· x6 + H13· x5 + H14· x4 + H15· x3 + H16· x2 + H17· x1 + H18· x0)· z^−1 + (H19· x13 + H20· x12 + H21· x11 + H22· x10 + H23· x9 + H24· x8 + H25· x7 + H26· x6 + H27· x5 + H28· x4 + H29· x3 + H30· x2 + H31· x1 + H32· x0)· z^−2 + (H33· x13 + H34· x12 + H35· x11 + H36· x10 + H37· x9 + H38· x8 + H39· x7 + H40· x6)· z^−3

```
y5 = H0· x5 + H1· x4 + H2· x3 + H3· x2 + H4· x1 + H5· x0 + (H6· x13 + H7· x12 + H8· x11 + H9· x10 + H10· x9 + H11· x8 + H12·
     x7 + H13· x6 + H14· x5 + H15· x4 + H16· x3 + H17· x2 + H18· x1 + H19· x0)· z^−1 + (H20· x13 + H21· x12 + H22· x11 +
     H23· x10 + H24· x9 + H25· x8 + H26· x7 + H27· x6 + H28· x5 + H29· x4 + H30· x3 + H31· x2 + H32· x1 + H33· x0)· z^−2 +
     (H34· x13 + H35· x12 + H36· x11 + H37· x10 + H38· x9 + H39· x8 + H40· x7)· z^−3

y6 = H0· x6 + H1· x5 + H2· x4 + H3· x3 + H4· x2 + H5· x1 + H6· x0 + (H7· x13 + H8· x12 + H9· x11 + H10· x10 + H11· x9 + H12·
     x8 + H13· x7 + H14· x6 + H15· x5 + H16· x4 + H17· x3 + H18· x2 + H19· x1 + H20· x0)· z^−1 + (H21· x13 + H22· x12 +
     H23· x11 + H24· x10 + H25· x9 + H26· x8 + H27· x7 + H28· x6 + H29· x5 + H30· x4 + H31· x3 + H32· x2 + H33· x1 + H34·
     x0)· z^−2 + (H35· x13 + H36· x12 + H37· x11 + H38· x10 + H39· x9 + H40· x8)· z^−3

y7 = H0· x7 + H1· x6 + H2· x5 + H3· x4 + H4· x3 + H5· x2 + H6· x1 + H7· x0 + (H8· x13 + H9· x12 + H10· x11 + H11· x10 + H12·
     x9 + H13· x8 + H14· x7 + H15· x6 + H16· x5 + H17· x4 + H18· x3 + H19· x2 + H20· x1 + H21· x0)· z^−1 + (H22· x13 + H23·
     x12 + H24· x11 + H25· x10 + H26· x9 + H27· x8 + H28· x7 + H29· x6 + H30· x5 + H31· x4 + H32· x3 + H33· x2 + H34· x1 +
     H35· x0)· z^−2 + (H36· x13 + H37· x12 + H38· x11 + H39· x10 + H40· x9)· z^−3

y8 = H0· x8 + H1· x7 + H2· x6 + H3· x5 + H4· x4 + H5· x3 + H6· x2 + H7· x1 + H8· x0 + (H9· x13 + H10· x12 + H11· x11 + H12·
     x10 + H13· x9 + H14· x8 + H15· x7 + H16· x6 + H17· x5 + H18· x4 + H19· x3 + H20· x2 + H21· x1 + H22· x0)· z^−1 + (H23·
     x13 + H24· x12 + H25· x11 + H26· x10 + H27· x9 + H28· x8 + H29· x7 + H30· x6 + H31· x5 + H32· x4 + H33· x3 + H34· x2
     + H35· x1 + H36· x0)· z^−2 + (H37· x13 + H38· x12 + H39· x11 + H40· x10)· z^−3

y9 = H0· x9 + H1· x8 + H2· x7 + H3· x6 + H4· x5 + H5· x4 + H6· x3 + H7· x2 + H8· x1 + H9· x0 + (H10· x13 + H11· x12 + H12·
     x11 + H13· x10 + H14· x9 + H15· x8 + H16· x7 + H17· x6 + H18· x5 + H19· x4 + H20· x3 + H21· x2 + H22· x1 + H23· x0)· z
     ^−1 + (H24· x13 + H25· x12 + H26· x11 + H27· x10 + H28· x9 + H29· x8 + H30· x7 + H31· x6 + H32· x5 + H33· x4 + H34·
     x3 + H35· x2 + H36· x1 + H37· x0)· z^−2 + (H38· x13 + H39· x12 + H40· x11)· z^−3

y10 = H0· x10 + H1· x9 + H2· x8 + H3· x7 + H4· x6 + H5· x5 + H6· x4 + H7· x3 + H8· x2 + H9· x1 + H10· x0 + (H11· x13 + H12·
      x12 + H13· x11 + H14· x10 + H15· x9 + H16· x8 + H17· x7 + H18· x6 + H19· x5 + H20· x4 + H21· x3 + H22· x2 + H23· x1 +
      H24· x0)· z^−1 + (H25· x13 + H26· x12 + H27· x11 + H28· x10 + H29· x9 + H30· x8 + H31· x7 + H32· x6 + H33· x5 + H34·
      x4 + H35· x3 + H36· x2 + H37· x1 + H38· x0)· z^−2 + (H39· x13 + H40· x12)· z^−3

y11 = H0· x11 + H1· x10 + H2· x9 + H3· x8 + H4· x7 + H5· x6 + H6· x5 + H7· x4 + H8· x3 + H9· x2 + H10· x1 + H11· x0 + (H12·
      x13 + H13· x12 + H14· x11 + H15· x10 + H16· x9 + H17· x8 + H18· x7 + H19· x6 + H20· x5 + H21· x4 + H22· x3 + H23· x2
      + H24· x1 + H25· x0)· z^−1 + (H26· x13 + H27· x12 + H28· x11 + H29· x10 + H30· x9 + H31· x8 + H32· x7 + H33· x6 +
      H34· x5 + H35· x4 + H36· x3 + H37· x2 + H38· x1 + H39· x0)· z^−2 + H40· x13· z^−3

y12 = H0· x12 + H1· x11 + H2· x10 + H3· x9 + H4· x8 + H5· x7 + H6· x6 + H7· x5 + H8· x4 + H9· x3 + H10· x2 + H11· x1 + H12·
      x0 + (H13· x13 + H14· x12 + H15· x11 + H16· x10 + H17· x9 + H18· x8 + H19· x7 + H20· x6 + H21· x5 + H22· x4 + H23· x3
      + H24· x2 + H25· x1 + H26· x0)· z^−1 + (H27· x13 + H28· x12 + H29· x11 + H30· x10 + H31· x9 + H32· x8 + H33· x7 +
      H34· x6 + H35· x5 + H36· x4 + H37· x3 + H38· x2 + H39· x1 + H40· x0)· z^−2

y13 = H0· x13 + H1· x12 + H2· x11 + H3· x10 + H4· x9 + H5· x8 + H6· x7 + H7· x6 + H8· x5 + H9· x4 + H10· x3 + H11· x2 + H12·
      x1 + H13· x0 + (H14· x13 + H15· x12 + H16· x11 + H17· x10 + H18· x9 + H19· x8 + H20· x7 + H21· x6 + H22· x5 + H23· x4
      + H24· x3 + H25· x2 + H26· x1 + H27· x0)· z^−1 + (H28· x13 + H29· x12 + H30· x11 + H31· x10 + H32· x9 + H33· x8 +
      H34· x7 + H35· x6 + H36· x5 + H37· x4 + H38· x3 + H39· x2 + H40· x1)· z^−2
```

Dado que solo disponemos de 14 entradas, deberemos agrupar los coeficientes en 14 grupos:

```
H0 = h0 + h14 + h28
H1 = h1 + h15 + h29
H2 = h2 + h16 + h30
H3 = h3 + h17 + h31
H4 = h4 + h18 + h32
H5 = h5 + h19 + h33
H6 = h6 + h20 + h34
```

```
H7 = h7 + h21 + h35
H8 = h8 + h22 + h36
H9 = h9 + h23 + h37
H10 = h10 + h24 + h38
H11 = h11 + h25 + h39
H12 = h12 + h26 + h40
H13 = h13 + h27
```

En Matlab deberemos hacer lo mismo, agrupar los coeficientes en arrays, tal y como se ve en la Figura 24.

```
%% Coeficientes del filtro de coseno alzado agrupados
hrrc_ri_id = hrrc_ri_id';
H0 = [hrrc_ri_id(1,1) hrrc_ri_id(15,1) hrrc_ri_id(29,1)];
H1 = [hrrc_ri_id(2,1) hrrc_ri_id(16,1) hrrc_ri_id(30,1)];
H2 = [hrrc_ri_id(3,1) hrrc_ri_id(17,1) hrrc_ri_id(31,1)];
H3 = [hrrc_ri_id(4,1) hrrc_ri_id(18,1) hrrc_ri_id(32,1)];
H4 = [hrrc_ri_id(5,1) hrrc_ri_id(19,1) hrrc_ri_id(33,1)];
H5 = [hrrc_ri_id(6,1) hrrc_ri_id(20,1) hrrc_ri_id(34,1)];
H6 = [hrrc_ri_id(7,1) hrrc_ri_id(21,1) hrrc_ri_id(35,1)];
H7 = [hrrc_ri_id(8,1) hrrc_ri_id(22,1) hrrc_ri_id(36,1)];
H8 = [hrrc_ri_id(9,1) hrrc_ri_id(23,1) hrrc_ri_id(37,1)];
H9 = [hrrc_ri_id(10,1) hrrc_ri_id(24,1) hrrc_ri_id(38,1)];
H10 = [hrrc_ri_id(11,1) hrrc_ri_id(25,1) hrrc_ri_id(39,1)];
H11 = [hrrc_ri_id(12,1) hrrc_ri_id(26,1) hrrc_ri_id(40,1)];
H12 = [hrrc_ri_id(13,1) hrrc_ri_id(27,1) hrrc_ri_id(41,1)];
H13 = [hrrc_ri_id(14,1) hrrc_ri_id(28,1)];
```

Figura 24: Agrupación de coeficientes en Matlab

Por tanto, con esta información, debemos cablear los 14 subfiltros tal y como se muestra en la Figura 25, la cual se corresponde con el subfiltro 0.

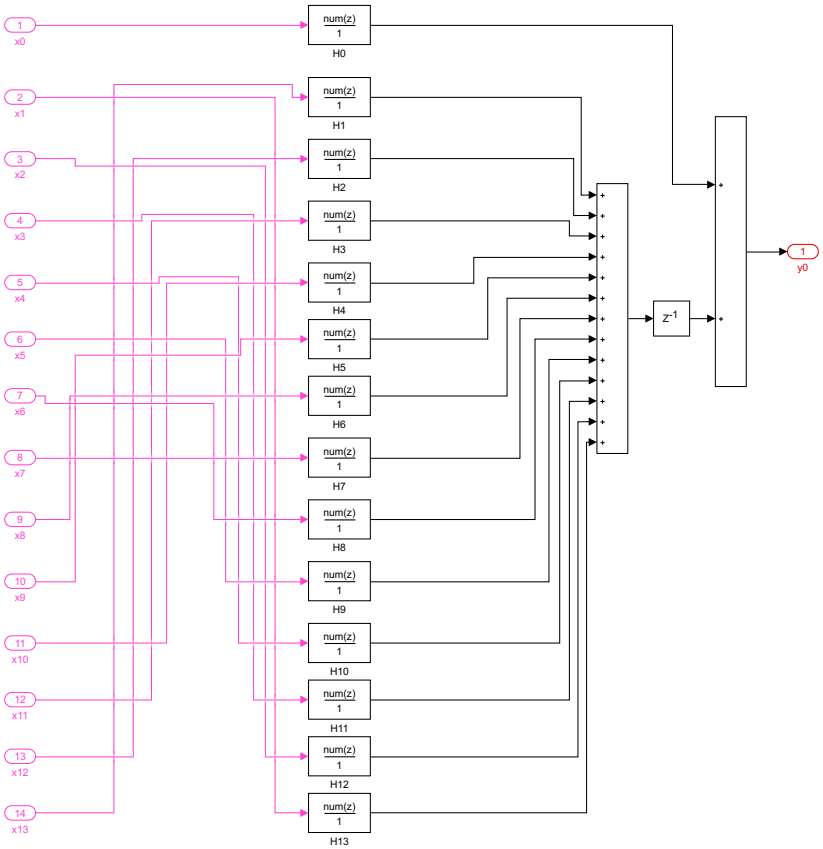

Figura 25: Esquema del subfiltro 0 (salida Y0)

A partir de esta imagen, se puede observar que cada subfiltro está compuesto por tres elementos:

filtros FIR (Discrete FIR Filter), sumadores y retardos. Cada grupo de coeficientes está asociado a un filtro FIR en concreto.

Cabe destacar que cada bloque de estos es equivalente al esquema de la Figura 26, el cual depende del número de coeficientes. En nuestro caso, el filtro RC está compuesto en su mayoría por filtros FIR de tres coeficientes, aunque el último FIR de cada subfiltro está compuesto únicamente por 2.

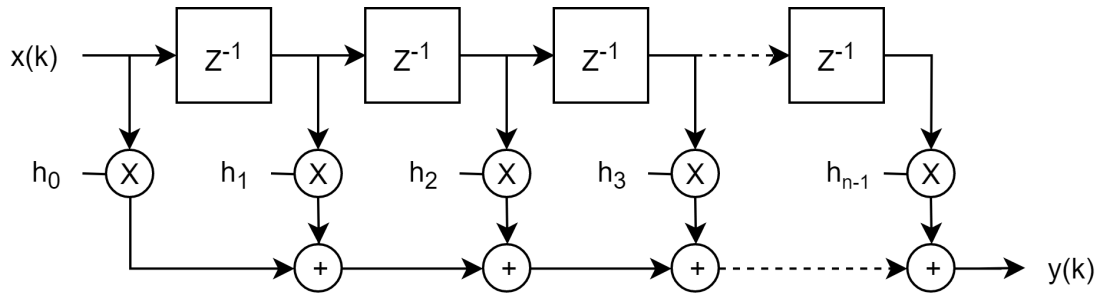

Figura 26: Filtro FIR de n coeficientes

Una vez que tenemos todos los subfiltros los conectamos entre sí (Figura 27).

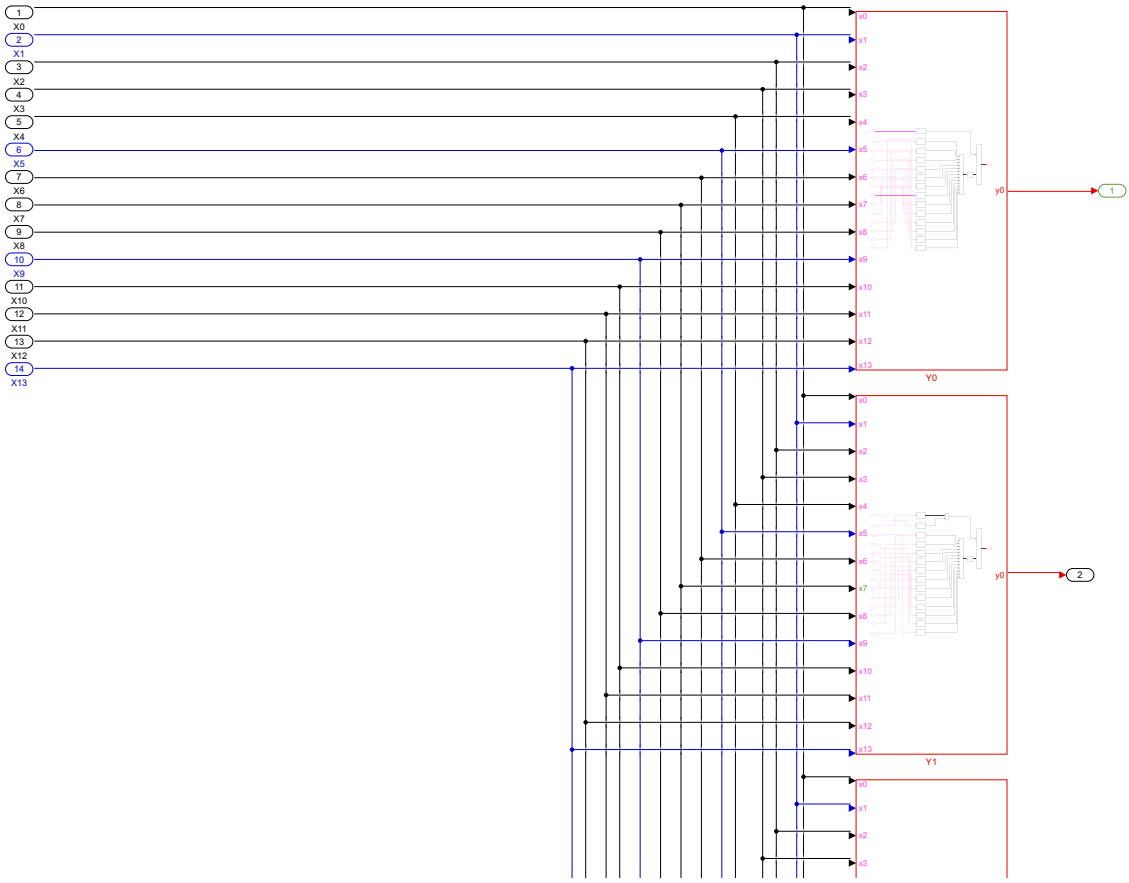

Figura 27: Parte de la interconexión entre subfiltros de un filtro RC polifásico

Tal y como se observa en la anterior imagen, los subfiltros son independientes entre sí, y dependen únicamente de las entradas.

Finalmente, para poder comprobar el funcionamiento de este filtro, deberemos excitar la entrada, aunque, para ello, primero deberemos paralelizarla. Si antes, al ser no paralelizada, disponíamos de una única entrada cada ciclo de reloj, ahora deberemos de proveer de 14 entradas cada 1/14 ciclos de reloj. Para ello haremos uso de diezmadores que permitan seleccionar una muestra cada 14 ciclos. Si combinamos esto con retardos, podemos hacer que cada entrada se encargue de procesar una muestra cada 14, es decir, la entrada 0 procesará la muestra 0, 13, 26, etc., la entrada 1 la muestra 1, 14, 27, etc., tal y como se muestra en la Figura 28.

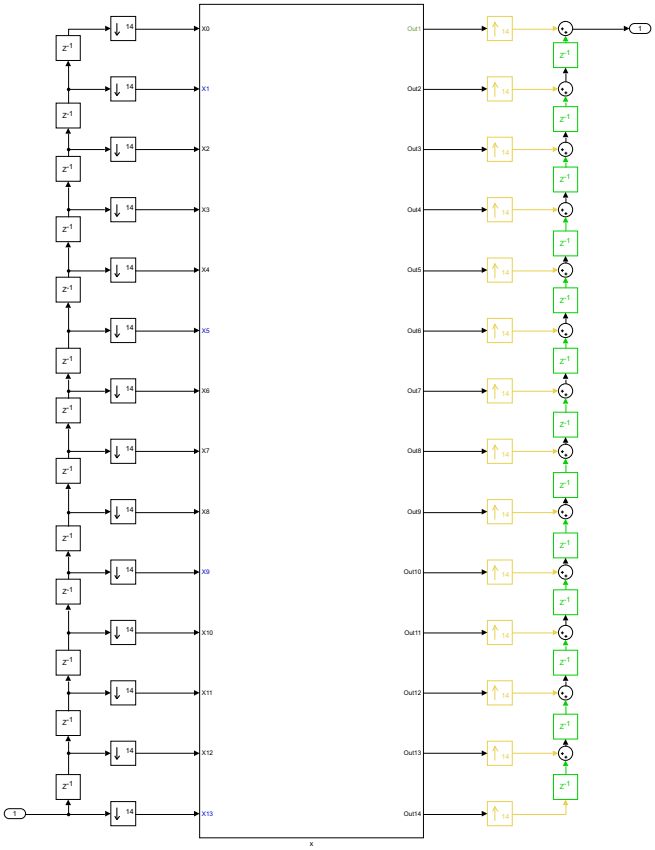

Figura 28: Filtro RC con arquitectura polifásica

Cabe destacar que esta última parte no estará en el modelo final, esto se ha realizado para comparar los modelos.

Para comprobar que está correctamente implementado, lo comparamos con el modelo no paralelizado que ya teníamos hecho (Figura 29).

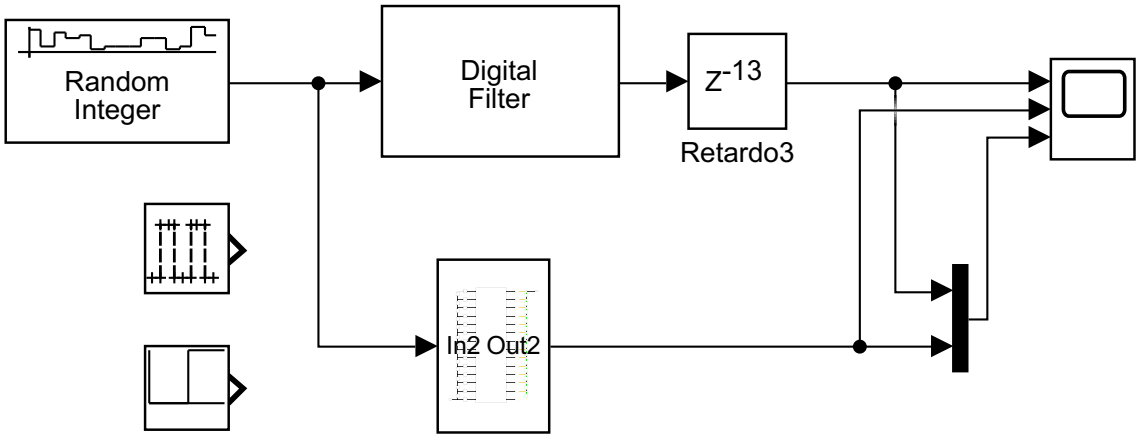

Figura 29: Comparación entre el filtro RC paralelo y no paralelo

En la Figura 30 se puede observar las salidas de ambos modelos. La primera se corresponde con la arquitectura no paralela, la segunda con la arquitectura polifásica y la tercera son las dos a la vez.

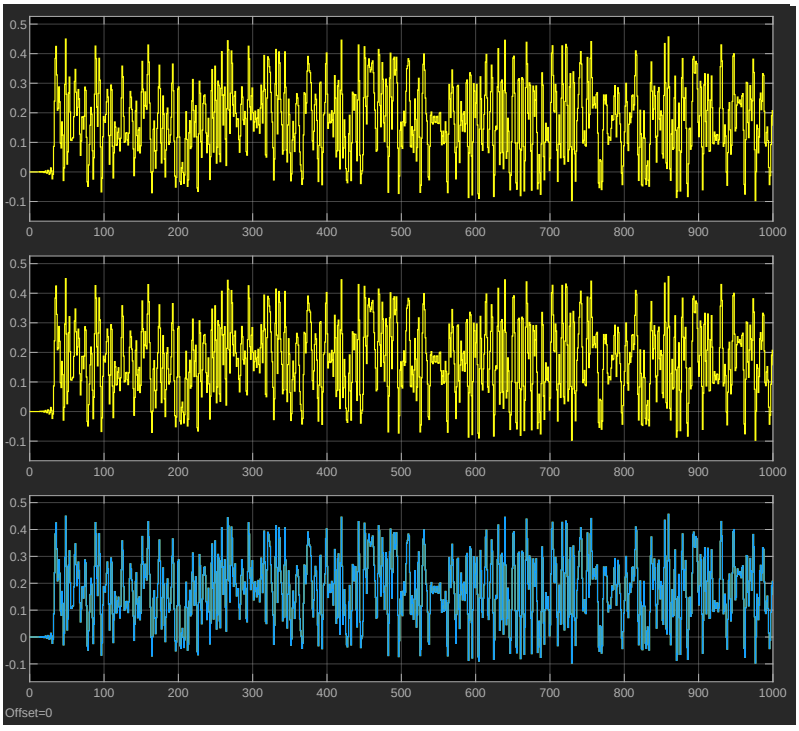

Figura 30: Gráfica comparativa entre la arquitectura no paralela y la paralela

3.1.2. Filtro de cambio de tasa por 7/8

En cuanto al filtro diezmador, el cálculo de coeficientes se ha realizado mediante el siguiente comando de Matlab:

firpm(n,f,a)

Donde:

- n: Orden del filtro
- f: Vector de los límites de la banda de frecuencia
- a: Vector de la amplitud en cada límite de f

En la Figura 31 puede verse la configuración de dichos coeficientes.

```
%% Coeficientes filtro diezmador

SPStotal = 16/7; % = 2.2857
Roll_mas_1 = SPStotal/2; % = 1.1429
Fmax = Roll_mas_1 / (2* 2*8);
orden_fir =48; % orden par

fir87=firpm(orden_fir,[0 Fmax 1/8-Fmax .5]*2,[1 1 0 0]);

h87=fir87*8;
h78_id=fir87*7;
```

Figura 31: Configuración del filtro diezmador

Tal y como se aprecia, el orden del filtro es de 48, por tanto dispondrá de 49 coeficientes.

La implementación de este filtro con arquitectura no paralela, es igual de simple que el filtro de coseno alzado, solo que en esta ocasión tendremos que añadir el bloque interpolador y el diezmador (Figura 32).

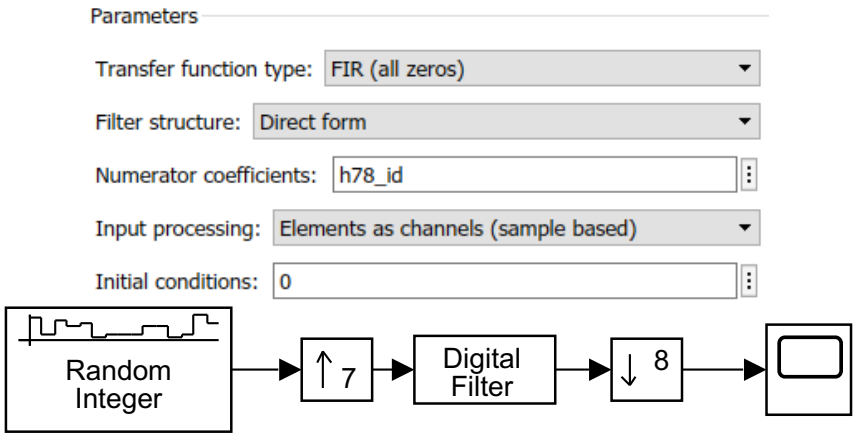

Figura 32: Filtro diezmador no paralelizado y configuración

Para aplicar la transformación polifásica, deberemos seguir los pasos descritos en el capítulo 2. Por tanto, lo primero que debemos hacer es identificar nuestros parámetros, tales como el factor de interpolación (L), el factor de diezmado (M), número de entradas paralelizadas (P_i) y número de salidas (P_o). En nuestro caso los parámetros fijos son los siguientes:

- $L = 7$
- $M = 8$
- $P_i = 16$

Por tanto, a partir de (6) podemos obtener el número de salidas de nuestro sistema:

$$P_o = \frac{7}{8} \cdot 16 = 14 \tag{9}$$

El número total de salidas de nuestro filtro de cambio de tasa implementado con una arquitectura polifásica será de 14.

Por tanto, siguiendo los pasos anteriormente explicados, debemos implementar en Simulink el esquema de la Figura 19, donde dispondremos de un total de 7 subfiltros, cuya implementación viene determinada por las ecuaciones de salidas obtenidas a partir de la Tabla 6.

	z^0	z^1	z^2	z^3	z^4	z^5	z^6	z^7	z^8	z^9	z^{10}	z^{11}	z^{12}	z^{13}	z^{14}	z^{15}	z^{16}	z^{17}	z^{18}	z^{19}	z^{20}	z^{21}
H_0	$h_0 \cdot x_0$	$h_0 \cdot x_1$	$h_0 \cdot x_2$	$h_0 \cdot x_3$	$h_0 \cdot x_4$	$h_0 \cdot x_5$	$h_0 \cdot x_6$	$h_0 \cdot x_7$	$h_0 \cdot x_8$	$h_0 \cdot x_9$	$h_0 \cdot x_{10}$	$h_0 \cdot x_{11}$	$h_0 \cdot x_{12}$	$h_0 \cdot x_{13}$	$h_0 \cdot x_{14}$	$h_0 \cdot x_{15}$						
H_1		$h_1 \cdot x_0$	$h_1 \cdot x_1$	$h_1 \cdot x_2$	$h_1 \cdot x_3$	$h_1 \cdot x_4$	$h_1 \cdot x_5$	$h_1 \cdot x_6$	$h_1 \cdot x_7$	$h_1 \cdot x_8$	$h_1 \cdot x_9$	$h_1 \cdot x_{10}$	$h_1 \cdot x_{11}$	$h_1 \cdot x_{12}$	$h_1 \cdot x_{13}$	$h_1 \cdot x_{14}$	$h_1 \cdot x_{15}$					
H_2			$h_2 \cdot x_0$	$h_2 \cdot x_1$	$h_2 \cdot x_2$	$h_2 \cdot x_3$	$h_2 \cdot x_4$	$h_2 \cdot x_5$	$h_2 \cdot x_6$	$h_2 \cdot x_7$	$h_2 \cdot x_8$	$h_2 \cdot x_9$	$h_2 \cdot x_{10}$	$h_2 \cdot x_{11}$	$h_2 \cdot x_{12}$	$h_2 \cdot x_{13}$	$h_2 \cdot x_{14}$	$h_2 \cdot x_{15}$				
H_3				$h_3 \cdot x_0$	$h_3 \cdot x_1$	$h_3 \cdot x_2$	$h_3 \cdot x_3$	$h_3 \cdot x_4$	$h_3 \cdot x_5$	$h_3 \cdot x_6$	$h_3 \cdot x_7$	$h_3 \cdot x_8$	$h_3 \cdot x_9$	$h_3 \cdot x_{10}$	$h_3 \cdot x_{11}$	$h_3 \cdot x_{12}$	$h_3 \cdot x_{13}$	$h_3 \cdot x_{14}$	$h_3 \cdot x_{15}$			
H_4					$h_4 \cdot x_0$	$h_4 \cdot x_1$	$h_4 \cdot x_2$	$h_4 \cdot x_3$	$h_4 \cdot x_4$	$h_4 \cdot x_5$	$h_4 \cdot x_6$	$h_4 \cdot x_7$	$h_4 \cdot x_8$	$h_4 \cdot x_9$	$h_4 \cdot x_{10}$	$h_4 \cdot x_{11}$	$h_4 \cdot x_{12}$	$h_4 \cdot x_{13}$	$h_4 \cdot x_{14}$	$h_4 \cdot x_{15}$		
H_5						$h_5 \cdot x_0$	$h_5 \cdot x_1$	$h_5 \cdot x_2$	$h_5 \cdot x_3$	$h_5 \cdot x_4$	$h_5 \cdot x_5$	$h_5 \cdot x_6$	$h_5 \cdot x_7$	$h_5 \cdot x_8$	$h_5 \cdot x_9$	$h_5 \cdot x_{10}$	$h_5 \cdot x_{11}$	$h_5 \cdot x_{12}$	$h_5 \cdot x_{13}$	$h_5 \cdot x_{14}$	$h_5 \cdot x_{15}$	
H_6							$h_6 \cdot x_0$	$h_6 \cdot x_1$	$h_6 \cdot x_2$	$h_6 \cdot x_3$	$h_6 \cdot x_4$	$h_6 \cdot x_5$	$h_6 \cdot x_6$	$h_6 \cdot x_7$	$h_6 \cdot x_8$	$h_6 \cdot x_9$	$h_6 \cdot x_{10}$	$h_6 \cdot x_{11}$	$h_6 \cdot x_{12}$	$h_6 \cdot x_{13}$	$h_6 \cdot x_{14}$	$h_6 \cdot x_{15}$

Tabla 6: Análisis temporal de la arquitectura polifásica de un filtro de cambio de tasa

A partir de la tabla anterior, las ecuaciones de salida son las siguientes:

```
y0 = H_X0· x0 + (H_X1· x15 + H_X2· x14 + H_X3· x13 + H_X4· x12 + H_X5· x11 + H_X6· x10)· z^−1
y1 = H_X0· x1 + H_X1· x0 + (H_X2· x15 + H_X3· x14 + H_X4· x13 + H_X5· x12 + H_X6· x11)· z^−1
y2 = H_X0· x2 + H_X1· x1 + H_X2· x0 + (H_X3· x15 + H_X4· x14 + H_X5· x13 + H_X6· x12)· z^−1
y3 = H_X0· x3 + H_X1· x2 + H_X2· x1 + H_X3· x0 + (H_X4· x15 + H_X5· x14 + H_X6· x13)· z^−1
y4 = H_X0· x4 + H_X1· x3 + H_X2· x2 + H_X3· x1 + H_X4· x0 + (H_X5· x15 + H_X6· x14)· z^−1
y5 = H_X0· x5 + H_X1· x4 + H_X2· x3 + H_X3· x2 + H_X4· x1 + H_X5· x0 + H_X6· x15· z^−1
y6 = H_X0· x6 + H_X1· x5 + H_X2· x4 + H_X3· x3 + H_X4· x2 + H_X5· x1 + H_X6· x0
y7 = H_X0· x7 + H_X1· x6 + H_X2· x5 + H_X3· x4 + H_X4· x3 + H_X5· x2 + H_X6· x1
y8 = H_X0· x8 + H_X1· x7 + H_X2· x6 + H_X3· x5 + H_X4· x4 + H_X5· x3 + H_X6· x2
y9 = H_X0· x9 + H_X1· x8 + H_X2· x7 + H_X3· x6 + H_X4· x5 + H_X5· x4 + H_X6· x3
y10 = H_X0· x10 + H_X1· x9 + H_X2· x8 + H_X3· x7 + H_X4· x6 + H_X5· x5 + H_X6· x4
y11 = H_X0· x11 + H_X1· x10 + H_X2· x9 + H_X3· x8 + H_X4· x7 + H_X5· x6 + H_X6· x5
y12 = H_X0· x12 + H_X1· x11 + H_X2· x10 + H_X3· x9 + H_X4· x8 + H_X5· x7 + H_X6· x6
y13 = H_X0· x13 + H_X1· x12 + H_X2· x11 + H_X3· x10 + H_X4· x9 + H_X5· x8 + H_X6· x7
y14 = H_X0· x14 + H_X1· x13 + H_X2· x12 + H_X3· x11 + H_X4· x10 + H_X5· x9 + H_X6· x8
y15 = H_X0· x15 + H_X1· x14 + H_X2· x13 + H_X3· x12 + H_X4· x11 + H_X5· x10 + H_X6· x9
```

Donde X implica el número del subfiltro en el que estemos. Los subfiltros son exactamente iguales entre sí, la única diferencia son los coeficientes de cada filtro FIR. Estos coeficientes están agrupados de la siguiente forma:

```
H0 = (h0, h7, h14, h21, h28, h35, h42)
H1 = (h1, h8, h15, h22, h29, h36, h43)
H2 = (h2, h9, h16, h23, h30, h37, h44)
H3 = (h3, h10, h17, h24, h31, h38, h45)
H4 = (h4, h11, h18, h25, h32, h39, h46)
H5 = (h5, h12, h19, h26, h33, h40, h47)
H6 = (h6, h13, h20, h27, h34, h41, h48)
```

Cada uno de estos grupos está vinculado a un subfiltro en concreto. Si volvemos a agrupar estos grupos dentro de cada subfiltro obtenemos lo siguiente:

H0	**H1**	**H2**	**H3**	**H4**	**H5**	**H6**
H_00 = h0	H_10 = h1	H_20 = h2	H_30 = h3	H_40 = h4	H_50 = h5	H_60 = h6
H_01 = h7	H_11 = h8	H_21 = h9	H_31 = h10	H_41 = h11	H_51 = h12	H_61 = h13
H_02 = h14	H_12 = h15	H_22 = h16	H_32 = h17	H_42 = h18	H_52 = h19	H_62 = h20
H_03 = h21	H_13 = h22	H_23 = h23	H_33 = h24	H_43 = h25	H_53 = h26	H_63 = h27
H_04 = h28	H_14 = h29	H_24 = h30	H_34 = h31	H_44 = h32	H_54 = h33	H_64 = h34
H_05 = h35	H_15 = h36	H_25 = h37	H_35 = h38	H_45 = h39	H_55 = h40	H_65 = h41
H_06 = h42	H_16 = h43	H_26 = h44	H_36 = h45	H_46 = h46	H_56 = h47	H_66 = h48

Tal y como se observa, cada filtro FIR estará compuesto por un único coeficiente. Esto se ha realizado de la misma forma en Matlab (Figura 33).

```
%% Subfiltros 7−8
h_1 = [h78(1) h78(8) h78(15) h78(22) h78(29) h78(36) h78(43)];
h_01 = [h78(1)];
h_02 = [h78(8)];
h_03 = [h78(15)];
h_04 = [h78(22)];
h_05 = [h78(29)];
h_06 = [h78(36)];
h_07 = [h78(43)];

h_2 = [h78(2) h78(9) h78(16) h78(23) h78(30) h78(37) h78(44)];
h_11 = [h78(2)];
h_12 = [h78(9)];
h_13 = [h78(16)];
h_14 = [h78(23)];
h_15 = [h78(30)];
h_16 = [h78(37)];
h_17 = [h78(44)];

h_3 = [h78(3) h78(10) h78(17) h78(24) h78(31) h78(38) h78(45)];
h_21 = [h78(3)];
h_22 = [h78(10)];
h_23 = [h78(17)];
h_24 = [h78(24)];
h_25 = [h78(31)];
h_26 = [h78(38)];
h_27 = [h78(45)];

h_4 = [h78(4) h78(11) h78(18) h78(25) h78(32) h78(39) h78(46)];
h_31 = [h78(4)];
h_32 = [h78(11)];
h_33 = [h78(18)];
h_34 = [h78(25)];
h_35 = [h78(32)];
h_36 = [h78(39)];
h_37 = [h78(46)];

h_5 = [h78(5) h78(12) h78(19) h78(26) h78(33) h78(40) h78(47)];
h_41 = [h78(5)];
h_42 = [h78(12)];
h_43 = [h78(19)];
h_44 = [h78(26)];
h_45 = [h78(33)];
h_46 = [h78(40)];
h_47 = [h78(47)];

h_6 = [h78(6) h78(13) h78(20) h78(27) h78(34) h78(41) h78(48)];
h_51 = [h78(6)];
h_52 = [h78(13)];
h_53 = [h78(20)];
h_54 = [h78(27)];
h_55 = [h78(34)];
h_56 = [h78(41)];
h_57 = [h78(48)];

h_7 = [h78(7) h78(14) h78(21) h78(28) h78(35) h78(42) h78(49)];
h_61 = [h78(7)];
h_62 = [h78(14)];
h_63 = [h78(21)];
h_64 = [h78(28)];
h_65 = [h78(35)];
h_66 = [h78(42)];
h_67 = [h78(49)];
```

Figura 33: Agrupación de coeficientes en Matlab

Por tanto, el equivalente a la Figura 19 para nuestro filtro sería el visto en la Figura 34.

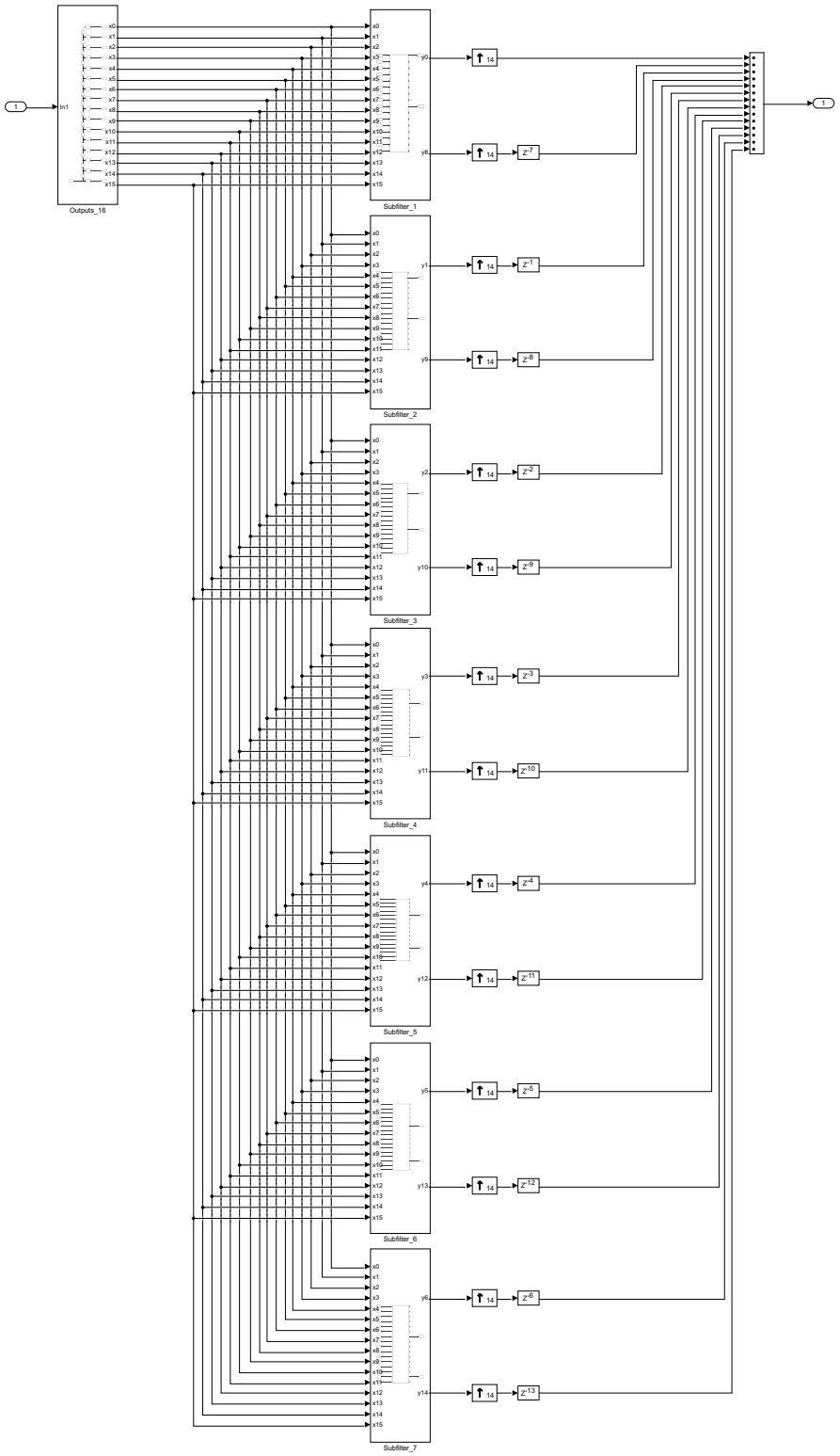

Figura 34: Esquema general del filtro de cambio de tasa con arquitectura paralela

En ella se puede observar como la entrada es paralelizada por 16 y alimenta a los 7 subfiltros. Finalmente, de estos salen 14 salidas.

El interior de uno de los subfiltros se muestra en la Figura 35. Tal y como se ha dicho anteriormente la única diferencia entre ellos son los coeficientes de los filtros FIR.

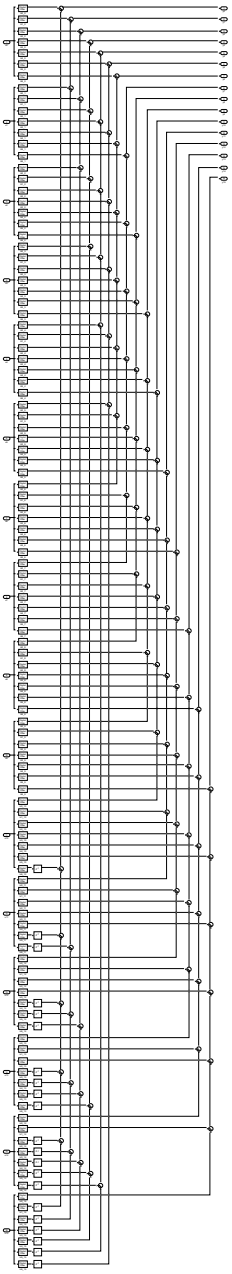

Figura 35: Esquema interno del subfiltro

Una vez llegados a este punto, debemos realizar el análisis temporal que hicimos en la Figura 21, pero aplicado a nuestro filtro. A partir de esta imagen podemos deducir que cada subfiltro deberá sacar las siguientes salidas:

- Subfiltro 1: {y0 y8}
- Subfiltro 2: {y1 y9}
- Subfiltro 3: {y2 y10}
- Subfiltro 4: {y3 y11}
- Subfiltro 5: {y4 y12}
- Subfiltro 6: {y5 y13}
- Subfiltro 7: {y6 y14}

Las salidas y_7 e y_{15} no son necesarias.

Por último, y con el fin de mantener la tasa de muestreo exigida, debemos interpolar la salida por L, en este caso 14 y adicionalmente añadimos el retardo correspondiente a cada muestra de salida (Figura 36).

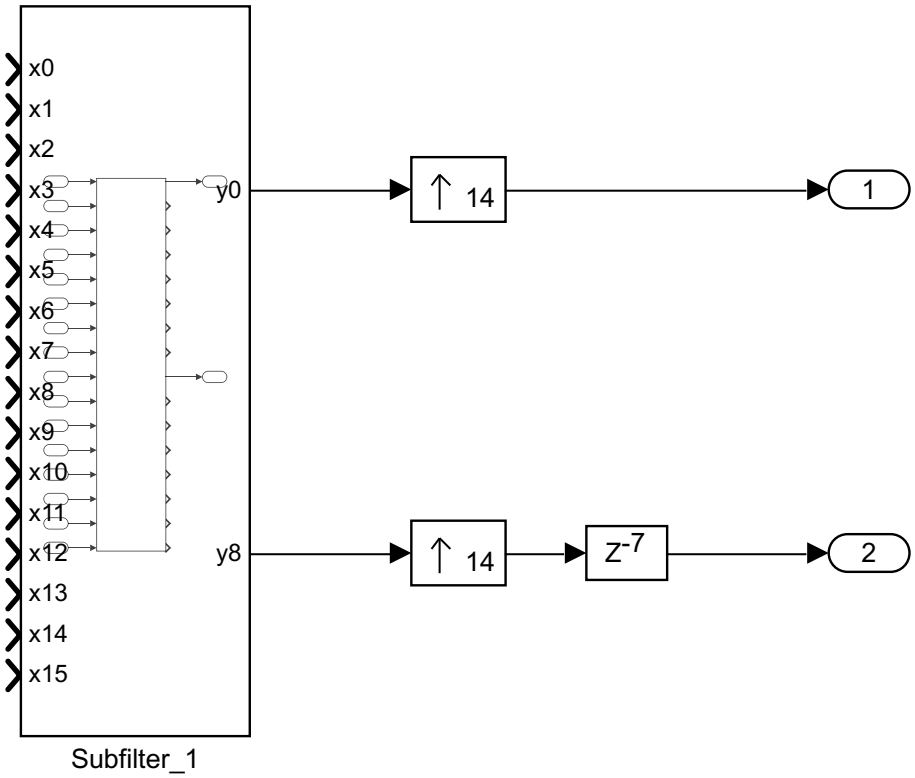

Figura 36: Interpolación y retardo a la salida del filtro

Para comprobar que está correctamente implementado, comparamos con el modelo no paralelizado que ya teníamos hecho (Figura 37).

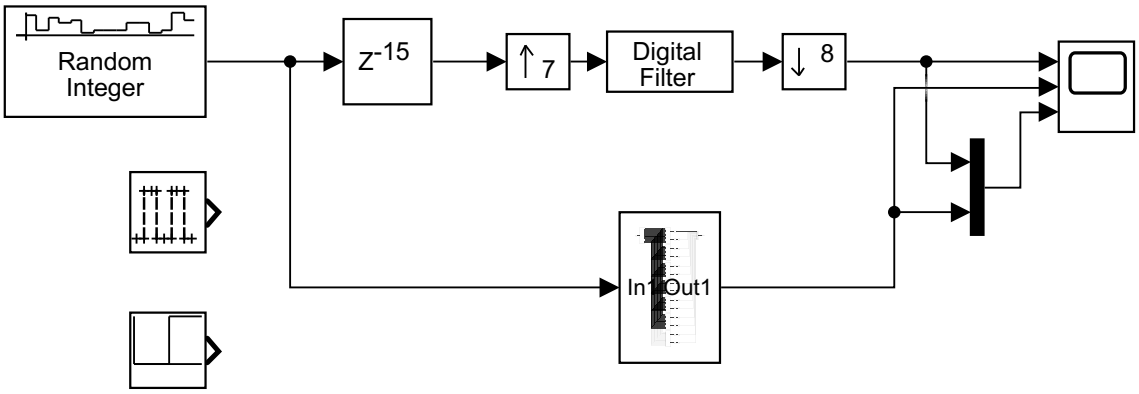

Figura 37: Comparación entre el filtro de cambio de tasa paralelo y no paralelo

En la Figura 38 se puede observar la comparación entre las salidas. La primera se corresponde con la arquitectura no paralela, la segunda con la arquitectura polifásica y la tercera las dos a la vez, ambas coinciden perfectamente.

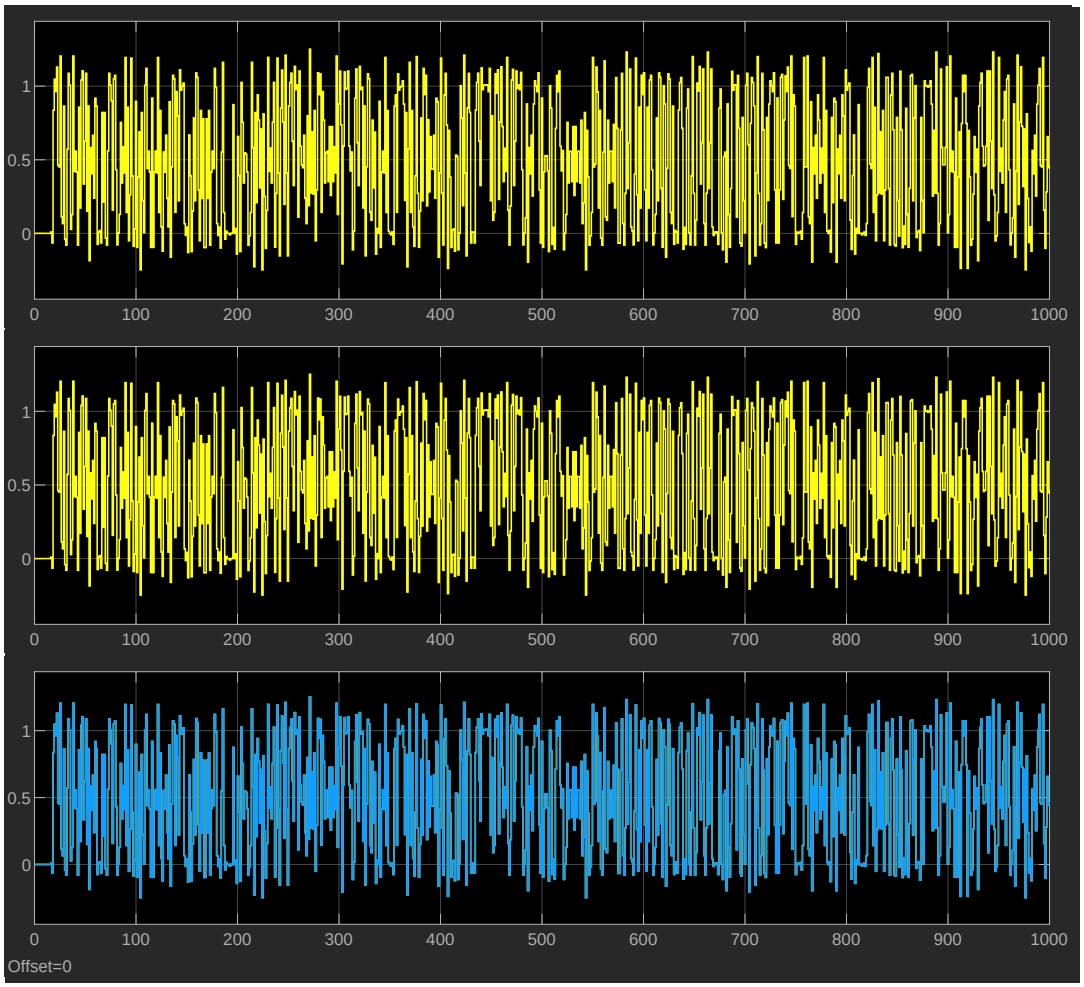

Figura 38: Gráfica comparativa entre la arquitectura no paralela y la paralela

3.1.3. Mezclador simplificado por $f_s/4$

En nuestro sistema, el mezclador se encargará de paralelizar la entrada y de distribuir las muestras de acuerdo a lo visto en la Tabla 1.

La implementación no paralela es bastante sencilla y, tanto para el canal I, como para el canal Q, se multiplica la entrada por una señal sinusoidal desfasada 90 grados entre sí (Figura 39).

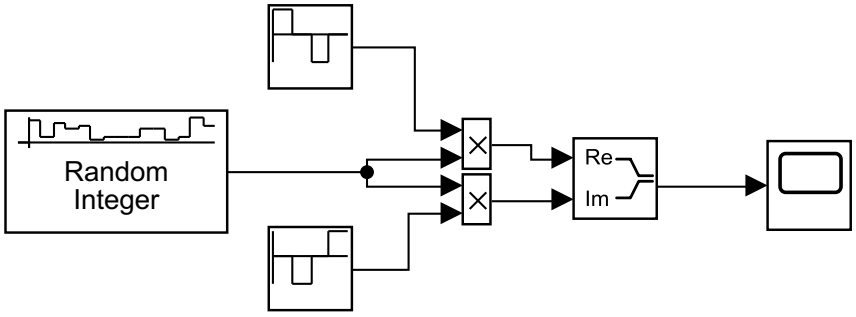

Figura 39: Mezclador no paralelo

Para la implementación paralela, deberemos implementar ambos canales (Figura 40).

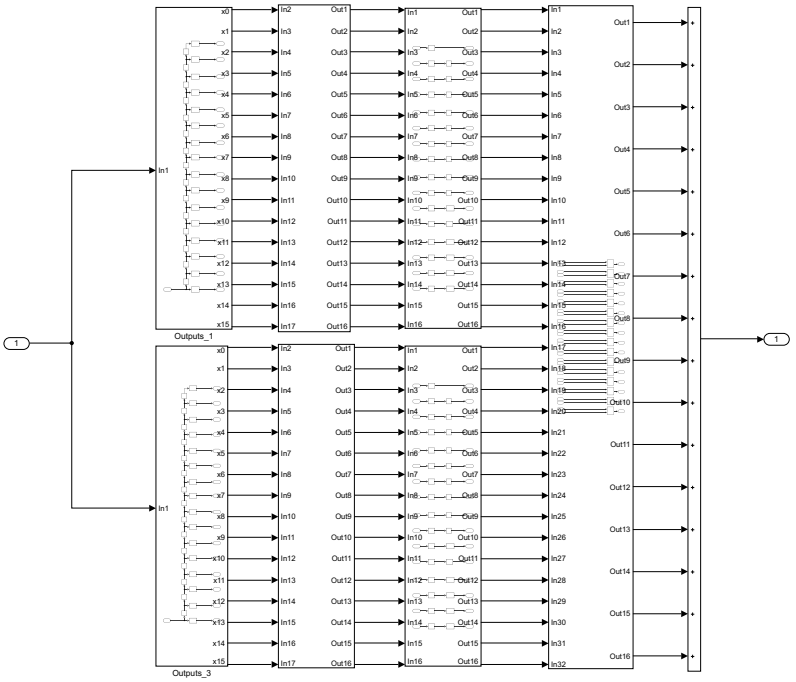

Figura 40: Esquema del mezclador paralelizado

Para la distribución de muestras simplemente deberemos multiplicar la muestra por 1, por -1 o en caso de ser 0, no se cablea a ningún sitio (Figura 41).

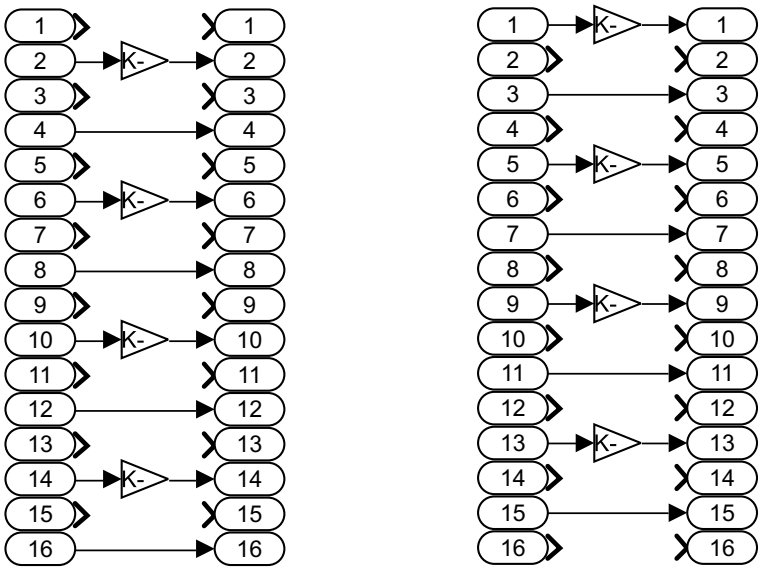

Figura 41: Distribución de muestras para el canal I (izquierda) y para el canal Q (derecha)

Para comprobar que todo funciona correctamente, se compara con el modelo no paralelizado (Figura 42).

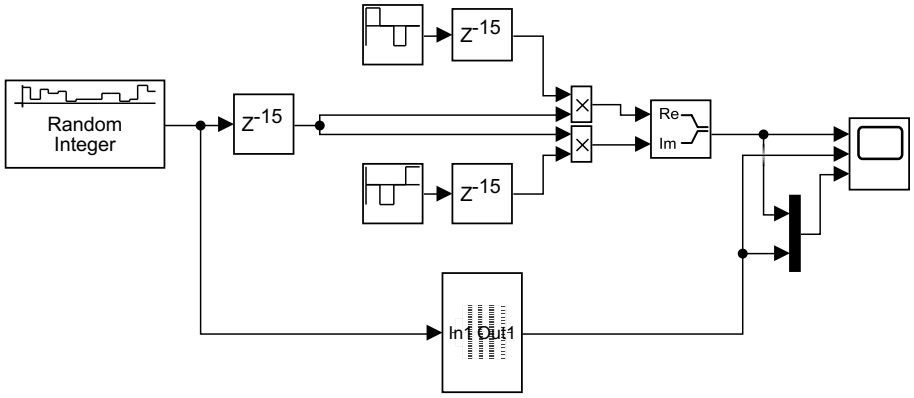

Figura 42: Comparación del mezclador no paralelizado y paralelizado

En la Figura 43 se puede observar la comparación entre las salidas. La primera se corresponde con la arquitectura no paralela, la segunda con la arquitectura paralela y la tercera las dos a la vez.

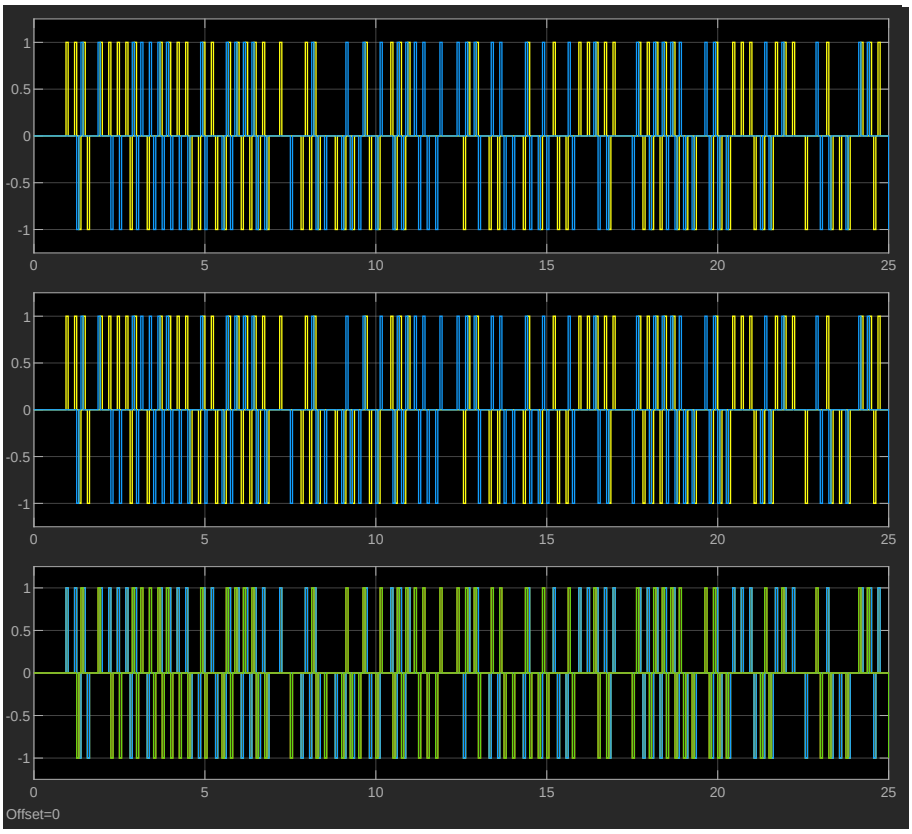

Figura 43: Comparación entre el modelo no paralelo y el paralelo

3.1.4. Receptor completo

Una vez modelados los elementos por separado, deberemos unirlos para formar el receptor (Figura 44).

En primer lugar, obtenemos el receptor no paralelizado, al cual se le ha añadido varias salidas para poder comparar con los otros modelos (Figura 44).

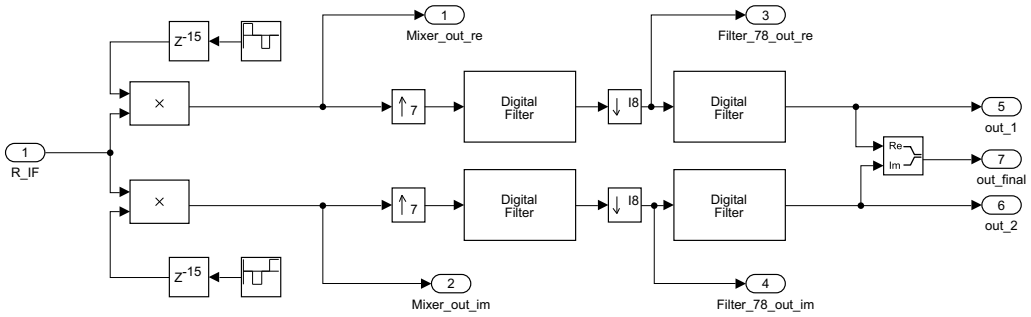

Figura 44: Receptor completo con arquitectura no paralela

Para implementar el receptor completo con arquitectura polifásica, deberemos unir todos los elementos anteriormente vistos, por tanto, a partir de una entrada secuencial, a una frecuencia de fclk,

paralelizamos la entrada en el mezclador para cada canal (Figura 41) y posteriormente aplicamos lo visto en la Figura 45.

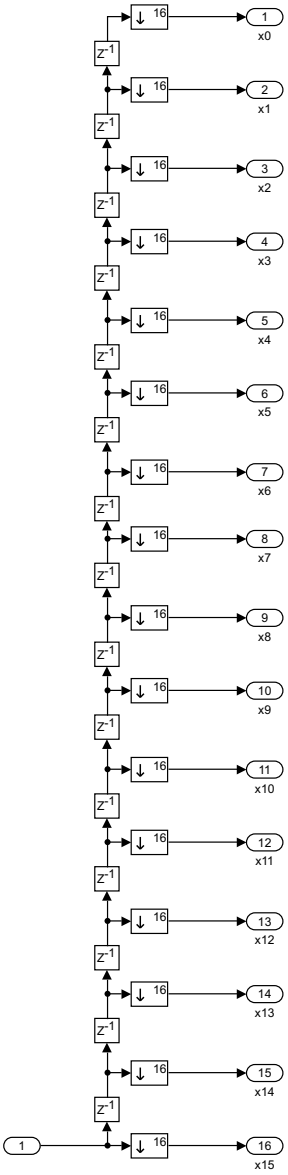

Figura 45: Entrada paralelizada

El resto de elementos se unen entre sí, tal y como se muestra en la Figura 46.

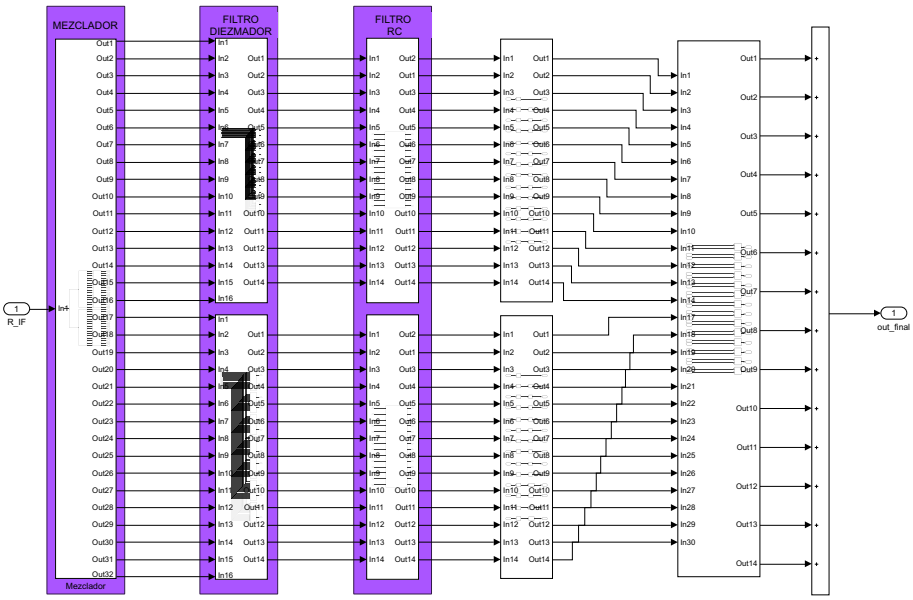

Figura 46: Receptor completo con arquitectura paralela

Sobre el modelo de la figura anterior y al igual que en la Figura 44, sacamos las salidas posteriores a cada elemento implementado, para poder comparar ambos (Figura 47).

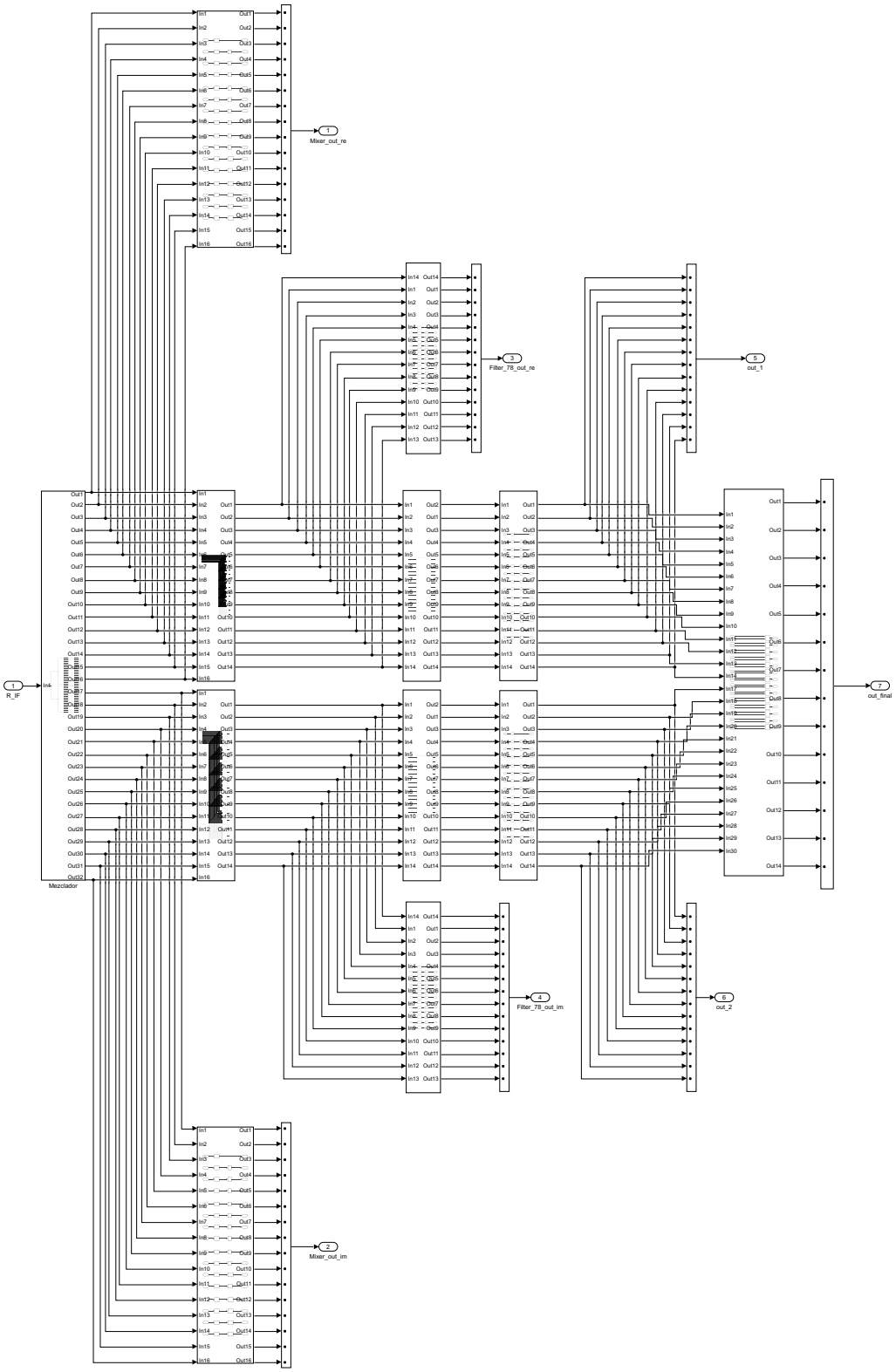

Figura 47: Receptor completo con arquitectura paralela con salidas de depuración

Tal y como se ha realizado en los apartados anteriores, para poder visualizar la salida debemos

volver a unir las muestras interpolando y sumando, es decir revertir lo hecho en la Figura 45.

Una vez hecho esto, comparamos ambos modelos y comprobamos que son equivalentes (Figura 48).

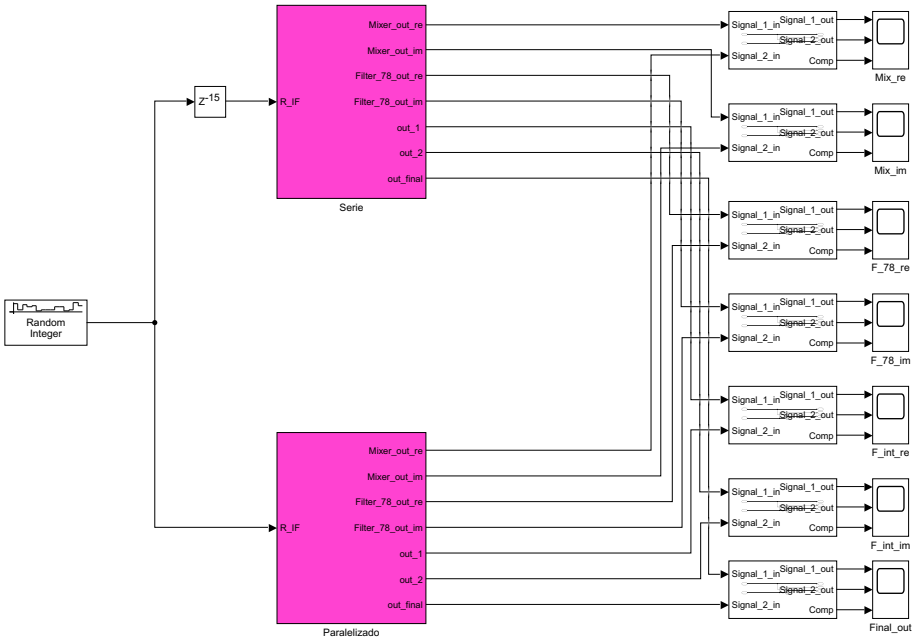

Figura 48: Comparación entre el modelo no paralelo y el modelo paralelo

Con el fin de simplificar la memoria, los resultados finales, fruto de la comparación entre ambos modelos, se realizará en el apartado 3.1.4.4 puesto que se pretende mejorar el modelo en apartados sucesivos.

3.1.4.1. Simplificación

Dada la arquitectura misma del receptor, observando la Figura 36 y la Figura 41, nos damos cuenta que hay ciertas líneas, que no aportan nada al resultado final, puesto que no están cableadas. Esto implica que todo lo relativo a estos canales puede ser eliminado. De esta manera es posible simplificar el mezclador y el filtro diezmador.

Por tanto, de la Figura 41 pasamos a la Figura 49.

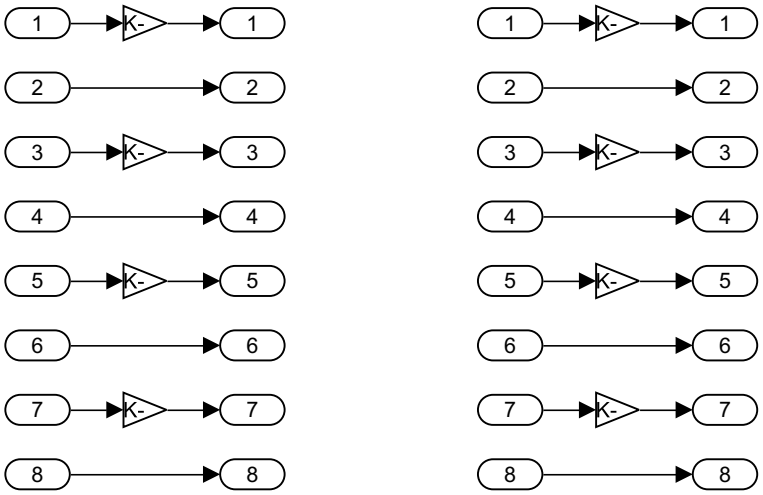

Figura 49: Distribución de muestras para el canal I simplificado (izquierda) y para el canal Q simplificado (derecha)

Esto repercute directamente sobre el filtro diezmador, por tanto, habrá que eliminar los cables que no están conectados a nada. Además, de cada subfiltro solo utilizamos dos salidas, por tanto todo lo que no dependa de esas salidas puede ser eliminado también (Figura 50).

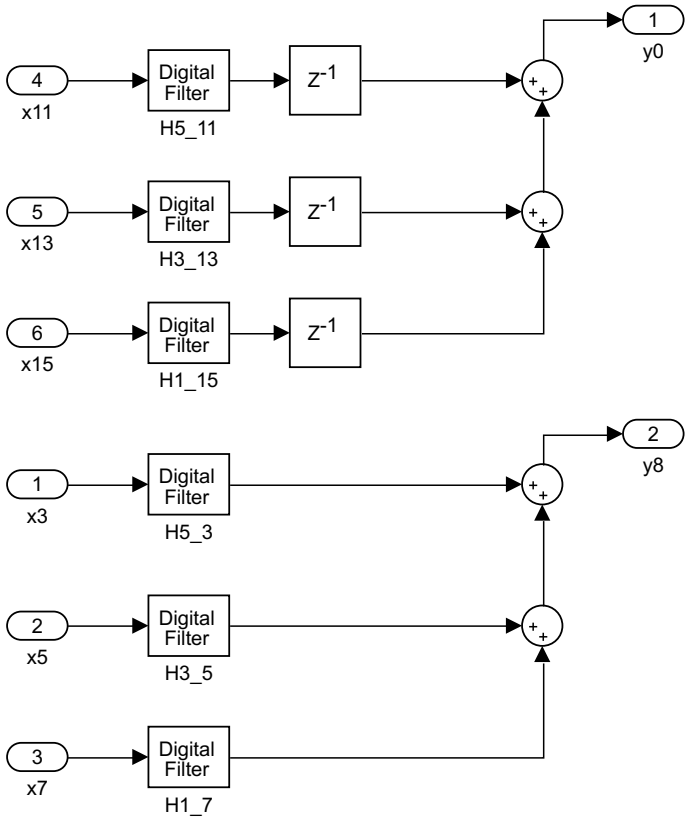

Figura 50: Primer subfiltro del filtro diezmador simplificado

Si lo comparamos con la Figura 35, podemos apreciar una gran diferencia en cuanto a componentes utilizados.

Como nota adicional, es posible simplificar aún más la Figura 49, hasta el punto de incluso eliminar ese bloque completamente. Esto se puede lograr actuando sobre los coeficientes del filtro diezmador. En vez de multiplicar por -1 la entrada, multiplicamos por -1 los coeficientes, esto lo hacemos desde Matlab (Figura 51).

```
%% Filtro 7-8

%Subfiltro 0
h_01 = -h78(1);
h_02 = h78(8);
h_03 = h78(15);
h_04 = -h78(22);
h_05 = -h78(29);
h_06 = h78(36);
h_07 = h78(43);

%Subfiltro 1
h_11 = -h78(2);
h_12 = -h78(9);
h_13 = h78(16);
h_14 = h78(23);
h_15 = -h78(30);
h_16 = -h78(37);
h_17 = h78(44);

%Subfiltro 2
h_21 = h78(3);
h_22 = -h78(10);
h_23 = -h78(17);
h_24 = h78(24);
h_25 = h78(31);
h_26 = -h78(38);
h_27 = -h78(45);

%Subfiltro 3
h_31 = h78(4);
h_32 = h78(11);
h_33 = -h78(18);
h_34 = -h78(25);
h_35 = h78(32);
h_36 = h78(39);
h_37 = -h78(46);

%Subfiltro 4
h_41 = -h78(5);
h_42 = h78(12);
h_43 = h78(19);
h_44 = -h78(26);
h_45 = -h78(33);
h_46 = h78(40);
h_47 = h78(47);

%Subfiltro 5
h_51 = -h78(6);
h_52 = -h78(13);
h_53 = h78(20);
h_54 = h78(27);
h_55 = -h78(34);
h_56 = -h78(41);
h_57 = h78(48);

%Subfiltro 6
h_61 = h78(7);
h_62 = -h78(14);
h_63 = -h78(21);
h_64 = h78(28);
h_65 = h78(35);
h_66 = -h78(42);
h_67 = -h78(49);
```

Figura 51: Coeficientes del filtro diezmador

Por tanto el mezclador quedará tal y como se muestra en la Figura 52.

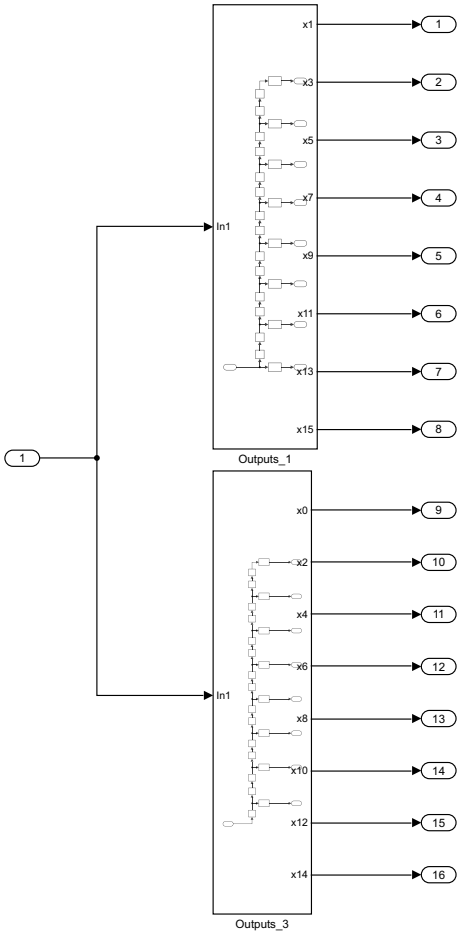

Figura 52: Mezclador simplificado

3.1.4.2. Parametrización

Una mejora propuesta es la de sustituir los subfiltros por un único subfiltro genérico que dependa de unos parámetros de entrada.

A continuación, se va a dar un breve resumen del procedimiento seguido para la parametrización de ambos filtros.

Para el filtro diezmador, se ha observado el número máximo de entradas, el número máximo de filtros FIR empleados y el número máximo de retardos en cada subfiltro. Una vez hecho esto, se ha creado el modelo de la Figura 53.

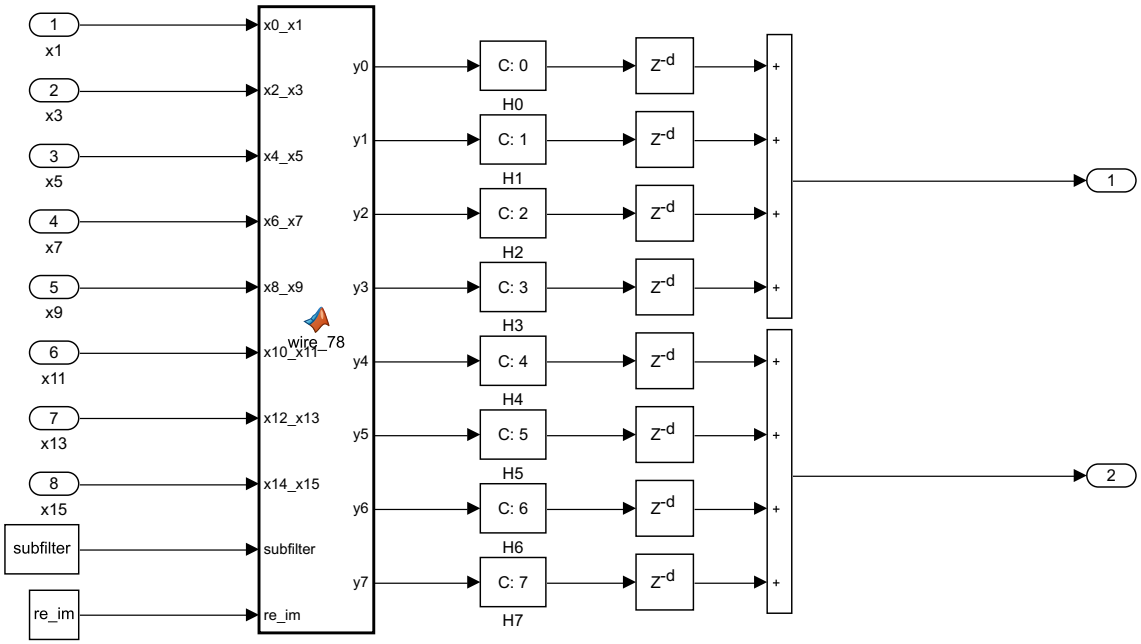

Figura 53: Subfiltro genérico parametrizado del filtro diezmador

Como se puede observar, hemos introducido un bloque de Matlab que nos cablea el subfiltro a partir del número de entradas. El cableado depende directamente del número de subfiltro y del canal que estemos (I o Q). Podemos ver el código implementado en la Figura 54.

```
function [y0,y1,y2,y3,y4,y5,y6,y7] = wire_78(x0_x1,x2_x3,x4_x5,x6_x7,x8_x9,x10_x11,x12_x13,x14_x15,
    subfilter,re_im)
%re_im -> re_im = 1 (re); re_im = 0 (im)

input_re = [x10_x11 x12_x13 x14_x15 0 x2_x3 x4_x5 x6_x7 0 x0_x1 x10_x11 x12_x13 x14_x15 x2_x3 x4_x5
    x6_x7 x8_x9 x0_x1 x12_x13 x14_x15 0 x4_x5 x6_x7 x8_x9 0 x0_x1 x2_x3 x12_x13 x14_x15 x4_x5 x6_x7
     x8_x9 x10_x11 x0_x1 x2_x3 x14_x15 0 x6_x7 x8_x9 x10_x11 0 x0_x1 x2_x3 x4_x5 x14_x15 x6_x7
    x8_x9 x10_x11 x12_x13 x0_x1 x2_x3 x4_x5 0 x8_x9 x10_x11 x12_x13 0];

input_im = [x0_x1 x10_x11 x12_x13 x14_x15 x2_x3 x4_x5 x6_x7 x8_x9 x0_x1 x12_x13 x14_x15 0 x4_x5
    x6_x7 x8_x9 0 x0_x1 x2_x3 x12_x13 x14_x15 x4_x5 x6_x7 x8_x9 x10_x11 x0_x1 x2_x3 x14_x15 0 x6_x7
     x8_x9 x10_x11 0 x0_x1 x2_x3 x4_x5 x14_x15 x6_x7 x8_x9 x10_x11 x12_x13 x0_x1 x2_x3 x4_x5 0
    x8_x9 x10_x11 x12_x13 0 x0_x1 x2_x3 x4_x5 x6_x7 x8_x9 x10_x11 x12_x13 x14_x15];

if re_im == 1
   y0 = input_re(1+8*(subfilter));
   y1 = input_re(2+8*(subfilter));
   y2 = input_re(3+8*(subfilter));
   y3 = input_re(4+8*(subfilter));
   y4 = input_re(5+8*(subfilter));
   y5 = input_re(6+8*(subfilter));
   y6 = input_re(7+8*(subfilter));
   y7 = input_re(8+8*(subfilter));
elseif re_im == 0
   y0 = input_im(1+8*(subfilter));
   y1 = input_im(2+8*(subfilter));
   y2 = input_im(3+8*(subfilter));
   y3 = input_im(4+8*(subfilter));
   y4 = input_im(5+8*(subfilter));
   y5 = input_im(6+8*(subfilter));
   y6 = input_im(7+8*(subfilter));
   y7 = input_im(8+8*(subfilter));
else
   y0 = 0;
   y1 = 0;
   y2 = 0;
   y3 = 0;
   y4 = 0;
   y5 = 0;
   y6 = 0;
   y7 = 0;
end
```

Figura 54: Código que rige el cableado del subfiltro del filtro diezmador

Después de cablear, se ha configurado los coeficientes de los filtros mediante una función (Figura 55) que, al igual que el cableado, depende directamente de si está en el canal I o en el canal Q, del número de subfiltro y en este caso también del canal del mismo.

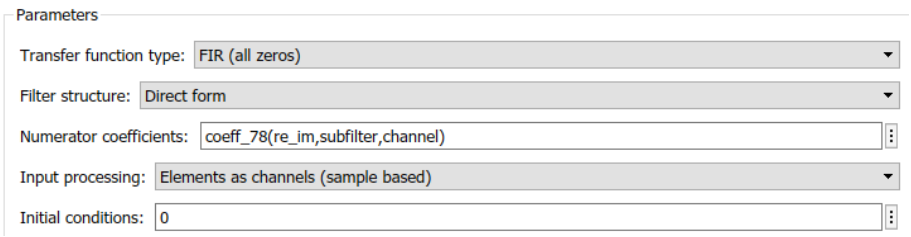

Figura 55: Parámetros de los filtros digitales

El código que implementa la función coeff_78 se muestra a continuación (Figura 56). Nota: Para simplificar la imagen se ha obviado la declaración de cada coeficiente, esto se puede observar en imágenes anteriores (Figura 31 y Figura 34).

```
function h = coeff_78(re_im,subfilter,channel) %re_im -> re_im = 1 (re); re_im = 0 (im)
%% Coeficientes filtro de retardo fraccionario

coeff_re = [h_06 h_04 h_02 0 h_06 h_04 h_02 0 h_11 h_17 h_15 h_13 h_17 h_15 h_13 h_11 h_22 h_26 h_24
      0 h_26 h_24 h_22 0 h_33 h_31 h_37 h_35 h_37 h_35 h_33 h_31 h_44 h_42 h_46 0 h_46 h_44 h_42 0
     h_55 h_53 h_51 h_57 h_57 h_55 h_53 h_51 h_66 h_64 h_62 0 h_66 h_64 h_62 0];

coeff_im = [h_01 h_07 h_05 h_03 h_07 h_05 h_03 h_01 h_12 h_16 h_14 0 h_16 h_14 h_12 0 h_23 h_21 h_27
      h_25 h_27 h_25 h_23 h_21 h_34 h_32 h_36 0 h_36 h_34 h_32 0 h_45 h_43 h_41 h_47 h_47 h_45 h_43
     h_41 h_56 h_54 h_52 0 h_56 h_54 h_52 0 h_67 h_65 h_63 h_61 h_67 h_65 h_63 h_61];

if re_im == 1
    h = coeff_re((1 + channel) + 8*subfilter);
elseif re_im == 0
    h = coeff_im((1 + channel) + 8*subfilter);
else
    h = 10;
end
```

Figura 56: Función que rige el coeficiente de cada filtro digital

Por último se ha de configurar el retardo, por ello, al igual que los filtros FIR, se utilizará una función que calcule el valor del retardo (Figura 58), cuyas dependencias son las mismas que la función de los coeficientes.

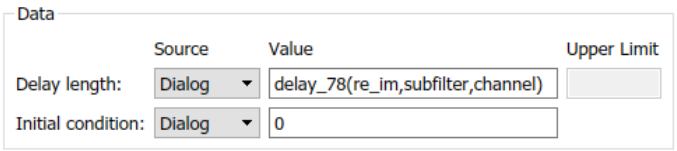

Figura 57: Valor del retardo

El código que implementa esta función se muestra en la Figura 58.

```
function Z = delay_78(re_im,subfilter,channel) %re_im -> re_im = 1 (re); re_im = 0 (im)

delay_re = [1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1
     0 0 0 0 0 0 0 0 0 0 0 0 0];
delay_im = [0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
     0 0 0 0 0 0 0 0 0 0 0 0 0];

if re_im == 1
    Z = delay_re(1 + channel + 8*subfilter);
elseif re_im == 0
    Z = delay_im(1 + channel + 8*subfilter);
else
    Z = 2;
end
```

Figura 58: Función que rige el valor de cada retardo

A partir de la Figura 54, la Figura 56 y la Figura 58 se puede observar que la manera de parametrizar ha sido la de crear un vector con los valores y a partir de los parámetros se calcula hacia donde apunta la salida. Si bien es cierto que este método no es el ideal, ya que esto debería calcularse aplicando un patrón lógico que rige la configuración de los subfiltros, tras la simplificación vista en el apartado 3.1.4.1, es bastante complicado hacerlo de esta manera.

Para el filtro RC, se ha seguido un procedimiento similar al del filtro diezmador (Figura 59).

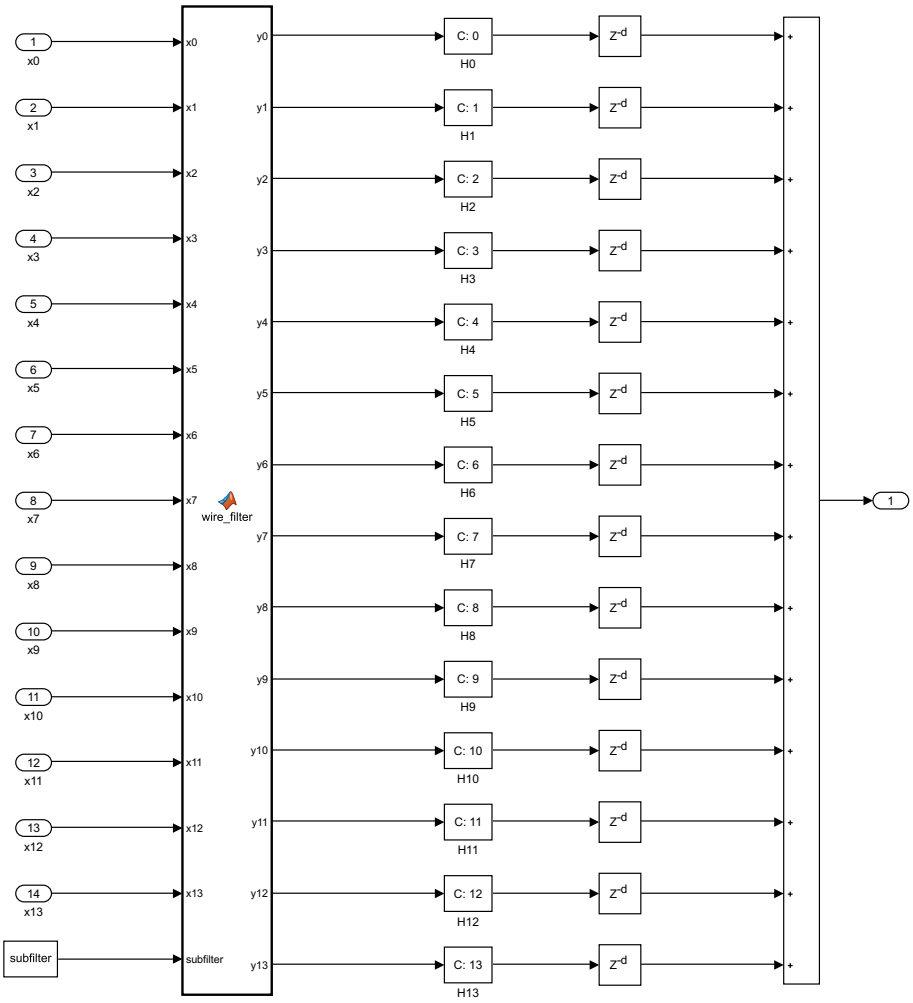

Figura 59: Subfiltro genérico parametrizado del filtro RC

El código que implementa el cableado del filtro RC se puede observar en la Figura 60 y depende únicamente del subfiltro.

```
function [y0,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,y13] =
wire_filter(x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,subfilter)
%re_im -> re_im = 1 (re); re_im = 0 (im)

input = [x0 x13 x12 x11 x10 x9 x8 x7 x6 x5 x4 x3 x2 x1];

input_shift = circshift(input,subfilter);

y0 = input_shift(1);
y1 = input_shift(2);
y2 = input_shift(3);
y3 = input_shift(4);
y4 = input_shift(5);
y5 = input_shift(6);
y6 = input_shift(7);
y7 = input_shift(8);
y8 = input_shift(9);
y9 = input_shift(10);
y10 = input_shift(11);
y11 = input_shift(12);
y12 = input_shift(13);
y13 = input_shift(14);
```

Figura 60: Código que rige el cableado del subfiltro del filtro RC

El código que implementa la función que calcula el coeficiente se muestra en la Figura 61 y depende solamente del canal. Nota: Para simplificar la imagen se ha obviado la declaración de cada coeficiente, esto se puede observar en imágenes anteriores (Figura 22 y Figura 24).

```
function h = coeff_filter(channel) %re_im -> re_im = 1 (re); re_im = 0 (im)

coeff = [H0 H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11 H12 H13];

h = coeff(channel*3 + 1:channel*3 + 3);
```

Figura 61: Función que rige el coeficiente de cada filtro digital

El código que implementa la función que calcula el retardo se muestra en la Figura 62, en este caso depende tanto del subfiltro como del canal.

```
function Z = delay_filter(subfilter, channel)

delay = [0 1 1 1 1 1 1 1 1 1 1 1 1 1];
delay(2:subfilter+1) = 0;
Z = delay(channel+1);
```

Figura 62: Función que rige el valor de cada retardo

Como se puede observar, a diferencia del filtro diezmador, aquí sí que es posible establecer una relación entre los subfiltros y en vez de crear un vector con los valores, se obtiene la salida a partir de una ecuación.

3.1.4.3. Cuantificación

Para la cuantificación de nuestro modelo, deberemos tener en cuenta el modelo de la FPGA donde va a ser implementado ya que cada una tendrá unos DSP slices propios.

En nuestro caso, al tratarse de una FPGA de serie 7, contamos con el slice DSP48E1 (Figura 63).

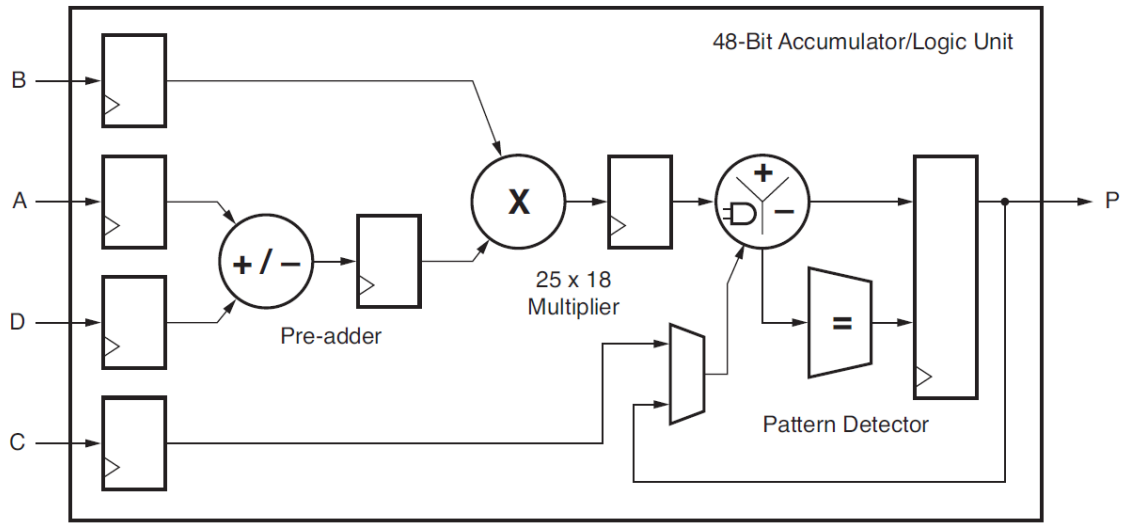

Figura 63: DSP48E1 slice

Tal y como se aprecia en la imagen, contamos con multiplicadores de 25 bits x 18 bits, además de un acumulador de 48 bits. La razón de fijarnos en los DSP es la de optimizar al máximo la implementación de nuestro modelo. No debemos sobrepasar los 25 bits en las multiplicaciones [4].

Por tanto, fijamos la entrada con un formato [10 9], ya que está impuesta por el ADC, y los coeficientes del filtro diezmador con un formato [25 24]. El resultado de la multiplicación entre ambos datos tendría un formato [34 33]. Dado que sobrepasamos lo máximo que permite el slice, truncamos a [25 24].

Ahora, partiendo del filtro RC, los datos de entrada tienen un formato de [25 24], mientras que los coeficientes tienen un formato [18 17]. La multiplicación entre ambos tiene un formato [42 41] y nuevamente truncamos a [25 24].

En la Figura 64 se puede observar el tamaño del flujo de datos.

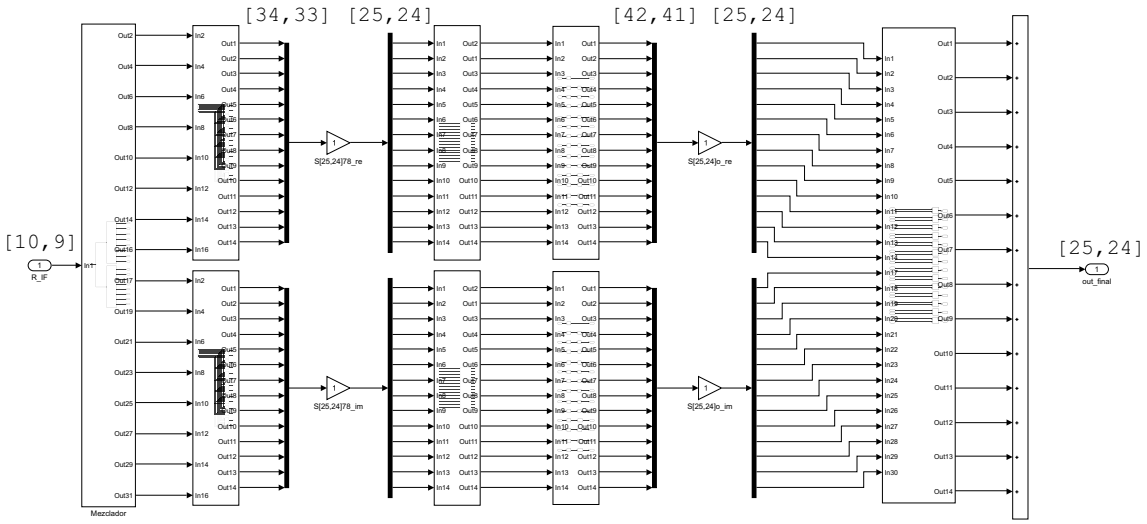

Figura 64: Cuantificación del receptor QAM

Para dejar la salida ajustada en el rango [-1, 1), se ha aumentado en un factor 2 los coeficientes del filtro RC.

3.1.4.4. Resultados

Una vez realizado todos los procesos anteriores, se han comparado los tres modelos principales entre sí, con el fin de ver cuál es el error entre ellos.

Los modelos que se han comparado han sido el modelo no paralelizado ideal (sin cuantificación), el modelo con arquitectura polifásica ideal (sin cuantificación) y el modelo con arquitectura polifásica cuantificado (Figura 65).

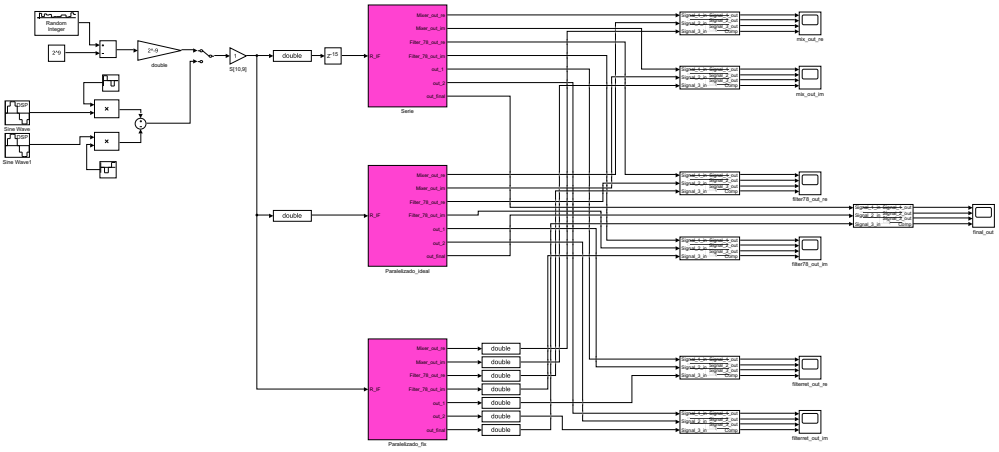

Figura 65: Esquema principal de los tres modelos

En la figura anterior se puede observar que comparamos en diferentes puntos, en concreto, esto se hace en cada canal después del mezclador, del filtro diezmador, del filtro RC y en la salida final del sistema.

Esta comparación no solo se hace de forma gráfica, sino también de forma numérica, enviando los datos de salida al workspace de Matlab (Figura 66).

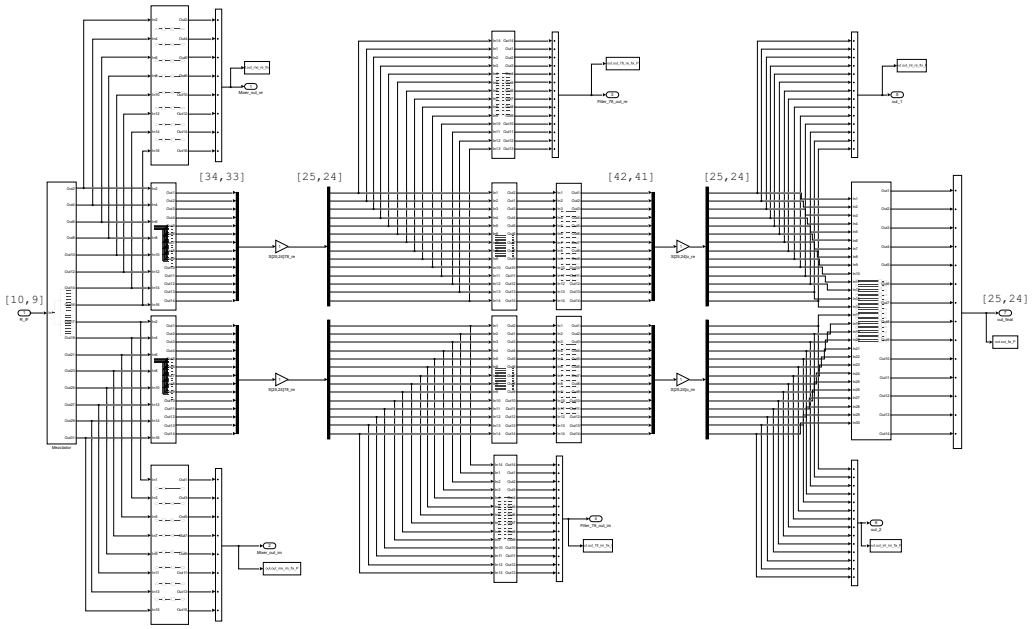

Figura 66: Esquema principal del modulador paralelizado

En la memoria solo se va a enseñar la comparación, tanto gráfica como numérica, de la salida final del sistema, el resto de señales se han utilizado únicamente para depuración.

En la Figura 67 se puede ver dicha comparación, aunque en este caso, es imposible observar a simple vista las diferencias entre los tres modelos.

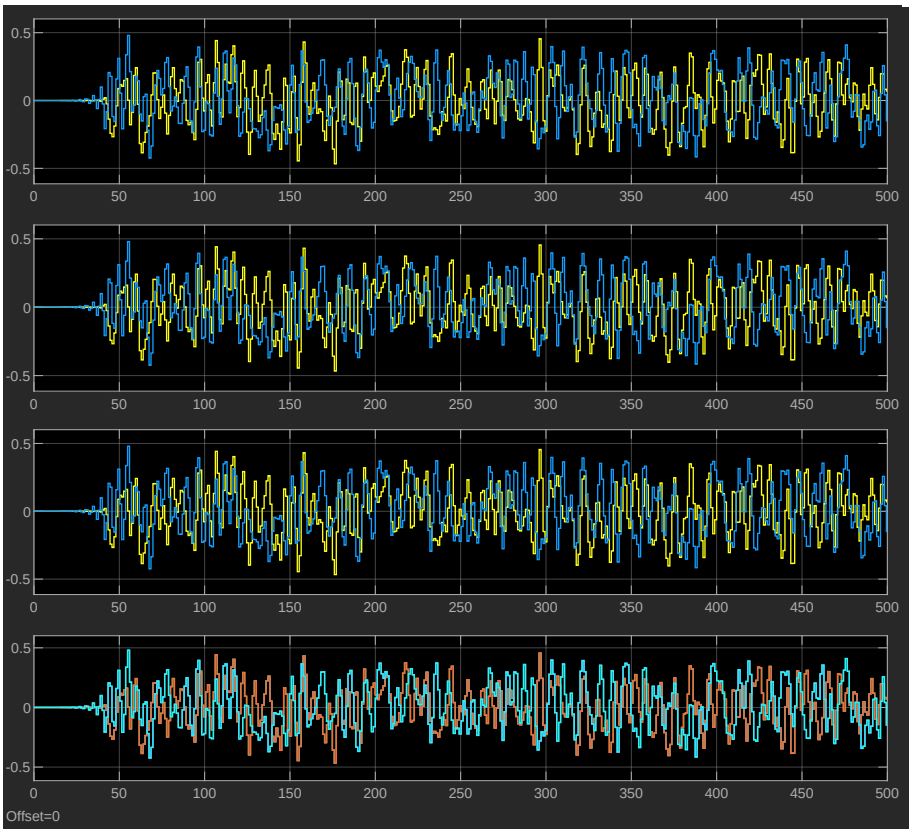

Figura 67: Comparación gráfica entre los tres modelos

Sin embargo, si los analizamos numéricamente, sí que se observan las diferencias, esto se puede observar en la Figura 68, Figura 69 y Figura 70.

```
****************************************
Non-parallelized Ideal vs Parallellized Ideal
****************************************
Error in filter 78
   re = 1.1409e-07
   im = 1.1094e-07
Error in filter RC
   re = 8.5296e-08
   im = 8.6128e-08
Error in final out = 9.8859e-08
****************************************
```

Figura 68: Comparación numérica entre el modelo no paralelizado ideal y el paralelizado ideal

```
****************************************
Parallellized Ideal vs Parallellized Fixed Point
****************************************
Error in mixer
   re = 0
   im = 0
Error in filter 78
   re = 2.9686e-08
   im = 2.9802e-08
Error in filter RC
   re = 5.1206e-08
   im = 4.6509e-08
Error in final out = 6.0843e-08
****************************************
```

Figura 69: Comparación numérica entre el modelo paralelizado ideal y el paralelizado cuantificado

```
****************************************

Non-parallelized Ideal vs Parallellized Fixed Point

****************************************

Error in filter 78

   re = 1.4086e-07

   im = 1.3341e-07

Error in filter RC

   re = 1.0718e-07

   im = 1.0628e-07

Error in final out = 1.231e-07

****************************************
```

Figura 70: Comparación numérica entre el modelo no paralelizado ideal y el paralelizado cuantificado

3.2. Implementación en HDL

Una vez se ha obtenido el modelo de Matlab, se realizará su implementación en lenguaje HDL, en este caso se ha elegido Verilog por tratarse del lenguaje utilizado a lo largo del master.

Tal y como se ha realizado en Matlab, en Verilog trataremos de dividir la implementación en pequeños bloques que después iremos instanciando en módulos superiores, de esta forma facilitaremos la implementación y la posterior verificación.

En la Figura 71 podemos ver la arquitectura utilizada en el proyecto.

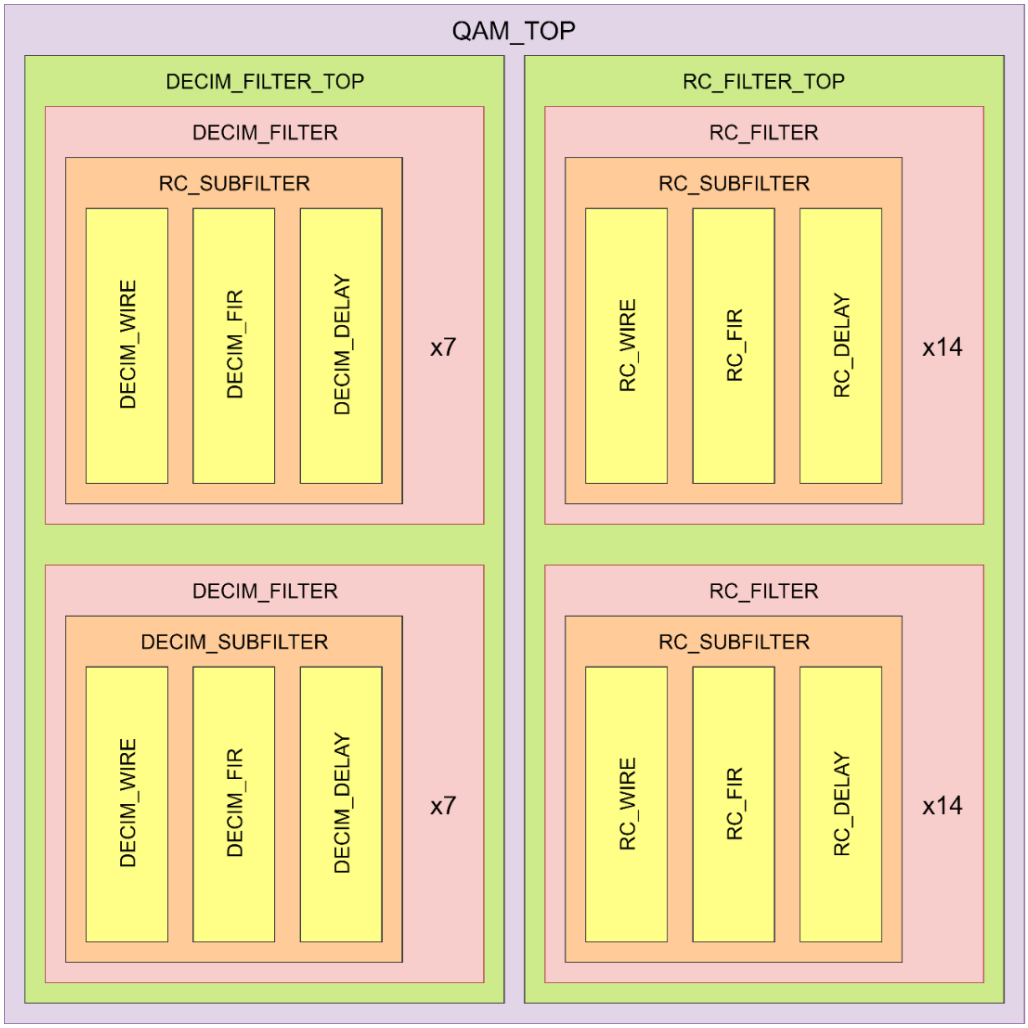

Figura 71: Arquitectura del modelo en HDL

A pesar de que la verificación se ha ido haciendo a medida que se han ido implementando los módulos, no entraremos en más detalles hasta el apartado 3.3, puesto que la metodología utilizada es algo compleja y requiere de un apartado propio.

La tarjeta para la que se ha implementado el modelo es la Virtex 7 VC707. En la Figura 72 se pueden ver las características básicas de este modelo de FPGA [5].

Logic Cells	485,760
DSP Slices	2,800
Memory (Kb)	37,080
GTX 12.5 GB/s Transceivers	56
I/O Pins	700

Figura 72: Características del modelo VC707

Una de las ventajas de este modelo es que dispone de un total de 2800 DSP slices y, dada las características de nuestro sistema, necesitaremos un gran número de ellos.

Al igual que en el apartado 3.1 se ira explicando uno a uno los elementos modelizados y los módulos relativos a ellos.

3.2.1. Filtro FIR

Antes de comenzar con el filtro RC y el filtro diezmador, deberemos tener en cuenta que a pesar de que el filtrado final en Simulink se hace mediante los bloques de "Filtro Digital" (Figura 73), nosotros deberemos implementar dichos bloques en Verilog.

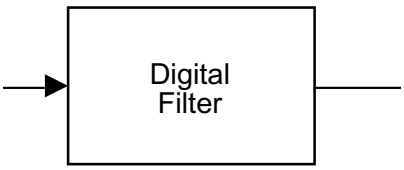

Figura 73: Filtro Digital de Simulink

El filtro FIR se utiliza tanto para el filtro RC como para el filtro diezmador. En el caso del filtro RC son, en la mayoría de los casos, de 3 coeficientes, mientras que en el filtro diezmador son de 1 único coeficiente. A continuación, se puede observar el esquema de un filtro FIR de tres parámetros en modo directo (Figura 74).

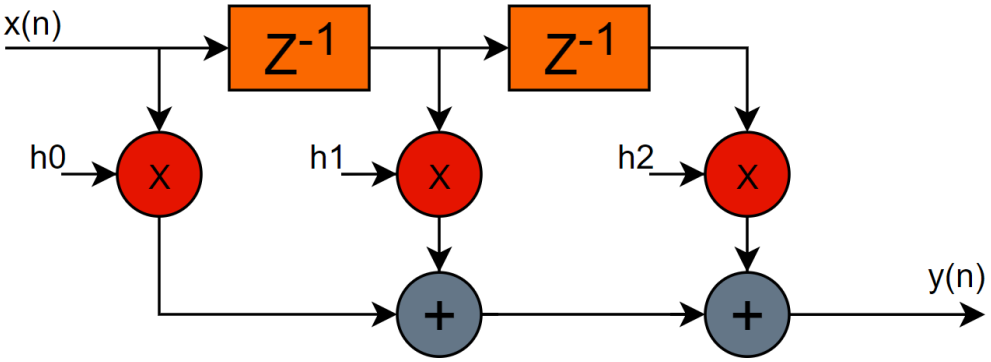

Figura 74: Esquema de un filtro FIR

Dado que necesitamos una frecuencia relativamente alta, deberemos segmentar este filtro para evitar problemas de tiempos (Figura 75).

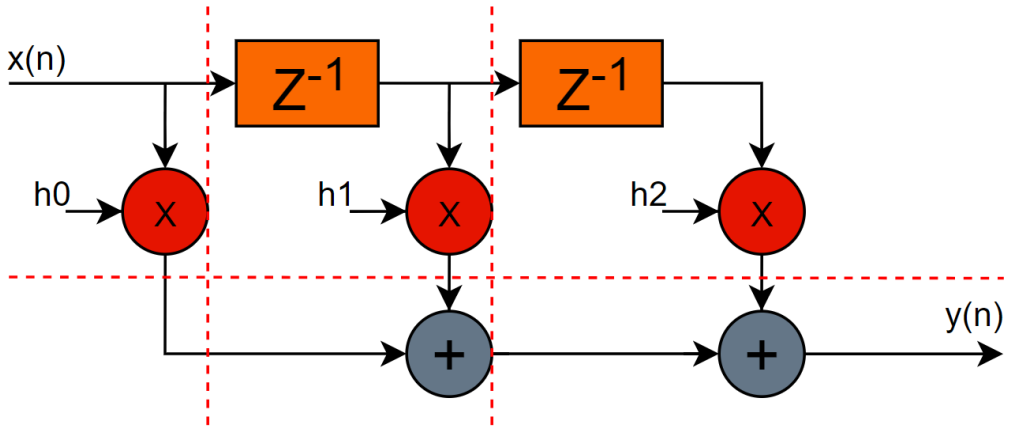

Figura 75: Filtro FIR con segmentación vertical y horizontal

Finalmente nos queda el filtro de la siguiente forma:

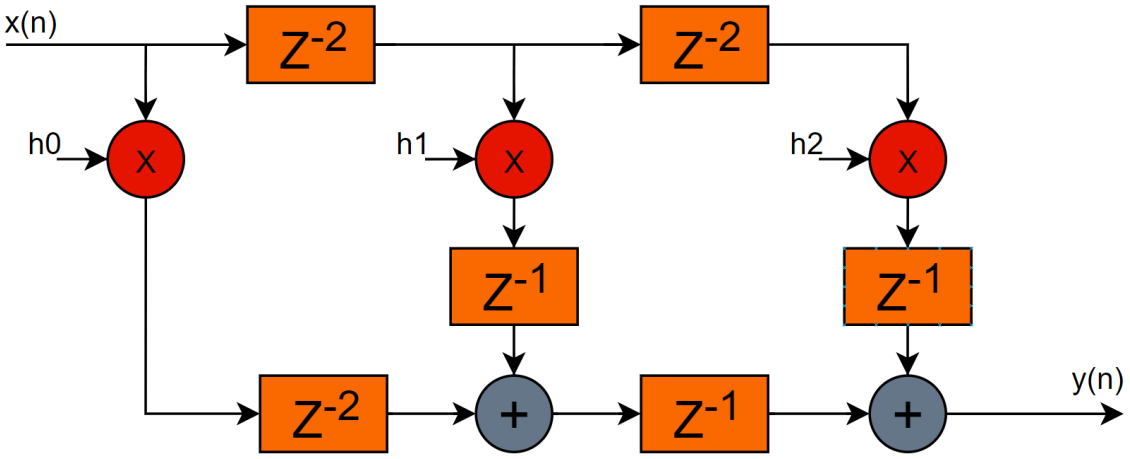

Figura 76: Filtro FIR segmentado

La implementación de este módulo se ha realizado de forma que disponga de una entrada de tamaño Win, una señal de reset, una señal de reloj, una señal de sincronización de entrada, una salida cuyo tamaño depende directamente del crecimiento de los datos y una señal de sincronización de salida. Los coeficientes se pasan mediante parámetros al módulo y su tamaño viene definido por Wcoeff.

Una vez instanciado el módulo deberemos pasarle las entradas y salidas, así como el parámetro Win, Wcoeff y los coeficientes H0, H1 y H2 (Figura 77).

```
FIR #( .Win(Win), .Wcoeff(Wcoeff), .H0(H0), .H1(H1), .H2(H2)) FIR_H0 (
  .iCLK(iCLK),
  .iRESET(iRESET),
  .iDATA(iDATA_0),
  .iVAL(iVAL),
  .oDATA(w_oDATA_0),
  .oVAL(oVAL)
  );
```

Figura 77: Instanciación del filtro FIR

La codificación en Verilog se ha realizado siguiendo el esquema de la Figura 76, teniendo en cuenta el crecimiento de los datos. Dado que nuestra entrada es de Win bits y los coeficientes de Wcoeff bits, el resultado, fruto de la multiplicación, es de Win+Wcoeff bits. El crecimiento en la primera suma es de 1 bit y en la segunda suma es de otro bit, por tanto, el tamaño de la salida será Win+Wcoeff+2 bits.

Cabe destacar que, si bien este filtro está diseñado para tres coeficientes, sirve para filtros con tres o menos coeficientes. Si le pasamos como parámetro un coeficiente de valor 0, el compilador de Vivado eliminará las ramas correspondientes.

En la Figura 78 podemos ver el esquema RTL del filtro FIR de tres coeficientes.

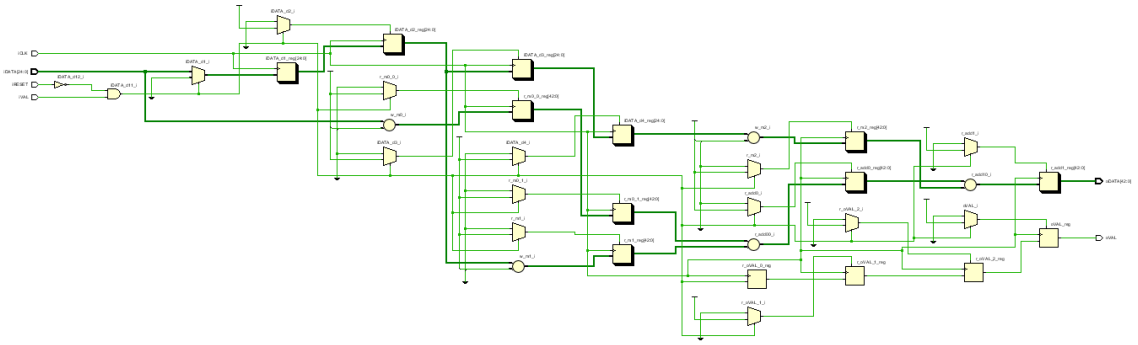

Figura 78: Esquema RTL del filtro FIR

3.2.2. Filtro RC

El filtro RC está compuesto por una serie de módulos que se implementan de forma muy parecida al modelo de Simulink. Está formado por los módulos RC_WIRE, RC_FIR, RC_DELAY, RC_SUBILTER, RC_FILTER y RC_TOP.

3.2.2.1. RC_WIRE

Basándonos en la Figura 59, la parte que implementa el módulo RC_WIRE es la del cableado de cada subfiltro (Figura 79).

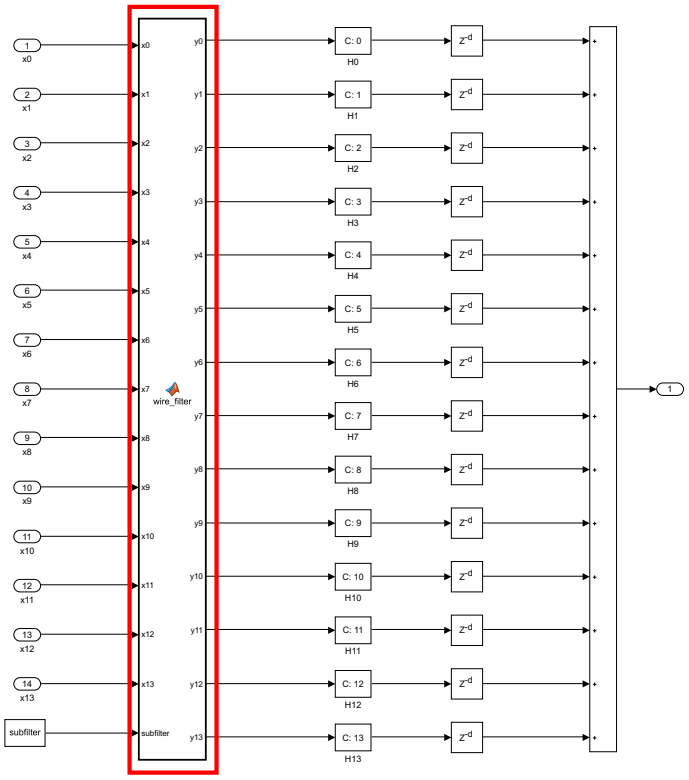

Figura 79: Correspondencia entre Simulink y Verilog del módulo RC_WIRE

Este módulo se encarga de asignar la entrada de los filtros FIR, la cual depende directamente del número de subfiltro.

La implementación en Verilog se ha procurado hacer parametrizable, de manera que según un parámetro llamado subfilter se generará un código u otro. Para ello se ha hecho uso de la instrucción generate (Figura 80).

```
generate
    if (subfilter == 0)
      always @(posedge iCLK)
        ...
    else if (subfilter == 1)
      always @(posedge iCLK)
        ...
    else if (subfilter == 2)
      always @(posedge iCLK)
        ...
Etc.
```

Figura 80: Ejemplo de uso de generate en el módulo RC_WIRE

El módulo consta de 14 entradas de tamaño Win, una señal de reset, una señal de reloj, una señal de sincronización de entrada, 14 salidas cuyo tamaño son exactamente iguales a las entradas (Win) y una señal de sincronización de salida.

En la Figura 81 podemos ver el esquema RTL (subfilter = 0).

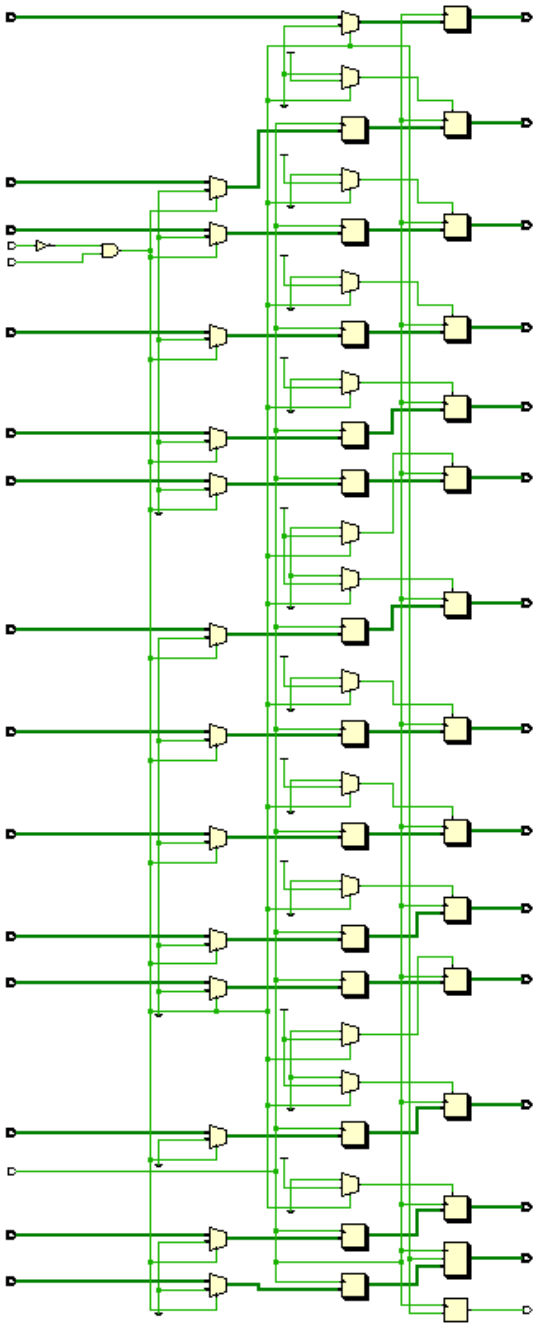

Figura 81: Esquema RTL del módulo RC_WIRE

3.2.2.2. RC_FIR

El módulo RC_FIR se corresponde con los filtros FIR (Figura 82)

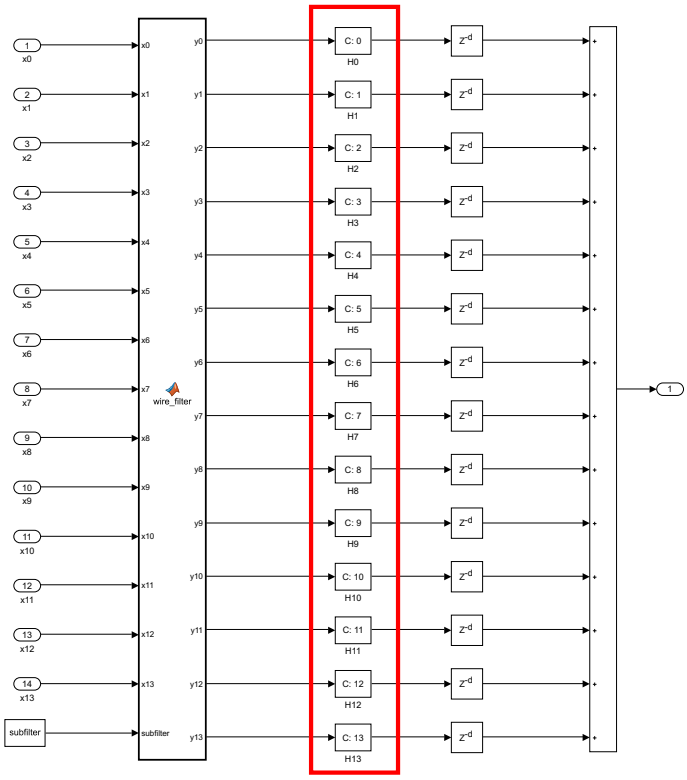

Figura 82: Correspondencia entre Simulink y Verilog del módulo RC_FIR

En este módulo se instancian los 14 filtros FIR que componen cada subfiltro, aunque, en este caso, este módulo es exactamente igual en cada uno de ellos.

Deberemos tener en cuenta el tamaño de entrada, el tamaño de salida y los 41 coeficientes. En cuanto a las entradas y salidas, este módulo consta de 14 entradas de tamaño Win, una señal de reset, una señal de reloj, una señal de sincronización de entrada, 14 salidas de tamaño Wo y una señal de sincronización de salida.

En la Figura 83 podemos ver el esquema RTL (subfilter = 0).

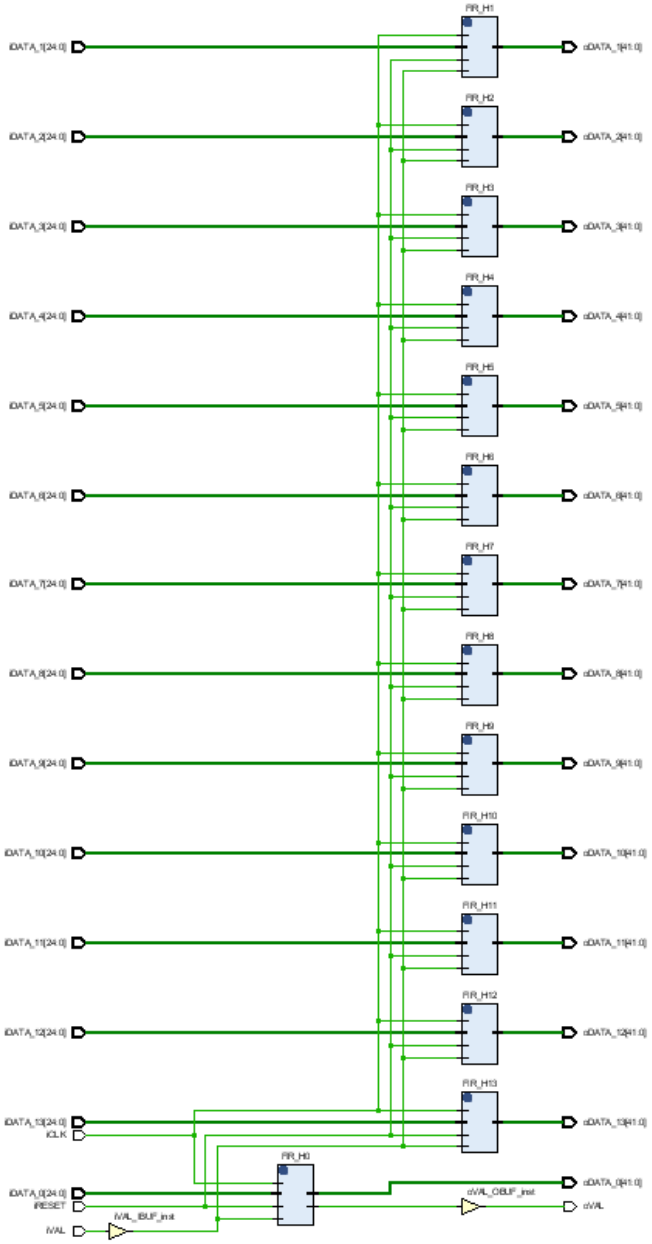

Figura 83: Esquema RTL del módulo RC_FIR

3.2.2.3. RC_DELAY

El módulo RC_DELAY equivale a la parte que aplica los retardos (Figura 84).

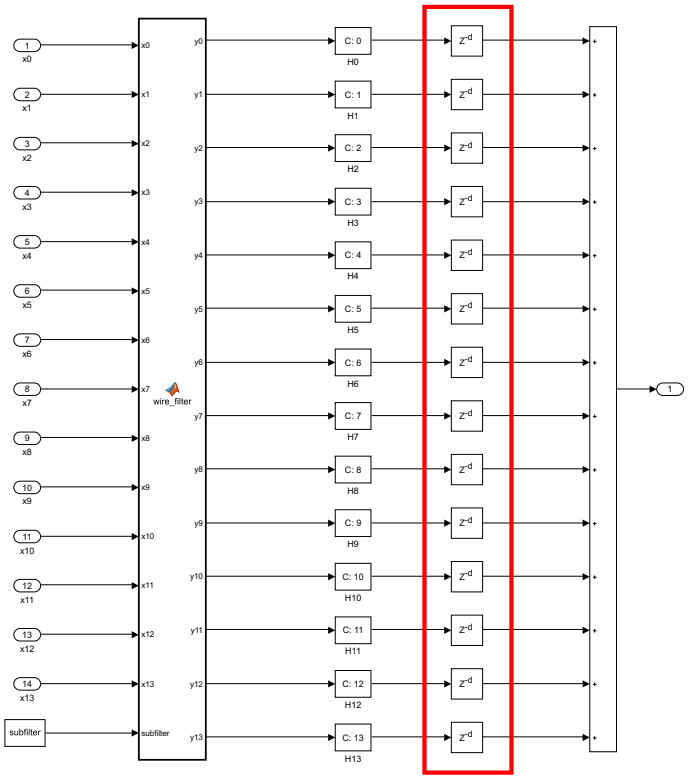

Figura 84: Correspondencia entre Simulink y Verilog del módulo RC_DELAY

La función de este módulo consiste en retrasar la señal, en nuestro caso se ha metido un retardo a todos los canales, y un segundo retardo que afecta solo a aquellos que en el modelo de Matlab/Simulink tienen un retardo.

Las entradas y salidas de este módulo son exactamente iguales a las salidas del módulo RC_FIR, aunque en este caso se espera que tanto entradas como salidas tengan el mismo tamaño (Win).

Al igual que el módulo RC_WIRE, este módulo depende directamente del subfiltro, por tanto se ha implementado la técnica usada en la Figura 80.

En la Figura 85 podemos ver el esquema RTL del módulo (subfilter = 0).

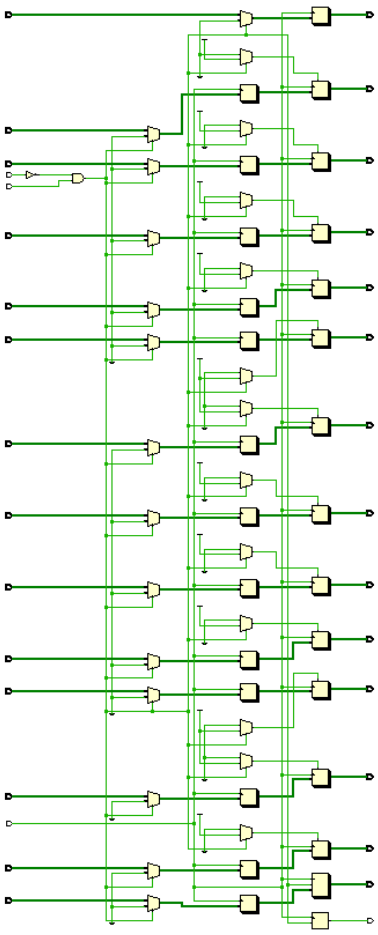

Figura 85: Esquema RTL del módulo RC_DELAY

3.2.2.4. RC_SUBFILTER

El módulo RC_SUBFILTER implementa el subfiltro completo (Figura 86)

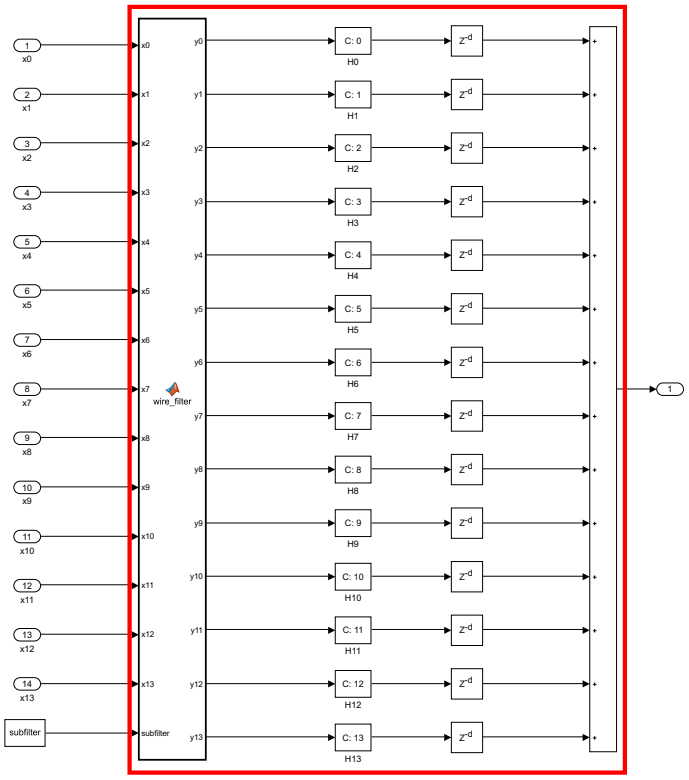

Figura 86: Correspondencia entre Simulink y Verilog del módulo RC_SUBFILTER

En él se instancian los módulos visto anteriormente y además se realiza la suma de todos los canales para obtener una única salida.

Esta suma se ha realizado en modo árbol y segmentando, con el fin de evitar problemas de tiempos (Figura 87).

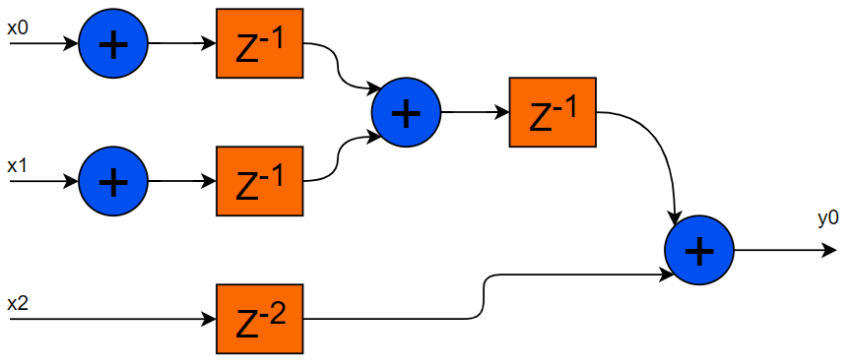

Figura 87: Ejemplo de suma en árbol

Este módulo consta de 14 entradas y una única salida, a parte de las señales genéricas anteriormente nombradas (clk, reset, etc), además de dos parámetros de configuración, el tamaño de entrada/salida (Win) y el número de subfiltro.

Cabe destacar que tras el módulo RC_FIR, los datos han crecido bastante debido a las operaciones aritméticas, por tanto, tras las sumas se ha truncado la salida para dejarlo en formato [25 24], es decir, igual que la entrada (Win):

```
assign oDATA = r_oDATA[Win+Wcoeff-2:Wcoeff-1];
```

En la Figura 88 podemos ver el esquema RTL del módulo (subfilter = 0).

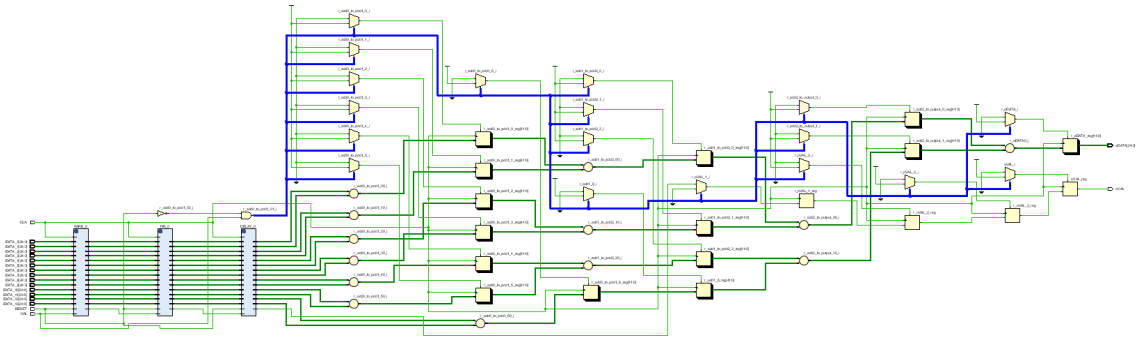

Figura 88: Esquema RTL del módulo RC_SUFBILTER

3.2.2.5. RC_FILTER

El módulo RC_FILTER se corresponde con el filtro RC completo (Figura 89).

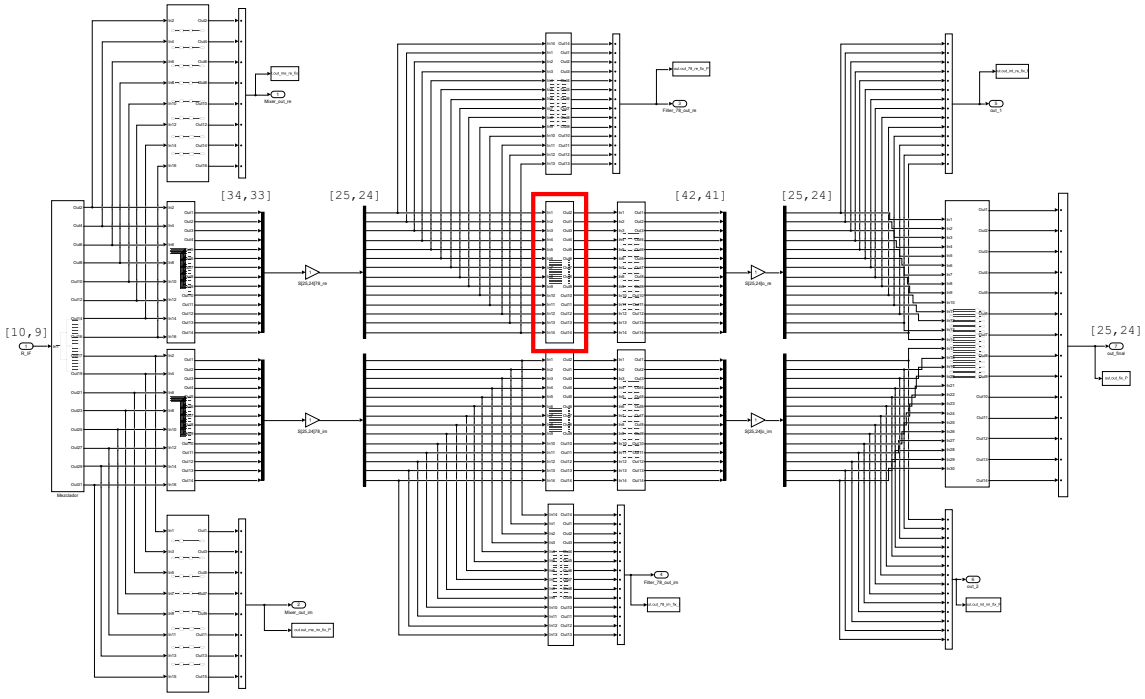

Figura 89: Correspondencia entre Simulink y Verilog del módulo RC_FILTER

En este módulo se ha instanciado 14 veces el módulo RC_SUBFILTER indicándole a cada uno el parámetro subfilter correspondiente.

En la Figura 90 podemos ver el esquema RTL del módulo.

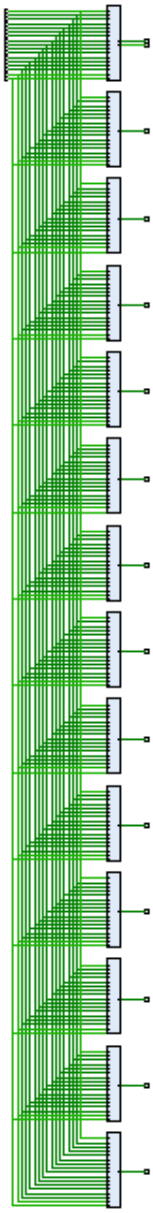

Figura 90: Esquema RTL del módulo RC_FILTER

3.2.2.6. RC_FILTER_TOP

El módulo RC_FILTER_TOP se encarga de agrupar el filtro RC del canal I y del canal Q (Figura 91).

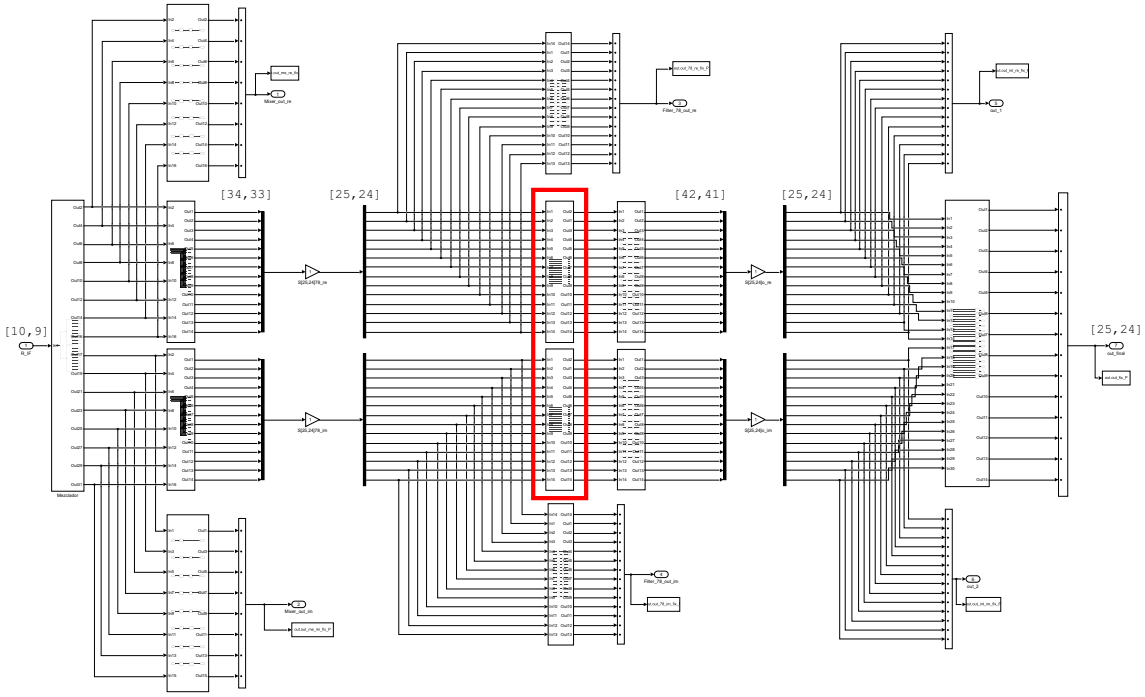

Figura 91: Correspondencia entre Simulink y Verilog del módulo RC_FILTER_TOP

Para implementar ambos canales se ha instanciado dos veces el módulo RC_FILTER.

En la Figura 92 podemos ver el esquema RTL del módulo.

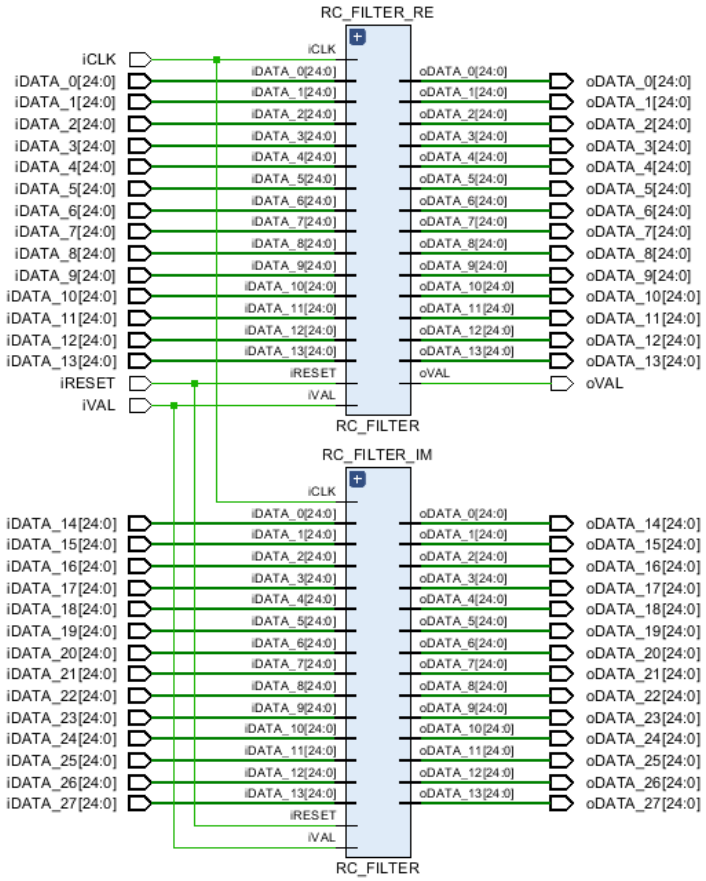

Figura 92: Esquema RTL del módulo RC_FILTER_TOP

3.2.3. Filtro de cambio de tasa por 7/8

Al igual que el filtro RC, el filtro diezmador está compuesto por una serie de módulos que se asemejan mucho a lo visto en el modelo de Matlab/Simulink. Los módulos que lo componen son DECIM_WIRE, DECIM_FIR, DECIM_DELAY, DECIM_SUBFILTER, DECIM_FILTER y DECIM_FILTER_TOP.

3.2.3.1. DECIM_WIRE

Para establecer la equivalencia entre el modelo de Matlab y el modelo en Verilog, esta vez nos basaremos en la Figura 53.

Este módulo, como su propio nombre indica, se encargará de implementar el cableado de los subfiltros (Figura 93).

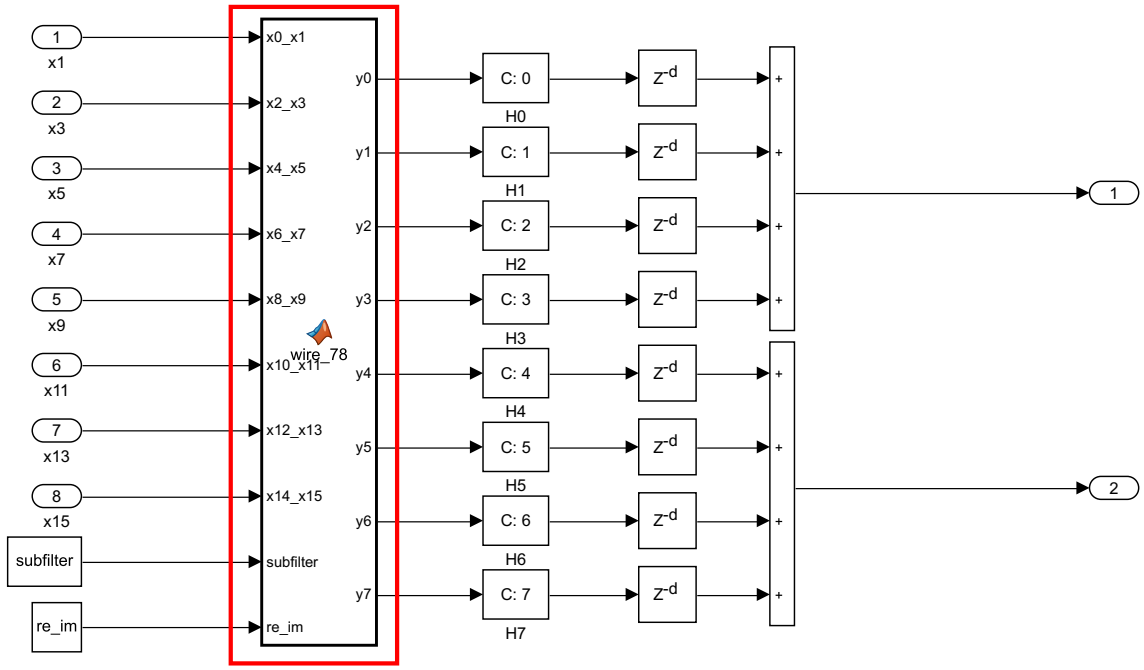

Figura 93: Correspondencia entre Simulink y Verilog del módulo DECIM_WIRE

Al igual que el módulo RC_WIRE, este se encarga de asignar las entradas a los filtros FIR.

Este módulo también se ha hecho de manera parametrizable, de manera que deberemos indicarle el tamaño de los datos de entrada (Win), el número de subfiltro y en que canal estamos (I o Q).

Aquí también se ha hecho uso del comando generate para generar código parametrizable (Figura 80).

En la Figura 94 podemos observar el esquema RTL de este módulo (re_im = 0; subfilter = 0).

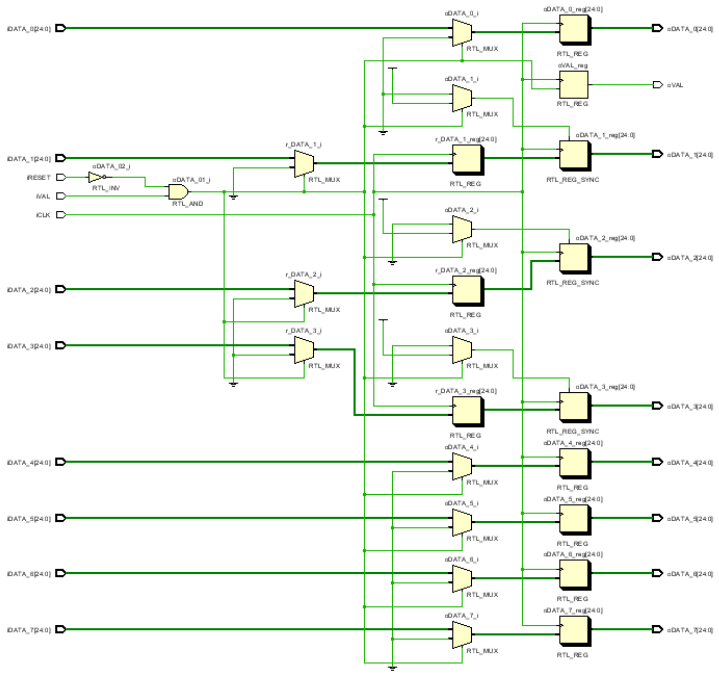

Figura 94: Esquema RTL del módulo DECIM_WIRE

3.2.3.2. DECIM_FIR

Este módulo se encarga de implementar la etapa de filtrado del filtro diezmador (Figura 95).

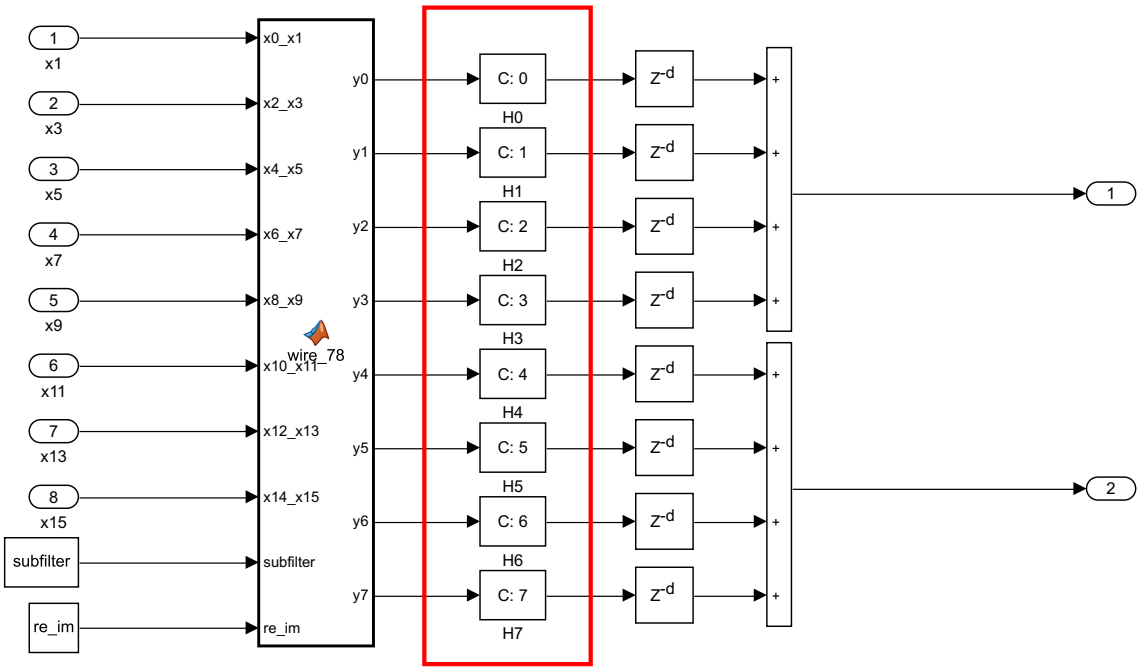

Figura 95: Correspondencia entre Simulink y Verilog del módulo DECIM_FIR

A diferencia del módulo RC_FIR, este es diferente para cada subfiltro y para cada canal (I o Q), por lo que, nuevamente, haremos uso del comando generate.

En esta ocasión, solo disponemos de un coeficiente por filtro, por lo que Vivado se encargará de eliminar las ramas que no se utilicen. El tamaño final de la salida deberá ser de Win+Wcoeff, ya que no hacemos uso de sumadores.

Podemos observar el esquema RTL generado en la Figura 96 (re_im = 0; subfilter = 0).

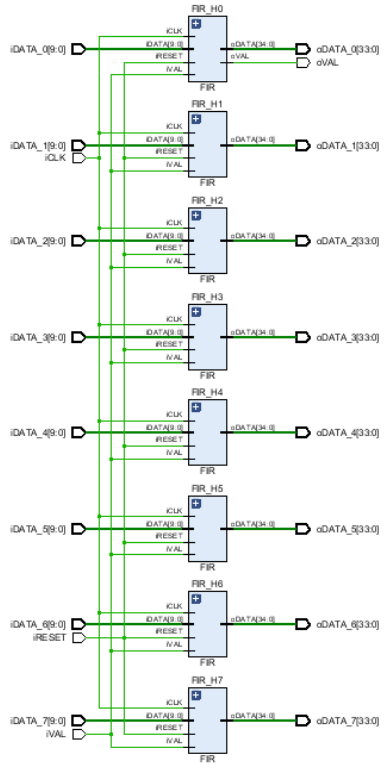

Figura 96: Esquema RTL del módulo DECIM_FIR

3.2.3.3. DECIM_DELAY

La etapa de retardo del filtro diezmador se implementa en este módulo (Figura 97).

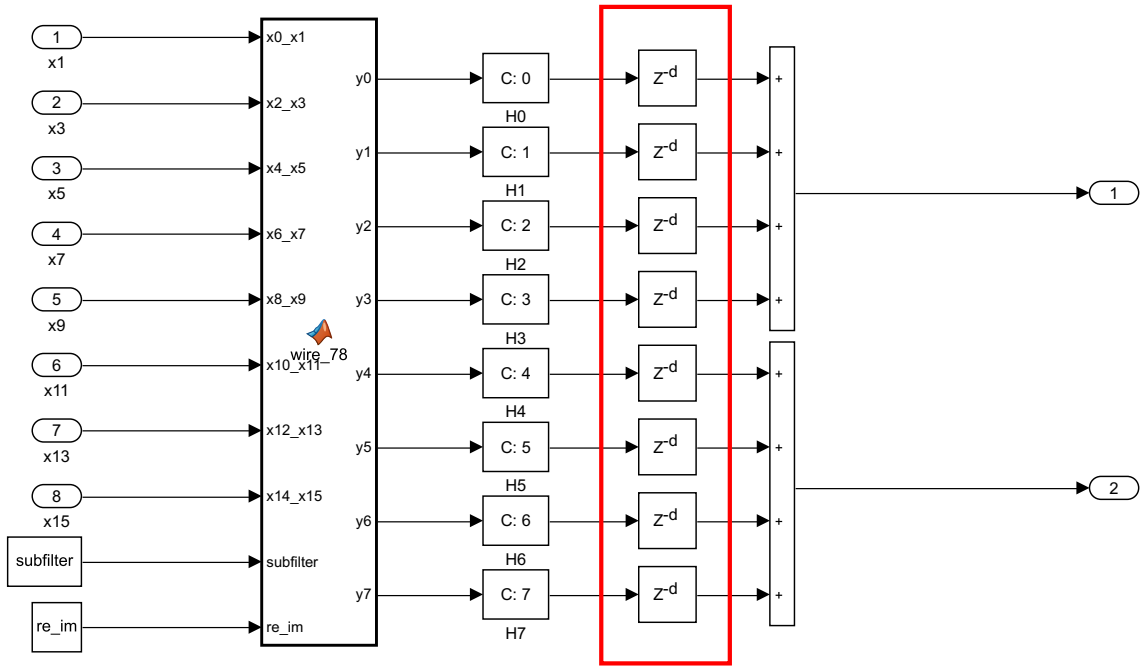

Figura 97: Correspondencia entre Simulink y Verilog del módulo DECIM_DELAY

Al igual que en el resto de módulos vistos en este filtro, este también depende del canal (I o Q) y del número de subfiltro, por tanto, también se ha hecho uso del comando generate para generar el código correspondiente.

La Figura 98 muestra el esquema RTL de este módulo (re_im = 0; subfilter = 0).

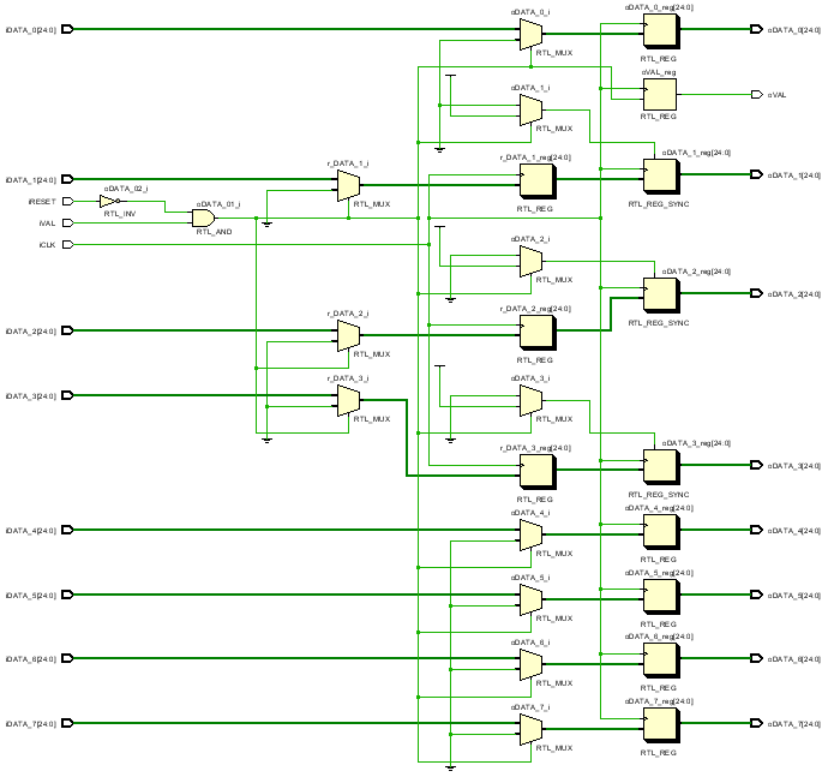

Figura 98: Esquema RTL del módulo DECIM_DELAY

3.2.3.4. DECIM_SUBFILTER

Al igual que RC_SUBFILTER, DECIM_SUBFILTER instancia los módulos anteriormente vistos con el fin de crear el subfiltro genérico (Figura 99).

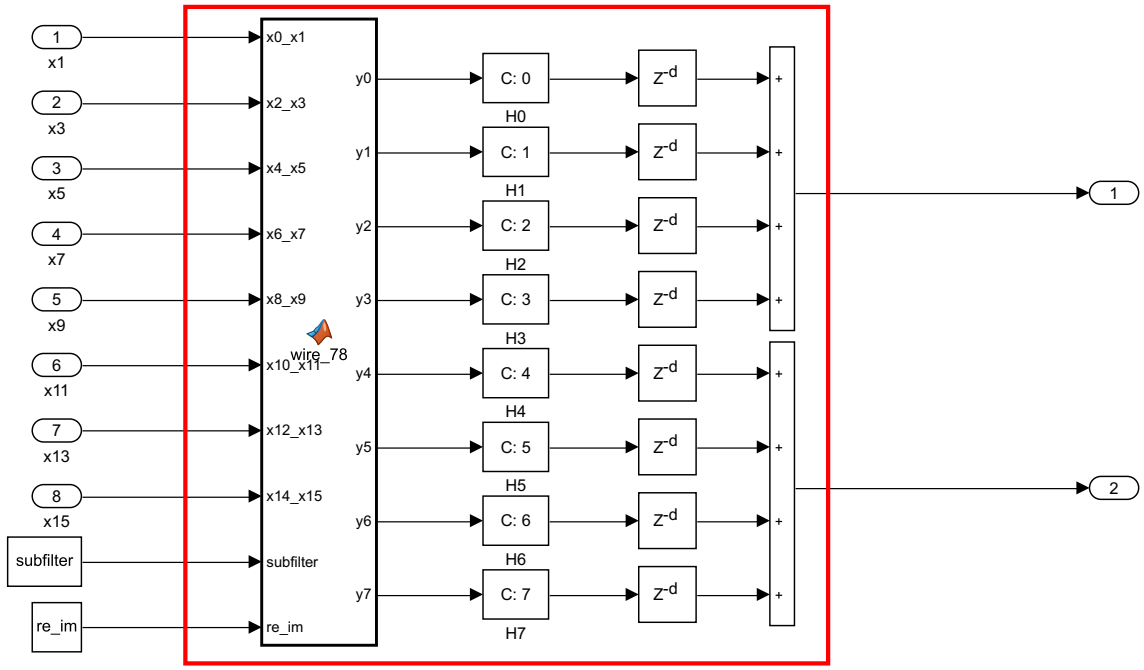

Figura 99: Correspondencia entre Simulink y Verilog del módulo DECIM_SUBFILTER

A partir de la última imagen, podemos observar que en esta ocasión tenemos dos salidas, las cuales se componen de la suma de los cuatro primeros y los cuatro últimos canales.

Estas sumas también se han implementado en árbol para evitar futuros problemas de tiempos.

Debido al crecimiento de los datos se han truncado los datos de salida a un formato [25 24]:

```
assign oDATA_0 = r_oDATA_0[Win+Wcoeff-2:Win-1];
assign oDATA_1 = r_oDATA_1[Win+Wcoeff-2:Win-1];
```

En la Figura 100 se muestra el esquema RTL de este módulo (re_im = 0; subfilter = 0).

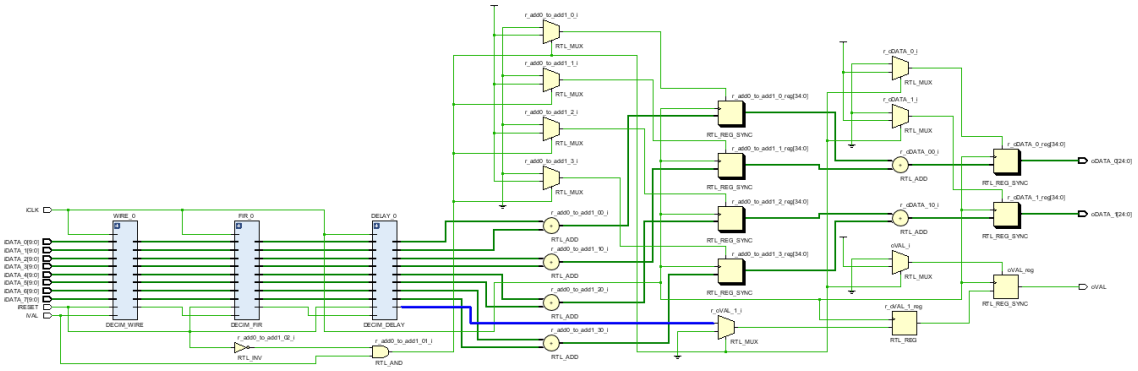

Figura 100: Esquema RTL del módulo DECIM_SUBFILTER

3.2.3.5. DECIM_FILTER

Este módulo se corresponde con el filtro diezmador completo (Figura 101).

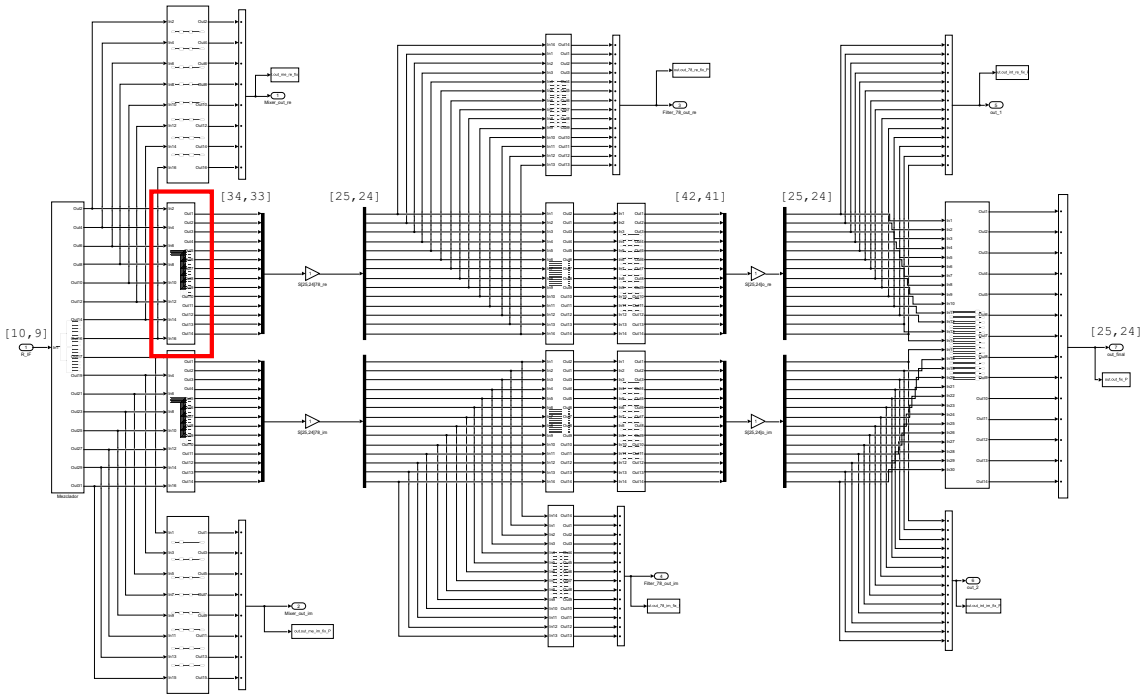

Figura 101: Correspondencia entre Simulink y Verilog del módulo DECIM_FILTER

En él se instancia 7 veces DECIM_SUBFILTER indicándole a cada uno de ellos el parámetro subfilter correspondiente.

Al contrario que RC_FILTER, este módulo es diferente según el canal en el que esté, por tanto, sigue dependiendo del parámetro re_im.

La Figura 102 muestra el esquema RTL del módulo (re_im = 0).

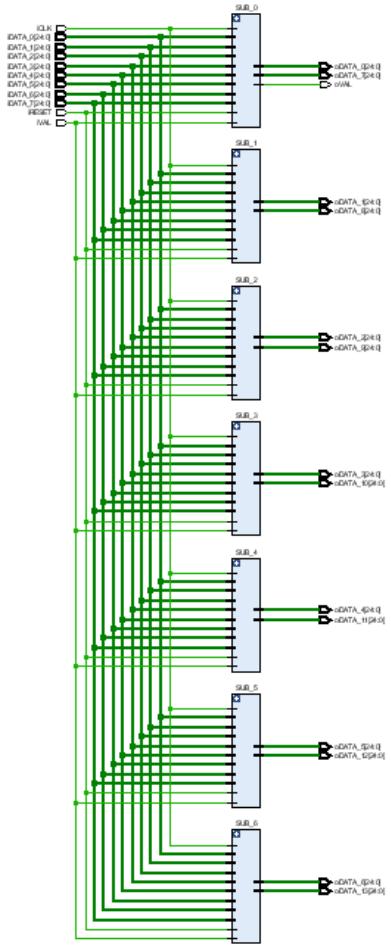

Figura 102: Esquema RTL del módulo DECIM_FILTER

3.2.3.6. DECIM_FILTER_TOP

Finalmente, este módulo recoge los dos filtros diezmadores necesarios para completar el modelo (Figura 103).

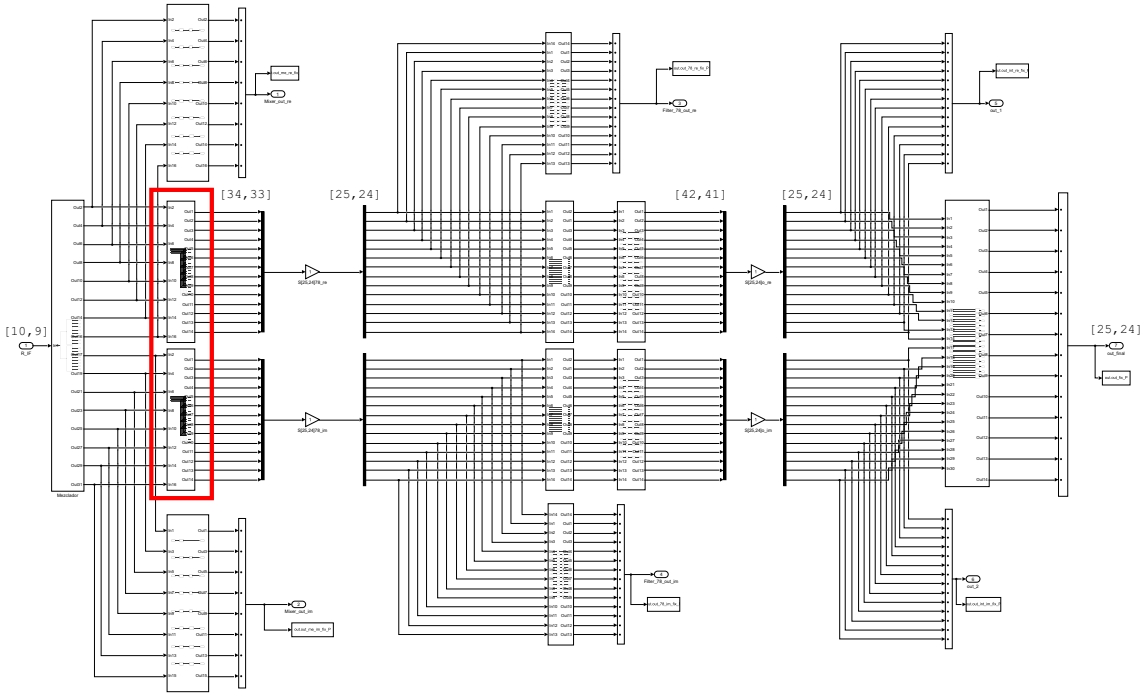

Figura 103: Correspondencia entre Simulink y Verilog del módulo DECIM_FILTER_TOP

En él tan solo instanciamos dos veces DECIM_FILTER indicando a cada uno el parámetro re_im correspondiente.

El resultado final se puede observar en la Figura 104.

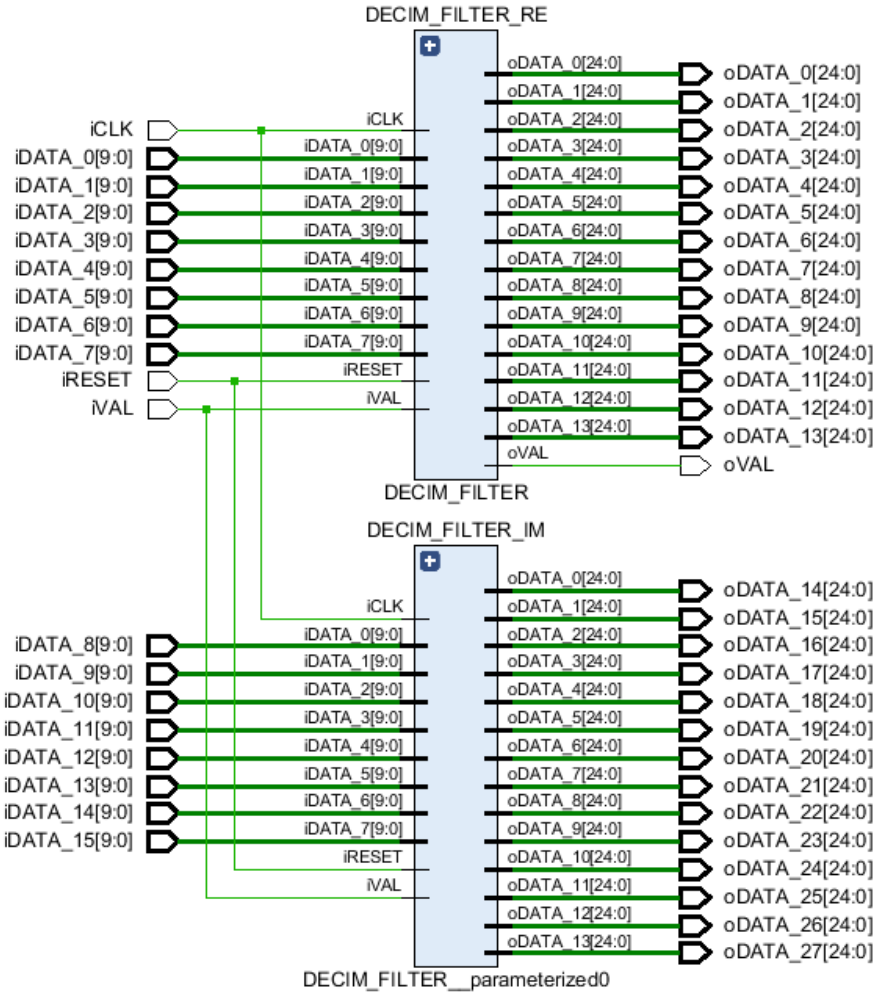

Figura 104: Esquema RTL del módulo DECIM_FILTER_TOP

3.2.4. Mezclador simplificado por $f_s/4$

Tras las optimizaciones realizadas en el apartado 3.1.4.1, el mezclador se ha visto reducido a distribuir las muestras de entradas entre el canal I y el canal Q.

El ADC de nuestra FPGA dispone de 4 canales a 1.25 GHz y cada uno de estos canales dispone de otros 4 a una frecuencia de 312.5 MHz, es decir, un total de 16. Esto significa que es el propio ADC el que cumple la función de mezclador, por tanto, este no necesita ser implementado en Verilog.

3.2.5. Receptor completo

Una vez codificado los módulos anteriores, se han instanciado en el módulo top, QAM_TOP.

En la Figura 105 se muestra el esquema RTL del mismo.

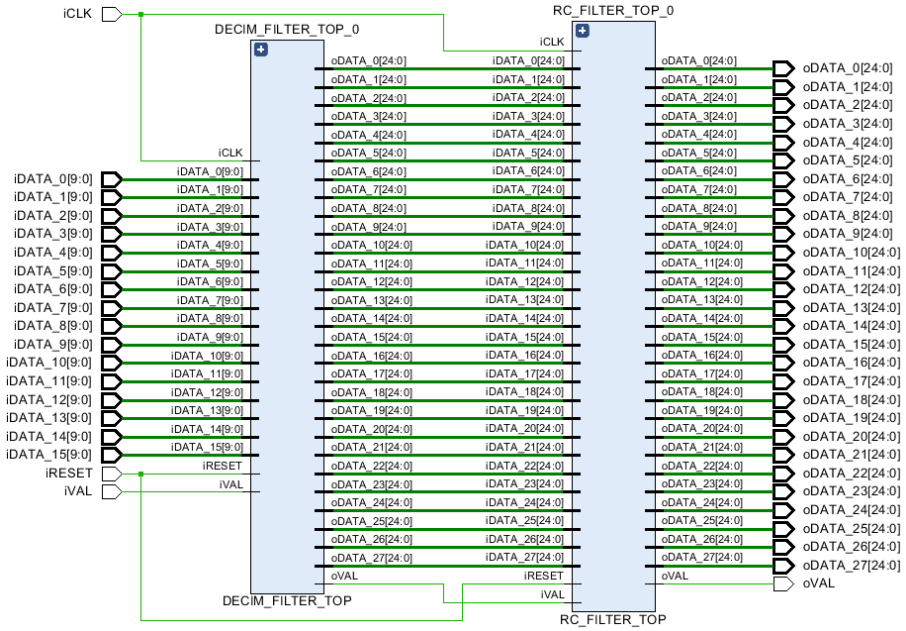

Figura 105: Esquema RTL del módulo QAM_TOP

3.2.5.1. Análisis de tiempo

Una vez se ha realizado el diseño completo, se ha analizado los tiempos con el fin de comprobar que este es capaz de funcionar a la frecuencia exigida (312.5 MHz).

Para ello hemos generado un archivo .xdc que contiene la restricción de tiempo:

```
create_clock -period 3.2 -name clk_QAM -waveform {0.000 1.6} [get_ports iCLK]
```

Esto generará un reloj de un periodo determinado y lo conectará a la entrada del reloj de QAM_TOP (iCLK). En la Figura 106 podemos ver el reloj generado.

Name	Waveform	Period (ns)	Frequency (MHz)
clk_QAM	{0.000 1.600}	3.200	312.500

Figura 106: Reloj generado durante el análisis de tiempos

Una vez hecho esto deberemos sintetizar el proyecto y generar un informe de tiempos (Figura 107)

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.492 ns	Worst Hold Slack (WHS):	0.049 ns	Worst Pulse Width Slack (WPWS):	1.250 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	170823	Total Number of Endpoints:	170823	Total Number of Endpoints:	55636

All user specified timing constraints are met.

Figura 107: Informe de tiempos para una frecuencia de 312.5 MHz

Tal y como se observa, disponemos un slack de 1.492 ns, esto quiere decir que nuestro proyecto es capaz de funcionar con un reloj de periodo 3.2 – 1.492 = 1.708 ns, es decir, 585 MHz, por tanto, cumplimos las restricciones de tiempo especificadas.

3.2.5.2. Recursos utilizados

A la hora de implementar un diseño en una placa es conveniente saber el número de recursos utilizados. Para conocer este dato deberemos sintetizar el módulo en concreto y observar el informe generado por Vivado. A continuación, se mostrará los recursos utilizados por cada módulo.

Cabe destacar que la mayoría de módulos están parametrizados y durante la síntesis se aplica el parámetro por defecto, dado que los recursos pueden variar según el parámetro utilizado, los recursos consumidos por la mayoría de módulos (sobre todo los de más bajo nivel) son una aproximación. Además, debemos contar con las optimizaciones realizadas por el compilador, las cuales pueden hacer que se consuman menos recursos de los esperados. Por supuesto, los recursos de los módulos tops de nuestro proyecto (RC_FILTER_TOP, DECIM_FILTER_TOP y QAM_TOP) muestran los recursos reales utilizados, ya que estos no están parametrizados.

En la Tabla 7 se puede observar los recursos consumidos por el filtro RC.

Filtro RC			
Módulos	**LUT**	**FF**	**DSP**
RC_WIRE	2	351	0
RC_FIR	2	371	41
RC_DELAY	2	676	0
RC_SUBFILTER	877	2432	41
RC_FILTER	8004	22051	574
RC_FILTER_TOP	16014	44087	1148

Tabla 7: Recursos utilizados en los módulos del filtro RC

RC_FIR incluye 14 filtros FIR donde 13 son de 3 coeficientes y 1 es de 2 coeficientes, el cual da como resultado $13 \times 3 + 1 \times 2 = 41$ multiplicadores.

RC_FILTER incluye 14 veces el módulo RC_SUBFILTER (el cual incluye RC_FIR), por tanto $14 \times 41 = 574$ multiplicadores.

Por último, RC_FILTER_TOP incluye 2 veces el módulo RC_FILTER, es decir, $574 \times 2 = 1148$ multiplicadores.

La Tabla 8 reúne los recursos utilizados por el filtro diezmador.

Filtro Diezmador			
Módulos	**LUT**	**FF**	**DSP**
DECIM_WIRE	2	81	0
DECIM _FIR	2	276	8
DECIM _DELAY	2	276	0
DECIM _SUBFILTER	208	920	8
DECIM _FILTER	3134	10954	64
DECIM_FILTER_TOP	2545	10310	98

Tabla 8: Recursos utilizados en los módulos del filtro diezmador

DECIM_FIR está compuesto, según el subfiltro, por 6 o por 8 filtros FIR, y cada uno con un único coeficiente, $8 \times 1 = 8$ multiplicadores.

DECIM_FILTER incluye 7 veces el módulo DECIM_SUBFILTER (el cual incluye DECIM_FIR), en este caso el valor por defecto del parámetro re_im es 0, es decir, estamos en el canal Q, por tanto, hay 3 subfiltros con 6 filtros FIR y 4 subfiltros con 8 filtros FIR. El número de multiplicadores es $3 \times 6 + 4 \times 8 = 50$.

Por último, DECIM_FILTER_TOP incluye dos veces el módulo DECIM_FILTER, uno equivale al canal I (4 subfiltros con 6 filtros FIR) y el otro al canal Q (3 subfiltros con 6 filtros FIR), es decir, el número de multiplicadores es $(4 \times 6 + 3 \times 8) + (3 \times 6 + 4 \times 8) = 98$.

Finalmente, la Tabla 9 contiene los recursos del receptor completo.

Receptor QAM			
Módulo	**LUT**	**FF**	**DSP**
QAM_TOP	18541 (6.11%)	54401 (8.96%)	1246 (44.5%)

Tabla 9: Recursos utilizados en los módulos del receptor completo

QAM_TOP instancia RC_FILTER_TOP y DECIM_FILTER_TOP, por tanto, el número de multiplicadores es 1148 + 98 = 1246.

Tal y como se observa en la tabla anterior, obtenemos que el receptor ocupa un 6.11 % del total de LUTs, un 8.96 % del total de FF y un 44.5 % del total de DSP slices.

3.3. Simulación y verificación

A lo largo de este apartado se explicará la metodología seguida a la hora de comprobar que el modelo implementado en Verilog funciona correctamente.

Lo primero que se ha hecho, con el fin de verificar el modelo, ha sido crear un modelo de refe-

rencia que nos permita comparar el modelo de HDL con uno equivalente. Por ello, y como se ha explicado anteriormente, se ha creado un modelo de precisión finita a partir de un modelo ideal. Esto es necesario ya que el modelo ideal trabaja con punto flotante, mientras que la FPGA trabaja con punto fijo. Se podría utilizar el modelo ideal como referencia, pero durante la verificación deberíamos tener en cuenta el error entre ambos modelos. Mediante el uso modelo de precisión finita no tenemos que tener en cuenta el error, ya que ambos modelos deberían ser exactamente iguales.

En la Figura 108 se puede observar que para verificar el modelo de precisión finita deberemos introducir un estímulo en ambos modelos y deberemos comprobar que el error está dentro de lo razonable. Si esto es así podemos tomar el modelo de precisión finita como referencia para verificar nuestro modelo HDL.

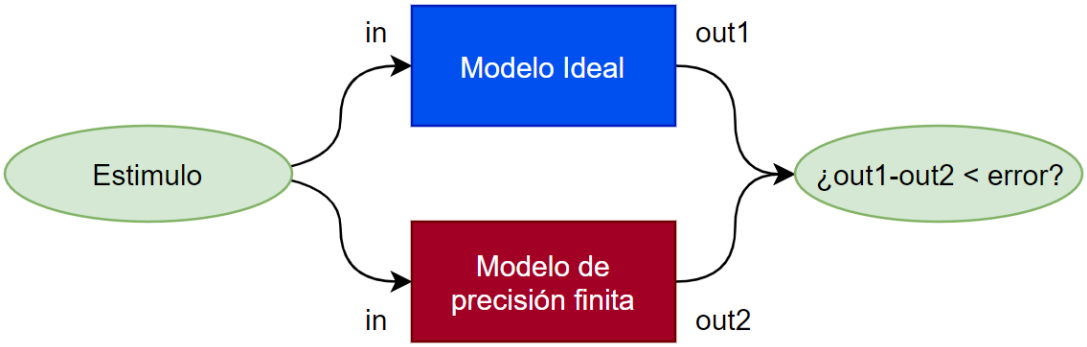

Figura 108: Modelo de referencia

Una vez disponemos de nuestro modelo de referencia, verificaremos el funcionamiento del modelo HDL de la misma manera (Figura 109). Introduciremos a ambos el mismo estímulo y compararemos sus salidas.

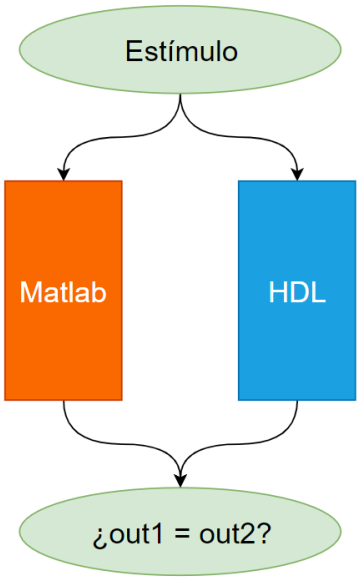

Figura 109: Verificación del modelo HDL

En este caso no se permite ningún error entre ambos modelos puesto que deben tener la misma cuantificación.

Por tanto, el procedimiento que deberemos seguir es el que se muestra en la Figura 110.

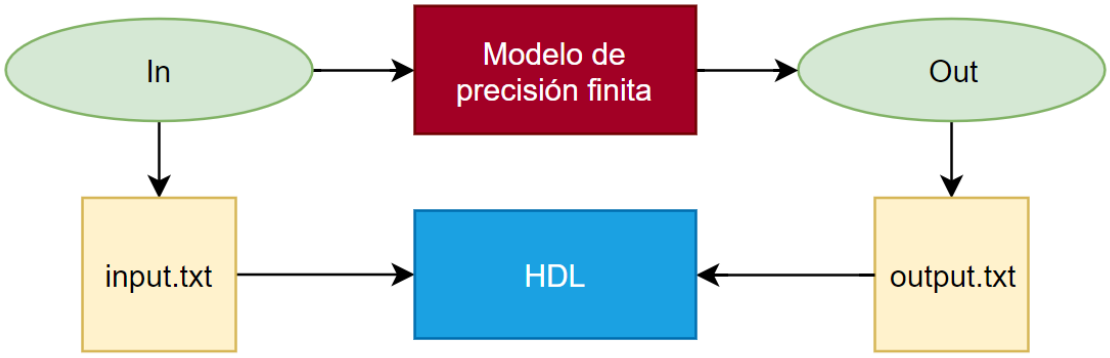

Figura 110: Generación de datos de entrada y de salida

Mediante Matlab/Simulink generaremos un estímulo sobre el modelo que queramos testear, este estímulo se guardará, al igual que la salida, en un archivo de texto. Posteriormente se creará un testbench que se encargará de leer los datos de entrada y de introducírselos a nuestro modelo HDL. Por último se encargará de comparar la salida del modelo con la salida almacenada en el archivo de texto. El modelo se simulará en ModelSim, el cual nos proveerá una salida gráfica y nos indicará si ha habido un error durante la verificación.

Esta metodología ha sido aplicada a todos los módulos vistos en el apartado 3.2, salvo en RC_FILTER_TOP y DECIM_FILTER_TOP, ya que la verificación de estos módulos no aporta ningún valor. De esta forma se facilita la depuración de errores.

A continuación, se irá entrando en detalle en cada uno de los procesos anteriormente mencionados.

3.3.1. Matlab

El proceso llevado a cabo en Matlab se realiza en tres pasos:

- Configuración del modelo y del tamaño de cuantificación.
- Ejecución del modelo y captura de datos.
- Cuantificación de los datos y generación de los archivos de texto.

Durante la configuración, deberemos generar los parámetros necesarios para que el modelo funcione correctamente, especificar el tamaño con el que deseamos cuantificar los datos de entrada, los datos de salida y, si fuese necesario, los coeficientes. También deberemos indicarle la ruta donde deseamos guardar los archivos de texto, que en este caso, con el fin de automatizar el proceso, se guardarán automáticamente en la carpeta del proyecto de ModelSim. Adicionalmente, y con el fin de evitar errores cuando metamos los coeficientes a mano en el modelo HDL, se generará un archivo de texto que contenga la cabecera de los coeficientes que deberemos utilizar en el módulo donde se asignen esos valores.

Una vez realizada la configuración, se ejecutará el modelo de Simulink, el cual exportará los datos

de entrada y de salida a Matlab.

Cuando haya terminado la ejecución, se ejecutará otro script de Matlab que se encargará de cuantificar los datos de entrada y de salida y, a partir de ellos, generará los archivos de texto correspondientes.

Todo este proceso queda resumido en la Figura 111.

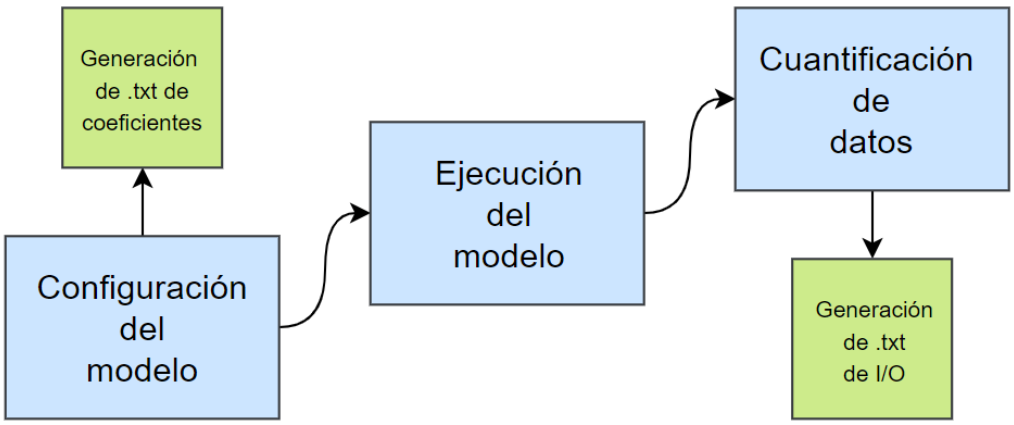

Figura 111: Proceso de generación de datos de entrada y salida

3.3.2. Testbench

El testbench se encargará de leer los datos de entrada y de enviárselos al módulo que está siendo testeado, asi como de comparar la salida del módulo con el archivo de texto correspondiente.

En este punto surge la problemática de que no sabemos cuándo empezar a tomar como válida la señal de salida, ya que hay retardos de por medio. Este problema se ha tenido en cuenta desde el principio y por ello se ha creado la señal iVAL. Esta señal tiene en cuenta todos los retardos internos del módulo y si introducimos un 1 por ella, cuando obtengamos un 1 por oVAL, significará que a partir de ese momento todos los datos de salida son válidos (Figura 112).

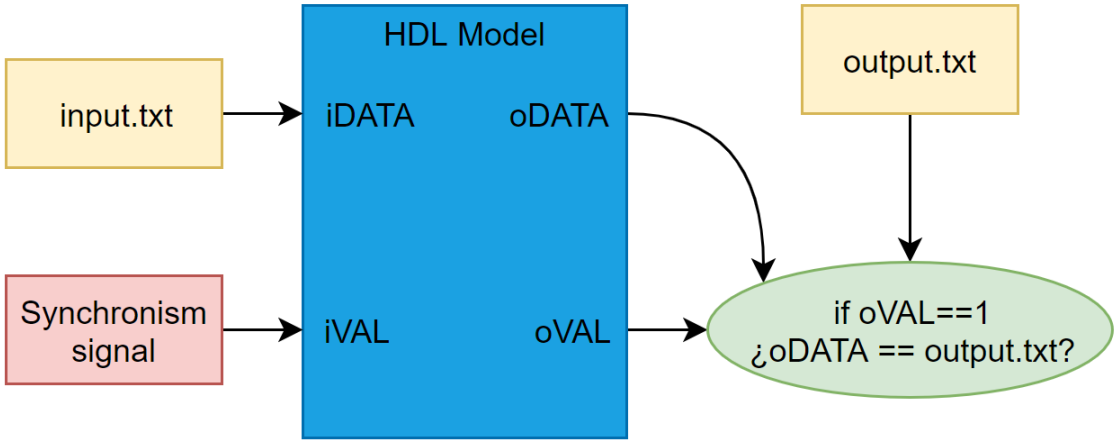

Figura 112: Testbench del modelo HDL

A partir de ese momento se comienzan a comparar las muestras y en caso de que sean distintas aumentará un contador de errores cuya función es la de indicarnos el número de errores totales.

3.3.3. Modelsim

Por último, utilizaremos ModelSim para simular el modelo y obtener las salidas gráficamente.

ModelSim nos permite observar cualquier señal del modelo, permitiéndonos de esta manera depurarlo en caso de que encontremos algún error.

A continuación, se va a mostrar el resultado de ModelSim para el módulo QAM_TOP (Figura 113).

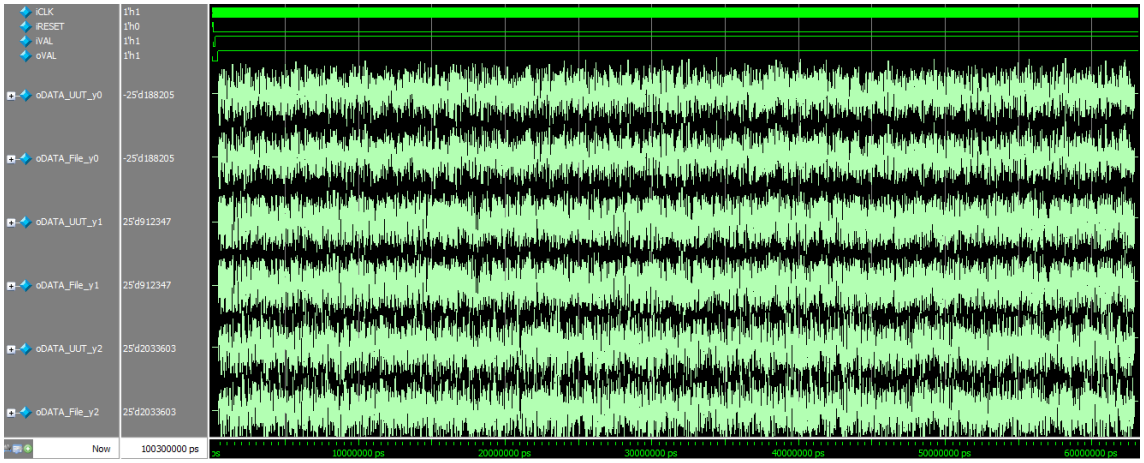

Figura 113: Simulación del módulo QAM_TOP

Como podemos observar, las salidas de nuestro modelo (oDATA_UUT_yX) coincide a la perfección con los datos del archivo de texto (oDATA_File_yX).

Si comprobamos la lista de errores, vemos que de las 6252 muestras analizadas no se ha encontrado ni un solo error:

```
# Channel y0: Number of checked samples        6252

#             Number of errors                    0

# Channel y1: Number of checked samples        6252

#             Number of errors                    0

# Channel y2: Number of checked samples        6252

#             Number of errors                    0

...

# Channel y26: Number of checked samples        6252
```

```
#              Number of errors                   0

# Channel y27: Number of checked samples        6252

#              Number of errors                   0

# Number of errors in all channels                0
```

Capítulo 4

Conclusiones

En este capítulo se expondrán las conclusiones del proyecto y se hablará de las posibles mejoras futuras del mismo.

4.1. Conclusiones

El presente proyecto ha tenido como propósito realizar el diseño e implementación de un receptor QAM en FPGA, el cual ha permitido lograr una serie de objetivos:

- Se ha estudiado la modulación QAM, su funcionamiento y sus diferentes aplicaciones. Nos hemos centrado en la etapa de recepción, haciendo hincapié en los principales componentes de este, el mezclador, el filtro de cambio de tasa y el filtro adaptativo.
- Se han estudiado las diferentes arquitecturas a aplicar sobre la modelización de nuestro receptor, teniendo en cuenta las características del sistema.
- Se ha diseñado en Matlab/Simulink los elementos descritos a lo largo del proyecto tanto en su forma no paralelizada, como en su forma paralelizada, con el fin de comparar ambas arquitecturas y comprobar la equivalencia de ambos modelos.
- Se han combinado los elementos anteriormente descritos en un único modelo que da forma al receptor. Con la arquitectura elegida es capaz de funcionar a la frecuencia exigida (5 GHz).
- Se ha simplificado el modelo con el fin de optimizar recursos, se ha parametrizado para facilitar la depuración del mismo y finalmente se ha pasado de un modelo de punto flotante a un modelo de punto fijo cuantificado de forma óptima de cara a los DSPs de la FPGA.
- Se ha implementado módulo a módulo los elementos antes mencionados en un lenguaje HDL.
- Al igual que en Matlab/Simulink, se unieron todos los elementos ya modelados en HDL para formar el receptor QAM, el cual satisface la velocidad exigida.
- Se ha verificado el funcionamiento de dicho receptor mediante el uso de una metodología robusta, basada en el uso de un modelo de referencia.

4.2. Trabajo futuro

Al tratarse de un Trabajo de Fin de Master, el tiempo empleado es limitado, sin embargo, se proponen una serie de mejoras a aplicar:

- Uso de técnicas de optimización con el fin de reducir los recursos utilizados.
- Parametrización del modelo de forma que, a partir de una serie de parámetros, sea capaz de autogenerar el receptor QAM con una serie de características propias, permitiendo de esta manera filtros con un numero diferente de entradas, etc. Para ello probablemente habría que utilizar un lenguaje HDL más adecuado, que permita más libertad a la hora de parametrizar el sistema.
- Completar el receptor con los bloques de sincronismo de tiempo, fase y frecuencia para que pueda funcionar en condiciones reales de trasmisión.

Bibliografía

[1] *QAM Formats: 8-QAM, 16-QAM, 32-QAM, 64-QAM, 128-QAM, 256-QAM*. URL: https://www.electronics-notes.com/articles/radio/modulation/quadrature-amplitude-modulation-types-8qam-16qam-32qam-64qam-128qam-256qam.php.

[2] "The first Nyquist criterion". En: (). URL: https://dspillustrations.com/pages/posts/misc/the-first-nyquist-criterion.html.

[3] "Multirate Signal Procesing". En: (). URL: http://eeweb.poly.edu/iselesni/EL713/zoom/mrate.pdf.

[4] *7 Series DSP48E1 Slice:* URL: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.

[5] *Xilinx Virtex-7 FPGA VC707*. URL: https://www.xilinx.com/products/boards-and-kits/ek-v7-vc707-g.html#hardware.