ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

Diseño e Implementación de un Sistema de Comunicaciones con Tasa de Muestreo Sub-Nyquist empleando Técnicas de Muestreo Compresivo

Autor: Rodrigo Serna Pérez

Director: Javier Matanza Domingo

Madrid

Junio 2018

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Diseño e Implementación de un Sistema de

Comunicaciones con Tasa de Muestreo

Sub-Nyquist empleando Técnicas de Muestreo

Compresivo

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2017/18 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Rodrigo Serna Pérez Fecha: ……/ ……/ ……

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Javier Matanza Domingo Fecha: ……/ ……/ ……

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D._____

DECLARA ser el titular de los derechos de propiedad intelectual de la obra:

_____, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e

intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a …….. de ……... de …….

ACEPTA

Fdo……

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

Diseño e Implementación de un Sistema de Comunicaciones con Tasa de Muestreo Sub-Nyquist empleando Técnicas de Muestreo Compresivo

Autor: Rodrigo Serna Pérez

Director: Javier Matanza Domingo

Madrid

Junio 2018

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE COMUNICACIONES CON TASA DE MUESTREO SUB-NYQUIST EMPLEANDO TÉCNICAS DE MUESTREO COMPRESIVO

Autor: Serna Pérez, Rodrigo.
Director: Matanza Domingo, Javier.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

En este proyecto se plantea el diseño de un sistema de comunicaciones de alto ancho de banda utilizando técnicas de muestreo compresivo para de esta forma poder utilizar un muestreador mucho más lento que el necesario por el criterio de Nyquist. También se implementa el sistema en un circuito digital.

Palabras clave: Muestreo Compresivo, FPGA, Banda Ancha, sub-Nyquist

1. Introducción

Las comunicaciones de banda alta son interesantes para aplicaciones como Internet Of Things, ya que permiten usar baja potencia de transmisión y dan alta resistencia a las interferencias. Pero, por otro lado, el empleo de un alto ancho de banda requiere un muestreador muy rápido, lo cual implica un alto consumo. Usando técnicas de muestreo compresivo se consigue utilizar un muestreador mucho más lento que el que indica el criterio de Nyquist.

2. Definición del proyecto

El proyecto consiste en estudiar cómo pueden aplicarse las técnicas del estado del arte de muestreo compresivo a un sistema de comunicaciones de banda ancha que permita aprovecharse sus ventajas a la hora de usar un muestreador más lento. Para ello, se realiza una simulación en Matlab del sistema de comunicaciones completo. Además, se implementa en un circuito digital una propuesta de un receptor para este sistema.

3. Descripción del sistema

Aunque la simulación desarrollada incluye todos los elementos del sistema de comunicaciones, la mayor parte del trabajo se centra en el receptor ya que es el que contiene la mayor parte del procesado de la señal. Un esquema del receptor desarrollado se puede ver en la Ilustración 1.

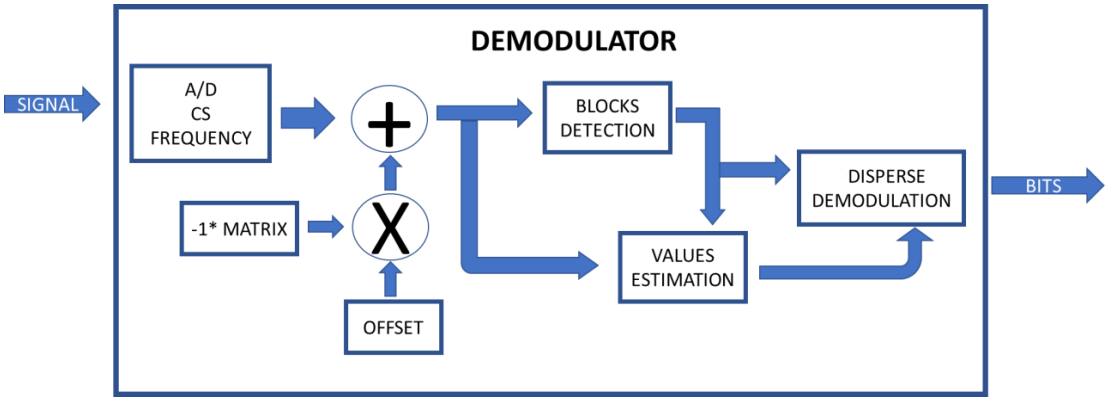

Ilustración 1 – Esquema de un receptor con CS

4. Resultados

Se demostró que es posible poner en funcionamiento un sistema de comunicaciones que utiliza un muestreador mucho más lento que el que indica el criterio de Nyqusit. Además, se implementó en lenguaje Verilog y se simuló para comprobar su correcto funcionamiento.

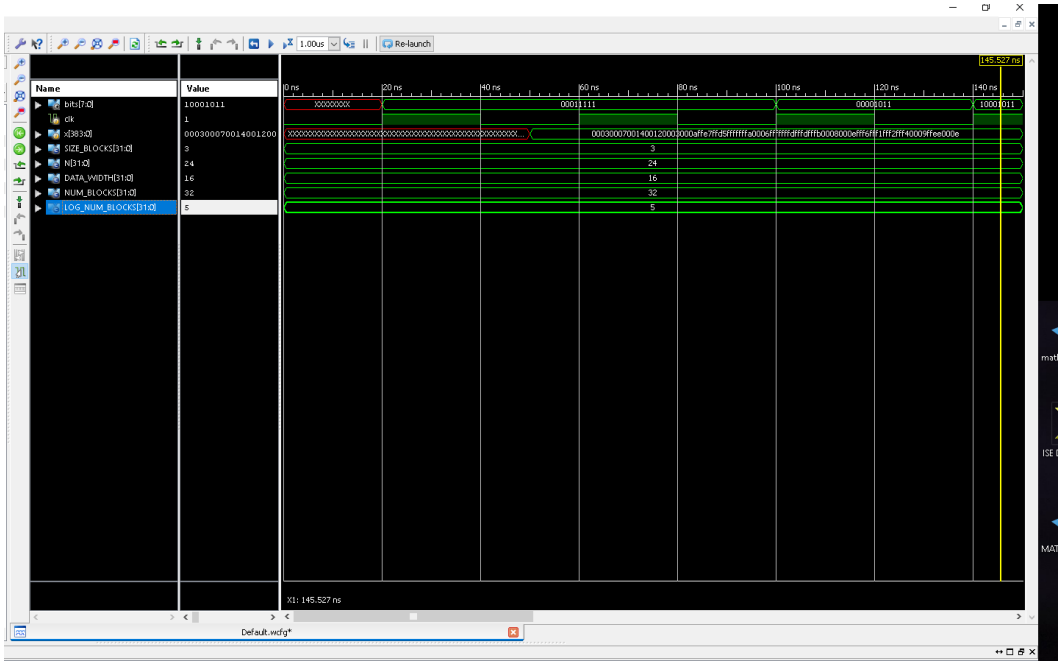

Ilustración 2 - Simulación del bucle completo de la etapa de frecuencias

5. Conclusiones

Como producto de este trabajo, se pudo probar que es posible transmitir en altas frecuencias usando un muestreador mucho más lento que el indicado por Nyquist pudiendo por tanto conseguir transmisiones de bajo consumo.

DESIGN AND IMPLEMENTATION OF A COMMUNICATIONS SYSTEM WITH SUB-NYQUIST SAMPLING FREQUENCY WITH COMPRENSIVE SAMPLING TECHNIQUES

Author: Serna Pérez, Rodrigo
Supervisor: Matanza Domingo, Javier

ABSTRACT

In this project the design of a high bandwidth communications system using compressive sampling techniques is proposed, in order to use a sampler much slower than the one required by the Nyquist criterion. The system is also implemented in a digital circuit.

Keywords: Compressive Sampling, FPGA, Wide Band, sub-Nyquist

1. Introduction

High band communications have interesting characteristics for applications such as Internet Of Things, since they allow to use low transmit power and give high resistance to interference. But, on the other hand, the use of a high bandwidth requires a very fast sampler, which implies a high consumption. Using compressive sampling techniques, it is possible to use a sampler much slower than the one indicated by the Nyquist criterion.

2. Definition of the project

The project consists of studying how state-of-the-art techniques of compressive sampling can be applied to a broadband communications system that allows to take advantage of its advantages when using a slower sampler. For this, a simulation in Matlab of the complete communications system is carried out. In addition, a proposal for a receiver for this system is implemented in a digital circuit.

3. Description of the system

Although the developed simulation includes all elements of the communication system, most of the work is focused on the receiver since it is the one that contains most of the processing of the signal. A scheme of the developed receiver can be seen in Illustration 1.

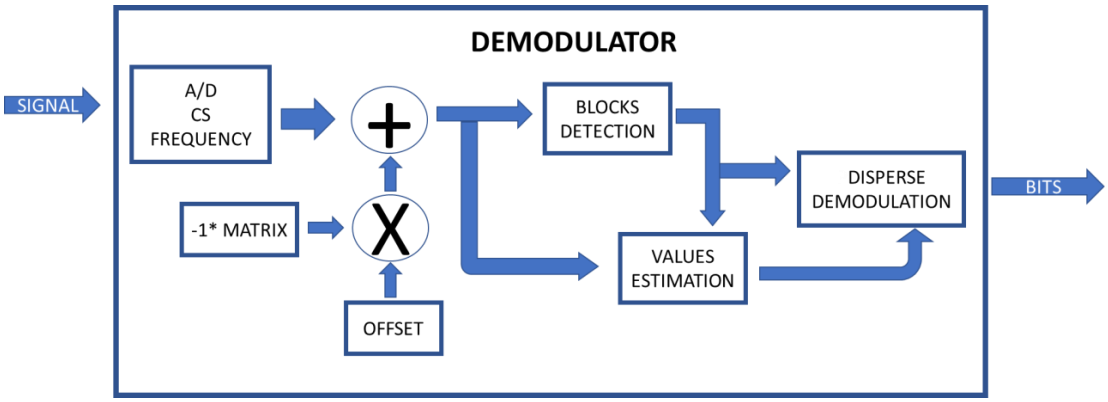

Illustration 1 Design of a receiver with CS

4. Results

It was shown that it is possible to put into operation a communications system that uses a sampler much slower than the one indicated by the Nyqusit criterion. In addition, it was implemented in the Verilog language and simulated to verify its correct functioning.

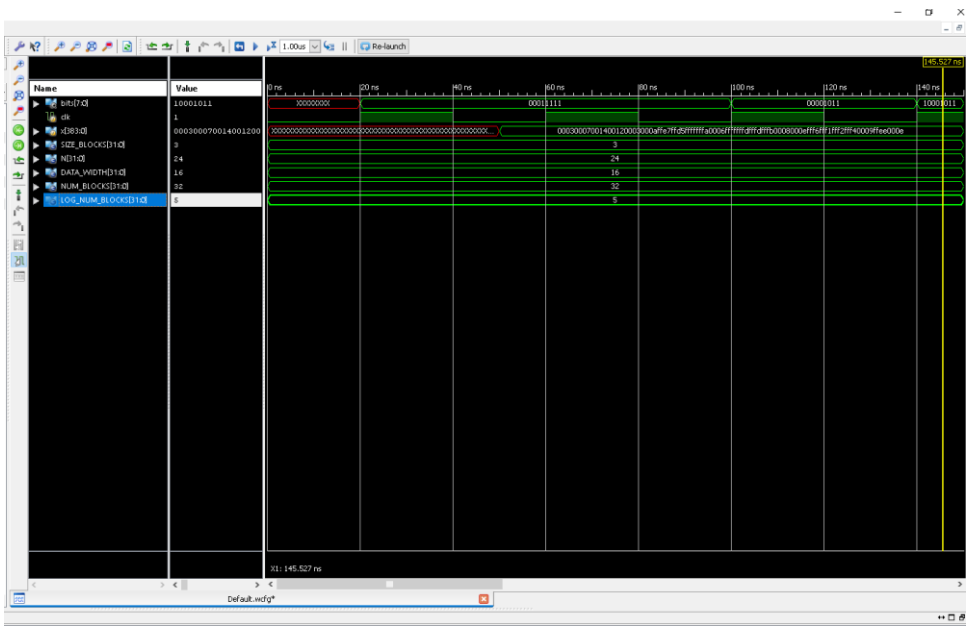

Illustration 2 Simulation of the behavior of the circuit

5. Conclusions

As product of this project, it was proved that it is possible to transmit at high frequencies using a sampler much slower than the one indicated by Nyquist, thus being able to obtain low power transmissions.

Índice de la memoria

Contenido

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

Las comunicaciones de banda ancha tienen ventajas que las hacen muy interesantes para aplicaciones que son tendencia, como pueden ser Internet Of Things. Esto se debe a que un ancho de banda alto permite transmitir usando baja potencia, además de que las dan una sensibilidad alta a las interferencias o a la presencia de multitrayectos..

Pero al mismo tiempo, puede ser un problema encontrar componentes que puedan funcionar a altas frecuencias y tengan un precio bajo. No solo eso, sino que el procesado de la señal de alta frecuencia requiere muestreadores muy rápidos los cuales además de tener un precio en el mercado muy alto (o incluso puede que la tecnología actual no permita acceder a frecuencias muy altas), el consumo necesario para funcionar sea demasiado elevado y por tanto haga imposible aplicarlo a aplicaciones de bajo consumo.

Las técnicas de muestreo compresivo (Compressive Sampling, CS) pueden ayudar en este último aspecto. Con ellas, es posible muestrear a frecuencias inferiores a la frecuencia de Nyquist y ser aun así posible recuperar la señal transmitida si se cumplen ciertas características. De esta forma, es posible muestrear a ratios cercanos al 1% del límite de Nyquist.

Este hecho da una importante ventaja a la hora de diseñar comunicaciones, ya que el hecho de usar un muestreador mucho más lento disminuye notablemente su consumo, lo que puede hacer que el sistema pueda ser aplicado en tecnologías de Internet Of Things o de otras aplicaciones que demandan un bajo consumo.

En este proyecto, se aplican técnicas de CS para lograr esta meta. Desgraciadamente, el modelado matemáticamente de CS es muy complejo y su resolución es difícilmente implementable para su ejecución en tiempo real. Por eso, es necesario aplicar simplificaciones que puedan permitir su aplicación.

Para comprobar que la solución es apta para aplicaciones reales, se ha implementado el modelo propuesto en un circuito digital, pudiendo de esta forma conocer los requisitos del dispositivo que ejecutará el algoritmo.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

La tecnología usada para simulación fue el lenguaje de programación Matlab, ya que permite operar con matrices fácilmente.

En cuanto la implementación, esta se realizó en lenguaje Verilog para descripción de circuitos digitales en una FPGA. La herramienta usada para esto es el IDE de Xilinx, ISE Design Suite.

Una FPGA (Field Programable Gate Array) es un dispositivo programable compuesto de cables y puertas lógicas que se conectan y configuran para formar el circuito lógico descrito en un lenguaje de descripción de hardware (HDL, Hardware Description Language) específico como VHDL o Verilog. Una FPGA puede reprogramarse para formar distintos circuitos digitales, por lo que es una herramienta muy adecuado para las tareas de prototipado.

La metodología para la programación de FPGAs es la siguiente [1]:

- Diseño digital del circuito usando bloques básicos como registros, multiplicadores o multiplexores, así como las señales que se van a usar.
- Descripción del circuito usando un lenguaje de descripción de hardware, como puede ser Verilog.
- El proceso de síntesis convierte el código escrito en elementos básicos (como puertas AND y OR) así como optimización del circuito. Este proceso se realiza automáticamente mediante herramientas con este fin.
- A partir del resultado de la síntesis del código, se consigue una secuencia de bits los cuales son cargados en la FPGA y forman el circuito descrito.

Además, es frecuente que se produzca un proceso de testeo y simulación que compruebe el funcionamiento del circuito. Esto se puede hacer tanto antes como después del proceso de síntesis.

Capítulo 3. ESTADO DE LA CUESTIÓN

Las técnicas de muestro compresivo (Compressive Sampling, CS) existen desde hace más de una década, pero no ha sido hasta los últimos años que se ha comenzado a aplicar a distintos campos [2]. Esta técnica consiste en muestrear señales a frecuencias de muestreo mucho menores que la frecuencia de Nyquist, explotando propiedades de la propia señal como es su dispersidad bajo algún dominio. Una señal se considera dispersa si la mayor parte de su energía se concentra en unos pocos coeficientes aportando el resto una parte muy baja de la energía total.

Es bastante usual que las imágenes se vuelvan dispersas bajo algún dominio, como la DCT (Discrete Cosine Transform) o la WT (Wavelet Transform), por lo que las señales multimedia es uno de esos campos a las que se puedan aplicar el Compressive Sampling [3]. Por ejemplo, la siguiente imagen:

Figura 1 Imagen "Cameraman" en el dominio espacial

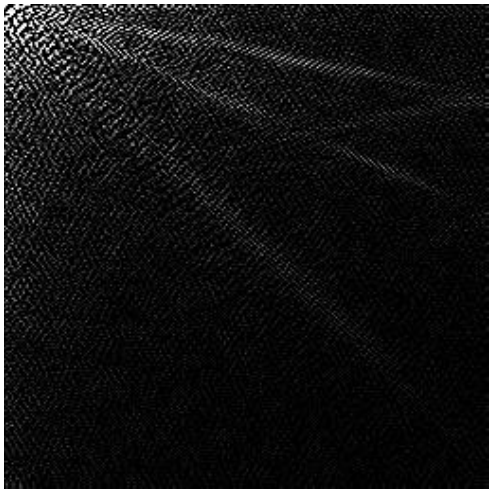

Figura 2 Imagen "Cameraman" en el dominio de la DCT

Como se observa, la imagen tiene valores muy cercanos a 0 (pixeles negros) para casi todos los píxeles en el dominio de la DCT, por lo que se puede considerar una señal dispersa.

Otros campos a los que se ha aplicado las técnicas de CS son los hologramas [4], las resonancias magnéticas [5] o las cámaras infrarrojas [6].

Estas técnicas también pueden ser aplicadas en el campo de las comunicaciones de varias formas. Por ejemplo, en [7] utilizan el CS para estimar el filtro de un canal inalámbrico para posteriormente realizar cancelación. En [8] utilizan el CS para diseñar un sistema de comunicaciones de para UWB (Ultra Wide Band). En este último trabajo, se centran en diseñar un sistema que funcione sin preocuparse por la capacidad conseguida. A partir del concepto planteado se tratará de construir un sistema que use de forma más eficiente el ancho de banda disponible.

Una comunicación se considera UWB si utiliza un ancho de banda de más de 500 Mhz, y presentan algunas propiedades interesantes como resistencia a fading, resistencia a señales de banda estrecha o buena resolución para geolocalización [9]. Pero también presentan como

problema que es necesario tener un muestreador muy rápido para implementar el receptor, por lo que resultan útiles las técnicas CS.

Existen distintos modos de resolver el problema de CS. Una forma es resolver el problema de minimización de norma 2 (l_2), suponiendo que una baja l_0 requiere una baja norma l_2. Esto se puede hacer con el método de mínimos cuadrados (Least Squares, LS), que tiene una baja complejidad pero no da muy buenos resultados.

Otra forma es resolver el problema de mínima l_1 , que se resuelve mediante SIMPLEX cuya complejidad es alta.

Pero la mejor forma de resolver el problema de CS es minimizando la l_0. Este es el método que mejores resultados da, pero al mismo tiempo el que más complejidad tiene. Una posible solución es el algoritmo Matched Pursuit (MP) [10] o su extensión Orthogonal Matched Pursuit (OMP) [11].

La implementación de OMP puede ser bastante compleja ya que requiere cómputos de pseudoinversas, por lo que es difícil usarlo en aplicaciones de tiempo real y con equipos de bajo coste. En trabajos como [12] consiguen lograr una implementación en FPGA.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Uno de los dispositivos más costosos de un sistema de comunicación es el muestreador que se usa en emisor/receptor. La complejidad de diseño del muestreador así como su consumo energético aumenta con la frecuencia de muestreo, por lo que este elemento es crítico cuando una tasa alta es necesaria.

Por el criterio de Nyquist, sabemos que la frecuencia de muestreo es proporcional al ancho de banda de una señal. El ancho de banda es un parámetro crítico que afecta a la comunicación ya que influye de manera proporcional en la capacidad. Teniendo en cuenta el límite de Shanon:

$$C = B * \log_2(1 + SNR)$$

Donde C es la capacidad del enlace, B es el ancho de banda y SNR es el cociente entre la potencia de la señal y la potencia de ruido. El hecho de que el SNR aparezca dentro del logaritmo provoca que sea necesario un aumento importante en el SNR para lograr una ganancia pequeña en la capacidad. Más aún, si partiendo de un enlace de capacidad C, ancho de banda B y con un cociente de potencia entre señal y ruido SNR lo modificamos para, manteniendo la capacidad, usar un mayor ancho de banda $B' = K\ x\ B$, el cociente necesario para mantener la capacidad C será:

$$SNR' = \sqrt[K]{1 + SNR} - 1$$

Por lo que provocaría una ganancia notable en la potencia necesaria en la transmisión.

Por otro lado, el hecho de aumentar el ancho de banda usado provocaría un aumento en la frecuencia de muestreo (por el criterio de Nyquist, F_s debe ser mayor que 2 veces la máxima

frecuencia), causando un aumento de potencia consumida y anulando la ganancia en la transmisión.

Las técnicas de muestreo compresivo permiten recuperar una señal que ha sido muestreado por debajo del límite de Nyquist siempre que se cumplan ciertas características en la señal muestreada. Suponiendo que sea una señal dispersa (la mayor parte de su energía se concentra en unos pocos coeficientes) puede muestrearse con hasta un 1% de la frecuencia de la frecuencia de Nyquist pudiendo así aprovecharse del hecho de aumentar el ancho de banda en la transmisión.

4.2 OBJETIVOS

El objetivo del proyecto es aplicar las técnicas del estado del arte en el campo del muestreo compresivo al diseño de un sistema de comunicaciones de alto ancho de banda usando un muestreador de baja frecuencia. Esto se conseguirá mediante:

- Modelado del problema de CS aplicado a un sistema de comunicaciones.
- Optimización de la solución obtenida para distintos casos de estudio.
- Simulación del sistema diseñado bajo distintas condiciones del canal de comunicaciones (distintos niveles de ruido).
- Análisis de sensibilidad de los parámetros del sistema.
- Efecto del proceso de conversión a digital en el receptor para distinto números de bits.
- Implementación del sistema en una FPGA.

4.3 METODOLOGÍA

Se comenzó el trabajo mediante el estudio de las técnicas del estado del arte para el muestreo compresivo. Después se estudió el modelado del problema haciendo especial énfasis en el

análisis de las condiciones en las que la solución puede simplificarse, ya que de la solución inicial es demasiado costosa de implementar.

Una vez comprendido el modelado, se realizaron simulaciones y análisis de sensibilidad en lenguaje Matlab. También se simuló el efecto de la implementación en punto fijo de las operaciones.

Finalmente se implementó el sistema diseñado usando lenguaje Verilog para una FPGA. También se simuló el circuito para analizar su funcionamiento correcto.

El siguiente gráfico muestra el diagrama de Gantt que ilustra la planificación seguida con el proyecto:

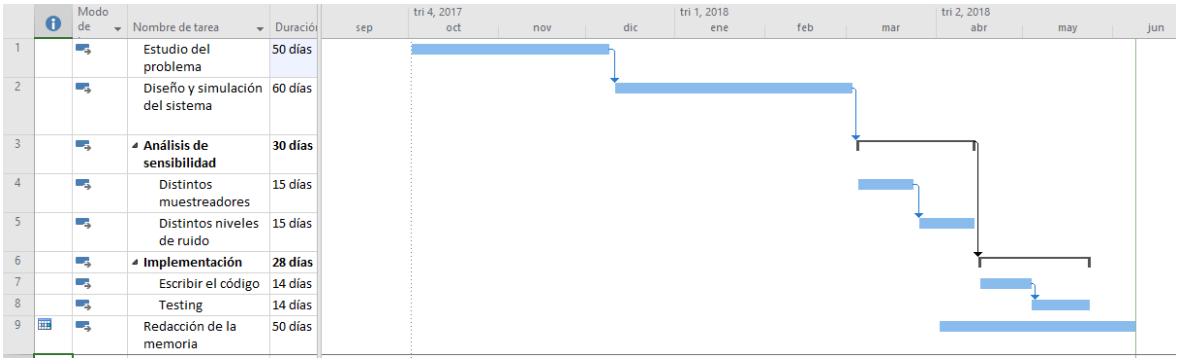

Figura 3 Diagrama de Gantt del proyecto

El proyecto comenzó con un estudio de las técnicas del estudio del arte, trabajos relacionados y las herramientas matemáticas necesarias para trabajar en el campo del muestreo compresivo. Una vez se comprendieron todos los conceptos, se pasó a trabajar en el diseño del sistema, el cual se puso a prueba mediante simulaciones hasta comprobar que el funcionamiento era el deseado.

Una vez decidida la estructura del sistema, se procedió a realizar un análisis de sensibilidad para los distintos parámetros del sistema con el fin de entender como afectaba cada uno de ellos. Esto se hizo para las frecuencias de muestreo y para los niveles de ruido en el canal.

La última etapa del proyecto consistió en la implementación, que incluye tanto la descripción del circuito digital como las simulaciones y testing.

En paralelo con estas tareas, durante los dos últimos meses del proyecto se trabajó en la redacción de esta memoria.

4.4 ESTIMACIÓN ECONÓMICA DE LA SOLUCIÓN

La solución propuesta se simuló para una FPGA, concretamente la XC6SLX9-3TQG144. En el momento de redacción de esta memoria su coste en el mercado es de cerca de 20€, siendo este por tanto el coste de la solución.

Capítulo 5. MODELO DESARROLLADO

En este apartado se detallará el trabajo realizado a lo largo de todas las fases.

5.1 MODELADO DEL PROBLEMA

5.1.1 TEORÍA DE MUESTREO

A partir de una señal $\overline{x} \in \mathbb{R}^N$, se construye una señal $\overline{y} \in \mathbb{R}^M$ con $M < N$ mediante la operación:

$$\bar{y} = A * \overline{x}$$

Donde A es una matriz $A \in \mathbb{R}^{M\ x\ N}$.

Para recuperar la señal x a partir de y, el proceso consistiría en resolver el sistema de ecuaciones $\overline{y} = A * \overline{x}$ siendo A y $\overline{y}$ conocidos. Pero al tener este sistema más incógnitas que ecuaciones (dado que $M < N$) este sistema tendrá infinitas soluciones distintas de las cuales no podremos diferenciar la señal $\overline{x}$ del resto.

Para poder diferenciar la señal original del resto, se introduce una función $f(x)$ definida para cualquier posible señal la cual se busca minimizar. Por tanto, el problema de reconstrucción de la señal $\hat{x}$ se puede modelar como:

$$\overline{\hat{x}} = \min_{\overline{\hat{x}}} f(\overline{\hat{x}}) \mid y = A * \overline{\hat{x}}$$

La función $f(\overline{x})$ explota alguna característica conocida de la señal $\overline{\hat{x}}$. Por ejemplo, podemos definir $f(x)$ como:

$$f(x) = \max_i(\overline{X}[i] \neq 0)$$

Donde $\overline{X}[i]$ es el elemento i del espectro de $\overline{x}$. Por tanto, si elegimos esta función, el problema consistirá en elegir la señal $\overline{x}$ más limitada en banda. Se puede demostrar que solamente una señal tendrá el armónico no nulo más alto en la posición menor que $M/2$, por lo que si conocemos que la señal $\overline{x}$ siempre cumplirá esta condición podremos discriminarla del resto de soluciones del sistema. Esta condición es el "Criterio de Nyquist".

Otras funciones $f(x)$ están basadas en la norma de la señal. Por ejemplo, podemos definirla en base a la norma 2.

$$f(x) = \|\overline{x}\|^2 = \sum_i x_i^2$$

Este problema se conoce como "Problema de mínima norma", y se puede demostrar que su solución se puede determinar por:

$$\hat{\overline{x}} = A^+ * \overline{y}$$

Donde $A^+ \in \mathbb{R}^{N\,x\,M}$ es la matriz pseudoinversa de A. Cabe destacar que la función usada es continua y derivable, razón por la cual es posible conseguir una expresión sencilla para resolver el problema.

Del mismo modo podemos definir la función $f(x)$ como la norma 1 de $\overline{x}$:

$$f(\overline{x}) = \|\overline{x}\|^1 = \sum_i |x_i|$$

En este caso, la función es continua pero no derivable. El problema se puede resolver mediante un algoritmo iterativo como SIMPLEX, el cual supone mucha complejidad en el número de operaciones.

Finalmente, otra posible función es la de la norma 0.

$$f(\overline{x}) = \|\overline{x}\|^0 = \sum_i x_i^0$$

Recordemos que $x_i^0 = 1$ si $x_i \neq 0$ y $x_i^0 = 0$ si $x_i = 0$. Por lo tanto, esta función cuenta la cantidad de ceros presente en el vector $\overline{x}$.

En este caso $f(\overline{x})$ no es continua ni derivable. El problema puede resolverse mediante SIMPLEX añadiendo una gran cantidad de variables binarias, quedando un método de resolución muy costoso. El algoritmo recursivo OMP lo resuelve de forma más eficiente.

Muchas veces, la señal $\overline{x}$ tiene una gran cantidad de ceros en algún dominio (por ejemplo, señales multimedia en el dominio del coseno discreto o de alguna transformada wavelet), como se ha explicado en el apartado de estado del arte. Este problema es conocido como Muestreo Compresivo (*Compressive Sampling*).

5.1.2 ALGORITMO OMP

El algoritmo OMP (*Orthogonal Matched Pursuit*) es un método recursivo para la obtención de la solución de un sistema de ecuaciones lineales que concentra la mayor energía en unos pocos coeficientes (conocida como "solución más dispersa").

Denotamos a_i la columna número i en A y A_I es el conjunto de las columnas de A cuyo índice aparece en el conjunto I. El conjunto $\tilde{I}$ contiene todos los elementos que no contiene el conjunto I.

El algoritmo comienza suponiendo que la solución $\hat{x}$ es un vector de ceros e incrementa en una unidad el número de posiciones no nulas. Los pasos en cada iteración son:

1. Se buscan las proyecciones de la matriz $A_{\tilde{I}}$ sobre $\overline{y}$.

$$\overline{c} = A_{\tilde{I}}^T * \overline{y}$$

2. Se busca el mejor candidato para ser un valor no nulo. Esto se hace mediante:

$$i = \max_i c_i^2$$

3. Se incorpora el índice al conjunto I:

$$I = \{i, I\}$$

4. Se dan dos casos, dependiendo de si el número de valores no nulos K es conocido o no:
    a. Si es conocido: si $|I| = K$ acaba el algoritmo ($|I|$ significa el tamaño del conjunto I), en otro caso se vuelve al paso 1.
    b. Si no es conocido: se calcula el vector estimado en cada momento como:

- $\hat{x} = A_I^+ * y$

Y se comprueba cual es el error en esta estimación como:

$$error = \left\| A * \overline{\hat{x}} - \overline{y} \right\|^2 = \|A * A_I^+ * \overline{y} - \overline{y}\|^2$$

Si $error < \epsilon$, para algún ϵ definido a priori, se para el algoritmo. En otro caso, se vuelve al paso 1.

Al finalizar, se puede calcular la estimación de x como:

$$E.\ 1\ \overline{\hat{x}} = A_I^+ * \overline{y}$$

5.1.3 MATRIZ DE MUESTREO EN CS

La matriz A debe cumplir condiciones para ser adecuada para realizar muestro. Por ejemplo, una matriz que contenga N columnas pero que, por ejemplo, $a_1 = a_2$ no será adecuada, ya que los vectores $v_1 = (1,0,0,\dots,0)$ y $v_2 = (0,1,0,\dots,0)$ producirán el mismo resultado y tienen el mismo número de elementos no nulos, por lo que no podremos diferenciar uno del otro.

Del mismo modo, si se cumple que $a_1 = a_2 + a_3$, el vector $(1,0,0,\dots,0)$ y $(0,1,1,\dots,0)$ tendrán el mismo resultado al ser multiplicado por la matriz. Dado que el v_1 solo tiene una posición no nula, este será el seleccionado por OMP.

El $spark$ de una matriz A es el máximo número tal que, si cogemos cualquier conjunto de $spark(A)$ columnas de A, podemos asegurar que estas serán linealmente independientes. Cualquier vector con K elementos no nulos siendo $K < \frac{spark(A)}{2}$ podrá recuperarse. [13] Si $K \geq M$, el vector no podrá recuperarse en ningún caso y en el intervalo $\frac{spark(A)}{2} < K < M$ podrá recuperarse o no en función del vector.

5.1.4 SEÑALES DISPERSAS POR BLOQUES

En algunas aplicaciones, no solo se cumple que la señal es dispersa sino que los elementos no nulos tienden a aparecer en bloques. Este hecho se puede explotar, ya que al permitir 'descartar' soluciones se puede muestrear a un ratio mucho más bajo que si no tenemos en cuenta este hecho.

La solución de este problema se puede hacer con el algoritmo BOMP (*Block Orthogonal Matched Pursuit*), muy similar a OMP. Consideramos B bloques distintos, y D_b, $b \in \{1, 2, \dots, B\}$ el conjunto de los índices de las columnas correspondientes al bloque número b. El siguiente ejemplo muestra la notación usada en una matriz A de 9 columnas divididas en 3 bloques de 3 columnas cada una (por lo que $B = \frac{9}{3} = 3$).

$$A = \begin{pmatrix} a_{11} & a_{21} & a_{31} & a_{41} & a_{51} & a_{61} & a_{71} & a_{81} & a_{91} \\ a_{12} & a_{22} & a_{32} & a_{42} & a_{52} & a_{62} & a_{72} & a_{82} & a_{92} \\ \underbrace{a_{13}}_{bloque\ 1} & \underbrace{a_{23}}_{bloque\ 1} & \underbrace{a_{33}}_{bloque\ 1} & \underbrace{a_{43}}_{bloque\ 2} & \underbrace{a_{53}}_{bloque\ 2} & \underbrace{a_{63}}_{bloque\ 2} & \underbrace{a_{73}}_{bloque\ 3} & \underbrace{a_{83}}_{bloque\ 3} & \underbrace{a_{93}}_{bloque\ 3} \end{pmatrix}$$

$$D_1 = \{1, 2, 3\}$$

$$D_2 = \{4, 5, 6\}$$

$$D_3 = \{7, 8, 9\}$$

Los pasos del algoritmo BOMP son los siguientes:

1. Se buscan las proyecciones de la matriz A sobre y.
$$\overline{c} = A_I^T * \overline{y}$$
2. Se calcula la aportación de cada uno de los bloques a y.
$$d_b = \sum_{n \in D_b} c_n^2 , \quad b \in \{1, 2, \dots, B\}$$

Mediante simulación, se comprobó que la expresión puede sustituirse por esta otra:

$$d_b = \sum_{n \in D_b} |c_n| , \quad b \in \{1, 2, \dots, B\}$$

Ya que solamente queremos saber cuáles son las columnas que más correlación tienen con la señal. Esta expresión puede ser más fácil de implementar, ya que no requiere una multiplicación.

3. Se busca el mejor candidato para ser un valor no nulo. Esto se hace mediante:
$$i = \max_i d_i$$
4. Se incorpora los índices al conjunto I:
$$I = \{D_i, I\}$$
5. Se dan dos casos, dependiendo de si el número de valores no nulos K es conocido o no:
    a. Si es conocido: si $|I| = K$ acaba el algoritmo, en otro caso se vuelve al paso 1.
    b. Si no es conocido: se calcula el vector estimado en cada momento como:
$$\overline{\hat{x}} = A_I^+ * \overline{y}$$

Y se comprueba cual es el error en esta estimación como:

$$error = \left\| A * \overline{\hat{x}} - \overline{y} \right\|^2 = \|A * A_I^+ * \overline{y} - \overline{y}\|^2$$

Si $error < \epsilon$, para algún ϵ definido a priori, se para el algoritmo. En otro caso, se para al paso 1.

Al igual que en OMP, la estimación de x se calculará como:

$$\overline{\hat{x}} = A_I^+ * \overline{y}$$

En cuanto a las condiciones de la matriz para ser apta para el muestreado, estos se relajan notablemente. Supongamos que dos bloques distintos tienen una columna en común. Si el resto de las columnas son todavía suficientemente diferentes, el algoritmo puede ser capaz de conseguir detectar que bloque es nulo y cual no lo es, cosa imposible en el caso de utilizar OMP. En cualquier caso, a la hora de estimar los valores seguiría siendo un problema ya que estaríamos recogiendo una muestra dos veces, aportando poca información.

5.1.5 SEÑALES DISPERSAS POR BLOQUES CON VALORES NO NEGATIVOS

En el caso de que todos los valores no nulos del vector sean mayores que cero, no solo el ratio de muestreo aumenta, sino que también lo hace la sensibilidad al ruido y la implementación se vuelve mucho más sencilla.

El algoritmo para resolver el problema es OPBOMP (Only Positive BOMP) El algoritmo es exactamente igual que BOMP, salvo en el paso 2 en el cual la expresión usada es:

$$E.\ 2\ d_b = \sum_{n \in D_b} c_n$$

Al realizar la suma con signo, si en algún bloque alguno de los elementos esta negativamente correlado con la señal $\overline{y}$, esto provocará un decremento en el elemento d_b correspondiente y por lo tanto no se tendrá en cuenta para seleccionar el máximo. Por ejemplo, pongamos que el vector de correlaciones es:

$$c = \{0.5, 0.5, 0.5, 1, -1, 1\}$$

$$D_1 = \{1, 2, 3\}$$

$$D_2 = \{4, 5, 6\}$$

Se ve claramente que la correlación con el segundo bloque es mayor que con el primero. Pero este bloque tiene una correlación negativa que vendría provocado por un coeficiente

negativo, lo cual es imposible. Por lo tanto, el bloque seleccionado debería ser el primero. Esto es exactamente lo que pasará al realizar la suma con signo, ya que:

$$d_1 = 0.5 + 0.5 + 0.5 = 1.5$$

$$d_2 = 1 - 1 + 1 = 1$$

Y el primer bloque será el seleccionado.

Las consecuencias de esta pequeña variación son muy grandes. Supongamos que la señal $\overline{y}$ esté corrupta por ruido, provocando que el vector de proyecciones calculado en el paso 1 del algoritmo en realidad sea:

$$c' = c + n$$

Donde n es un vector de ruido con potencia por muestra σ. En el caso de BOMP original, cada muestra del vector d también estará corrupta con ruido:

$$d'_b = d_b + n'$$

Donde n' tendrá potencia $|D_b| * \sigma$. Esto es, la potencia del ruido aumentará conforme aumenta la longitud del bloque. En cambio, al realizar la suma con signo:

$$d''_b = d_b + n''$$

Y la potencia de n'' será $\frac{\sigma}{|D_b|}$. Por lo tanto, el efecto del ruido será hasta $|D_b|^2$ menor que en el caso original.

Otra gran ventaja viene en la implementación. El paso 2 podemos escribirlo de forma matricial, usando una matriz con unos y ceros convenientemente colocados. Por ejemplo, si c tiene 4 posiciones y consideramos 2 posibles bloques de 2 valores cada uno, podemos calcular d como:

$$\overline{d} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} * \overline{c}$$

Llamando a esta matriz M, podemos escribir los pasos del 1 y 2 como:

$$\overline{c} = A_{\tilde{I}}^{T} * \overline{y}$$
$$\overline{d} = D * \overline{c} = M * A_{\tilde{I}}^{T} * \overline{y}$$

Si calculamos a priori la matriz $S = M * A^T$, el cálculo de d será únicamente:

$$\overline{d} = S_{\tilde{I}} * \overline{y}$$

Recordemos que $\overline{d} \in \mathbb{R}^D$ (D es el número de bloques considerado), mucho más pequeño que $\overline{c} \in \mathbb{R}^N$. El número de operaciones en el cómputo de $\overline{d}$ es muy inferior al que había que realizar en BOMP. Este paso no se podía realizar entonces ya que había una operación no lineal entre $\overline{c}$ y $\overline{d}$.

Las condiciones de la matriz también se relajan notablemente. Si dos de los sub-bloques de la matriz A, digamos A_{D_1} y A_{D_2} cumplen que $A_{D_2} = -A_{D_1}$, ambos bloques serían diferenciables.

5.1.6 CASO PARTICULAR: SOLO UNO DE LOS BLOQUES ES NO NULO

Si sabemos a priori que un máximo de un bloque puede estar activo, las condiciones sobre la matriz se relajan al máximo y podemos lograr una implementación muy eficiente.

La matriz A está dividida en bloques tal que:

$$A = [A_1, A_2, \dots, A_B]$$

Podemos construir la matriz A tal que todos los bloques sean exactamente el mismo, cambiando algunos signos correspondientes a algunas de las columnas. Por ejemplo, en una matriz de 12 columnas y 3 bloques:

$$A = [\overline{a_1}, \overline{a_2}, \overline{a_3}, \overline{a_4}| - \overline{a_1}, \overline{a_2}, -\overline{a_3}, \overline{a_4}|\overline{a_1}, -\overline{a_2}, \overline{a_3}, -\overline{a_4}]$$

De esta forma, la cantidad de columnas iguales entre dos bloques es un máximo de 2, teniendo las otras 2 signo contrario. Esto significa que OP-BOMP debería ser siempre capaz de diferenciar cual de los bloques es el que está activo, ya que al hacer suma con signo los signos negativos repartidos penalizarán la suma total.

Podemos escribir A como:

$$A = \left[A^* * T^{(1)}, A^* * T^{(2)}, \dots, A^* * T^{(B)}\right] = A^* * \left[T^{(1)}, T^{(2)}, \dots, T^{(B)}\right]$$

Siendo $T^{(k)}$ una matriz diagonal con valores 1 o -1 y A^* la matriz base.

Si hacemos que las columnas de A^* sean ortonormales entre sí ($A^* * T^{(k1)}$ también lo será) la pseudoinversa puede calcularse como:

$$(A^* * T^{(k)})^+ = (A^* * T^{(k)})^T = \ T^{(k)} * A^{*T}$$

Para realizar la estimación de $\overline{x}$:

$$\overline{\hat{x}} = (A^* * T^{(k)})^+ * \overline{y} = T^{(k)} * A^{*T} * \overline{y}$$

En la práctica, la única operación que sería necesario hacer es $A^{*T} * y$. Después:

$$\overline{\hat{x}} = \left|A^{*T} * \overline{y}\right|$$

Y el bloque en el que están los valores no nulos se puede saber a partir de los signos:

$$T^{(k)} = sign(A^{*T} * \overline{y})$$

Esto solo podrá hacerse en el caso de que no puedan existir solapes entre bloques (es decir, una misma muestra no puede pertenecer a dos bloques diferentes). Si esto ocurre, cada

conjunto de columnas de la matriz seguirá siendo ortonormal así que la pseudoinversa seguirá siendo igual a la transpuesta.

5.1.7 CASO PARTICULAR: MÁS DE UN BLOQUE ES NO NULO

En este caso, no es conveniente que todos los bloques sean el mismo con algunos signos cambiados ya que si $A_m = -A_n$ y ambos bloques son no nulos con los mismos coeficientes la señal se verá anulada. Por lo tanto, lo máximo que podemos hacer es forzar que las columnas de cada bloque A_k sean ortogonales entre sí para cualquier k.

Desgraciadamente, no es posible calcular de forma fácil la pseudoinversa de una matriz $A = [A_1, A_2, A_3, \dots, A_N]$ donde cada uno de los bloques es una matriz unitaria. Si que existe un método para calcular $[A, B]^+$ a partir de A^+, pero requiere el cómputo de dos inversiones de matrices que puede no ser posible en tiempo real.

En la práctica, una forma de implementarlo podría ser tener calculado a priori $[A_{m_1}, A_{m_2}, \dots, A_{m_k}]$ para cualquier combinación $(m_1, m_2, \dots, m_k)$. Si hay un total de b bloques diferentes de los cuales k pueden ser no nulos a la vez, el número de matrices a almacenar será:

$$\frac{b!}{k! * (b-k)!}$$

Esta cantidad crece muy rápido con b y k, pero puede ser algo asumible para números bajos.

Otra forma de hacerlo sin tener que almacenar todas las posibles pseudoinversas es tratar de calcularlas de forma recursiva.

Supongamos que queremos calcular:

$$A_{k+1}^{+} = [A_k|D_k]^{+} = \begin{pmatrix} A_k^T * A_k & A_k^T * D_k \\ D_k^T * A_k & D_k^T * D_k \end{pmatrix}^{-1} * \begin{pmatrix} A_k^T \\ D_k^T \end{pmatrix}$$

Si suponemos que cada uno de los bloques es ortonormal:

$$A_{k+1}^{+} = [A_k|D_k]^{+} = \begin{pmatrix} A_k^T * A_k & A_k^T * D_k \\ D_k^T * A_k & I \end{pmatrix}^{-1} * \begin{pmatrix} A_k^T \\ D_k^T \end{pmatrix}$$

La parte compleja del cálculo es $\begin{pmatrix} A_k^T * A_k & A_k^T * D_k \\ D_k^T * A_k & I \end{pmatrix}^{-1}$. Esta matriz es una matriz simétrica, por lo que su inversa también lo será.

$$\begin{pmatrix} A_k^T * A_k & A_k^T * D_k \\ D_k^T * A_k & I \end{pmatrix} * \begin{pmatrix} X & Y \\ Y^T & Z \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}$$

Por lo que:

$$X * A_k^T * A_k + Y^T * D_k^T * A_k = I$$

$$X * A_k^T * D_k + Y = 0$$

$$Y^T * A_k^T * D_k + Z = I$$

Y podemos despejar las matrices $X, Y\ y\ Z$.

$$X = (A_k^T * A_k - A_k^T * D_k * D_k^T A_k)^{-1}$$

$$Y = -X * A_k^T * D_k$$

$$Z = I - Y^T * A_k^T * D_k$$

5.2 APLICACIÓN A SISTEMA DE COMUNICACIONES

Partiendo de una señal, se le puede realizar una modulación en la forma y fase de una señal dispersa. Por ejemplo, una posible modulación puede ser la mostrada en las siguientes figuras. Los tres primeros bits (en rojo) están modulados en la posición y los tres siguientes (en negro) en la forma de la onda.

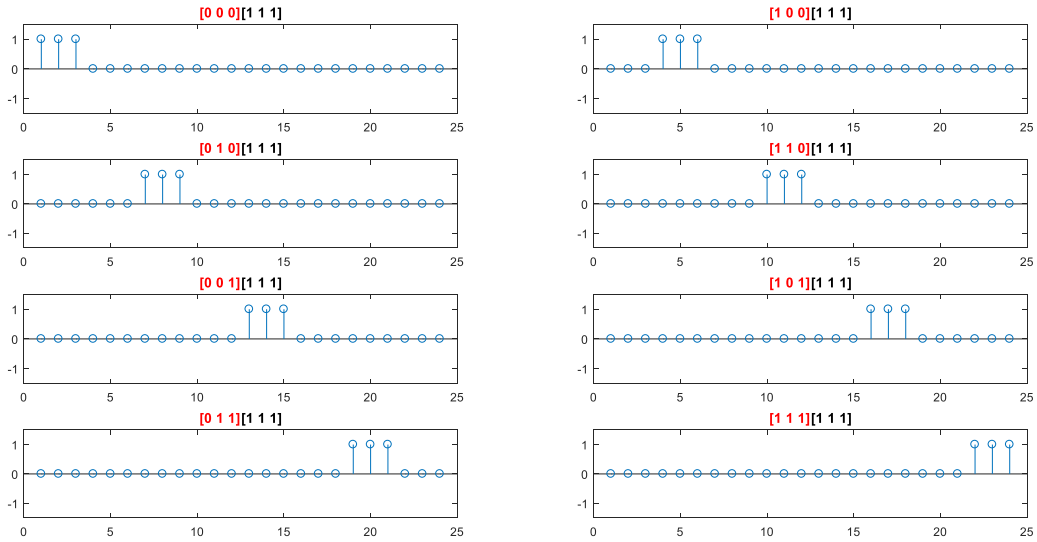

Figura 4 Modulación de bits en la fase de la señal

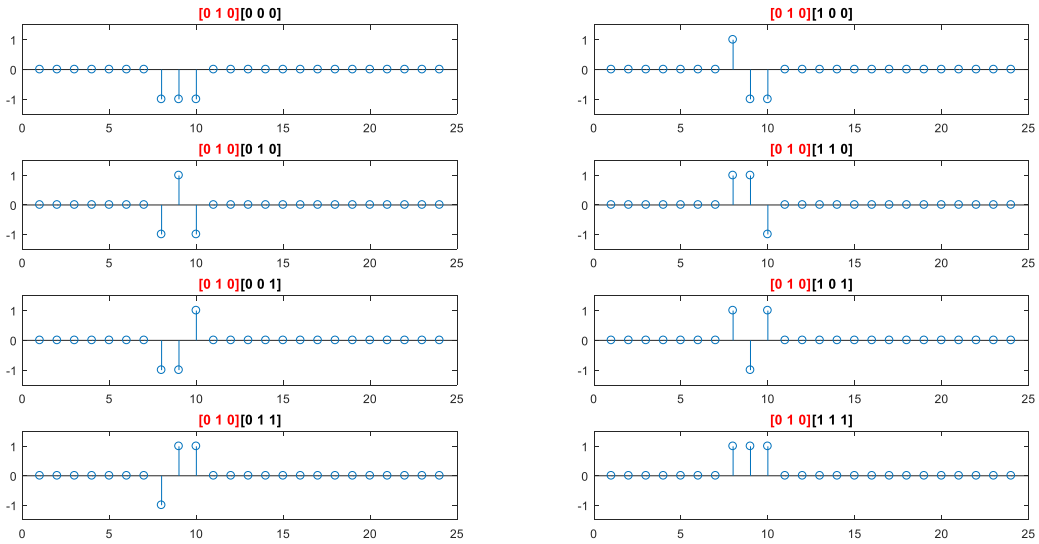

Figura 5 Modulación de bits en la forma de la señal

Si puede haber más de un bloque no nulo a la vez, la información de fase irá en cada combinación de bloques que pueda estar activo.

En un símbolo, el número de bits que se transmite es:

$$bits\ per\ symbol = \log_2 \frac{b!}{k!\,(b-k)!} + L * \log_2 N$$

Donde b es el número de bloques diferentes, k es la cantidad de bloques que puede haber activos al mismo tiempo, L es la longitud del bloque y N el número de niveles de información en cada una de las muestras no nulas.

5.2.1 INTRODUCCIÓN DE OFFSET

El hecho de usar coeficientes todos del mismo signo introduce una pérdida de eficiencia en términos de potencias (es más eficiente modular los 0s y 1s como -A y +A que +A y +2A). Por ello, es conveniente introducir un vector de offset, de forma que en lugar de enviar:

$$\overline{y} = A * \overline{x}$$

Se envíe:

$$\overline{y} = A * (\overline{x} - \overline{o_{ff}})$$

Por lo que el sistema a resolver será:

$$\overline{y} + A * \overline{o_{ff}} = A * \overline{x}$$

El vector $\overline{o_{ff}}$ tendrá que tener todos sus coeficientes iguales. Se puede calcular cual es el vector óptimo de forma que minimice la potencia de $\overline{x} - \overline{o_{ff}}$.

Pongamos que vamos a modular en señales de N puntos con K valores no nulos. Suponemos que modulamos los 0s y 1s con los valores $+1$ y $+2$. Entonces la energía de $\overline{x} - \overline{o_{ff}}$ siendo $\overline{o_{ff}}$ un vector con elementos s será:

$$f(s) = \frac{(1-s)^2 + (2-s)^2}{2} * K + s^2 * (N-K)$$

$$\frac{df}{ds} = (1-s) * K + (2-s) * K + 2 * s * (N-K) = 0$$

$$K - K * s + 2 * K - K * s + 2 * s * N - 2 * s * K = 0$$

$$K - 3 * K * s + 2 * s * N = 0$$

$$s = \frac{K}{3 * K - 2 * N}$$

5.2.2 PRESENCIA DE FILTRO EN EL CANAL

Es frecuente que el canal de comunicaciones pueda modelarse como un filtro, por lo que la señal transmitida será recibida en el receptor deformada. Esto provoca que el proceso de ecualización sea necesario para garantizar una buena calidad en la transmisión. En el dominio de la frecuencia, el proceso de ecualización es relativamente fácil siendo esta una

de las principales ventajas de sistemas como OFDM. En este apartado se plantea alguna idea para aplicarlo en el sistema diseñado.

El efecto de un filtro en el dominio del tiempo acepta un modelado en forma matricial. Por ejemplo, suponiendo un filtro h cuyos coeficientes son $[h_0, h_1, h_2]$ que provoca que al enviar una señal $x[n]$ de 5 muestras se reciba la señal $y[n]$.

$$y[n] = h_0 * x[n] + h_1 * x[n-1] + h_2 * x[n-2]$$

Esta expresión también puede escribirse como:

$$\bar{y} = \begin{pmatrix} h_0 & 0 & 0 & 0 & 0 \\ h_1 & h_0 & 0 & 0 & 0 \\ h_2 & h_1 & h_0 & 0 & 0 \\ 0 & h_2 & h_1 & h_0 & 0 \\ 0 & 0 & h_2 & h_1 & h_0 \end{pmatrix} \bar{x}$$

Por lo tanto, si transmitimos la señal dispersa $\bar{x}$ modulada con la matriz A obteniendo de esta forma la señal transmitida por el canal y se recibe la señal $\bar{y}$, esta señal puede escribirse como:

$$\bar{y} = \underbrace{\begin{pmatrix} h_0 & 0 & 0 & 0 & 0 \\ h_1 & h_0 & 0 & 0 & 0 \\ h_2 & h_1 & h_0 & 0 & 0 \\ 0 & h_2 & h_1 & h_0 & 0 \\ 0 & 0 & h_2 & h_1 & h_0 \end{pmatrix}}_{A'} * A * \bar{x} = A' * \bar{x}$$

Por lo que puede tratar de resolverse el problema simplemente usando una matriz distinta.

5.3 *DESCRIPCIÓN DEL SISTEMA*

Como cualquier sistema de comunicaciones, se compondrá de un emisor y un receptor.

El emisor recibe los bits que se desean transmitir y los modula como una señal la cual es transmitida. La Figura 6 muestra un esquema de cómo está formado el modulador de los bits.

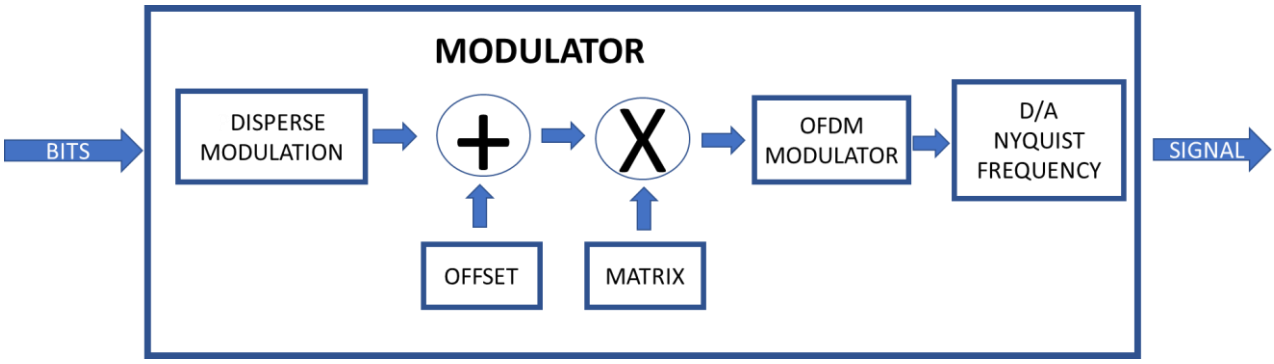

Figura 6 Esquema del modulador

Los bits son modulados en una señal dispersa que asegure que las técnicas de CS sean aplicables. El concepto detrás de esta modulación es el descrito en Figura 4 y Figura 5.

Con el objetivo de aumentar la eficiencia en la potencia de la señal transmitida, se le añade a la señal dispersa un vector de offset tal y como se explicó en el desarrollo en la sección 5.2.1.

La señal resultante es multiplicada por una matriz con las características descritas en la sección 5.1.6. Esta matriz puede ser generada aleatoriamente a partir de una semilla, y los cálculos para obtenerla son desarrollados en posteriores secciones.

Los elementos conseguidos se utilizan como coeficientes de una modulación OFDM para la banda seleccionada. En la práctica, la modulación OFDM puede realizarse de forma matricial como un producto por la matriz de la DFT, por lo que la operación de multiplicación por la matriz de muestreo y la modulación OFDM pueden realizarse en un solo paso.

Finalmente, la señal es convertida a analógico usando un muestreador cuya frecuencia de muestreo es aquella indicada por el criterio de Nyquist.

Del mismo modo, el receptor demodula los bits a partir de la señal recibida. Su esquema se muestra en la Figura 7.

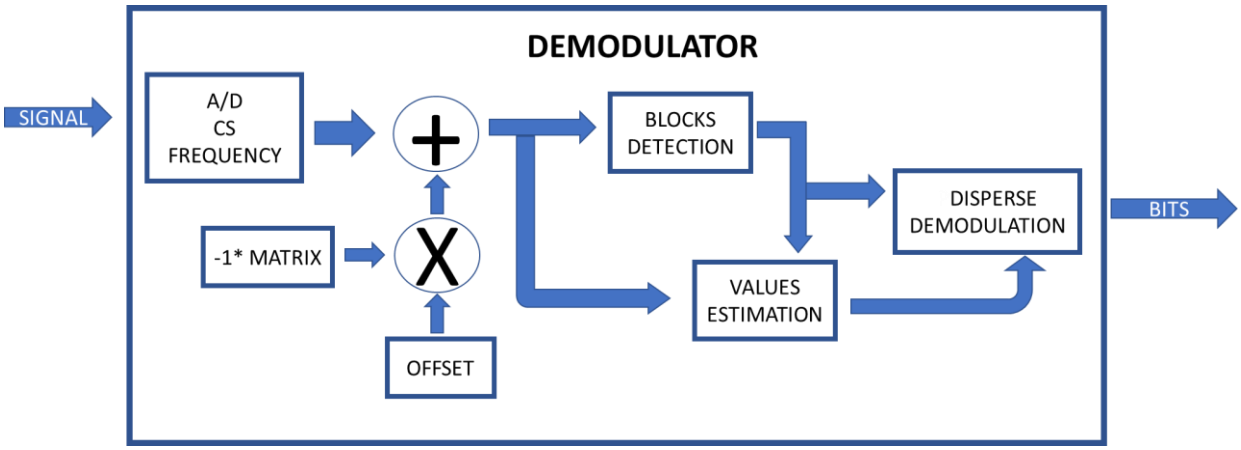

Figura 7 Esquema del demodulador

Las muestras discretas se consiguen a partir de la señal continua con un conversor A/D con una frecuencia de muestreo muy inferior a la necesaria por el criterio de Nyquist, gracias a que sobre ella se aplicarán las técnicas de CS.

Antes de comenzar la resolución del sistema de ecuaciones, se debe eliminar el efecto del offset añadido en el transmisor. Para ello, solamente hay que sumar $-A * \overline{o_{ff}}$ a las muestras.

Después se aplica el algoritmo OPBOMP para localizar la localización de los valores no nulos y posteriormente estimarlos. Como se explicó anteriormente, la localización de las muestras es necesaria para hacer la estimación.

Finalmente, se deshace la modulación dispersa a partir de las muestras obtenidas y su localización y se obtienen los bits transmitidos.

5.3.1 PARÁMETROS

La siguiente tabla muestran los parámetros que caracterizan el sistema de comunicaciones:

Parámetro	*Descripción*
Tiempo de símbolo ($Tsym$)	La longitud de cada uno de los símbolos
Frecuencia inicial (F_{ini})	La primera frecuencia que se usará para la transmisión
Ancho de banda (B)	La banda de frecuencias que se usará para transmitir.
Frecuencia de muestreo del receptor ($F_{s_{CS}}$)	Es la frecuencia de muestreo inferior a la necesaria por Nyquist
Número de bloques activos (N_b)	Número de bloques que pueden tomar valores no nulos al mismo tiempo
Número de muestras no nulas (K)	Número de muestras en la señal dispersa que toman valores distintos de 0.

Tabla 1 Parámetros principales del sistema

A partir de estos parámetros, es posible calcular otros derivados que condicionan algunas características de la transmisión.

Parámetro	*Descripción*	*Cálculo*
Frecuencia final (F_{end})	Última frecuencia usada para la transmisión	$F_{end} = F_{ini} + B$
Distancia entre portadoras (Δf)	Distancia entre una portadora y la siguiente	$\Delta f = \frac{1}{T_{sym}}$
Número de portadoras (N_p)	Número de portadoras que se usa en la transmisión	$N_p = floor\left(\frac{B}{\Delta f}\right)$
Número de muestras de la FFT ($NFFT$)	Número de muestras usado en el cálculo de la FFT/IFFT	$NFFT = 2 * floor\left(\frac{F_{end}}{\Delta f}\right)$
Frecuencia de muestreo de Nyquist ($F_{s_{Ny}}$)	Frecuencia de muestreo en el transmisor según el criterio de Nyquist	$F_{s_{Ny}} = 2 * F_{end}$

Tabla 2 Parámetros derivados del sistema

5.3.2 CREACIÓN DE LOS BLOQUES

Esta sección trata de como seleccionar que portadoras pertenecen a cada uno de los bloques. En código Matlab, esto puede hacerse como:

```
function D=createBlocks(NumberActiveSamples,NumberActiveBlocks,N)
% CREARBLOQUES Devuelve una matriz donde la fila i contiene las portadoras
asociadas al
% bloque i.
%   D=CREARBLOQUES(NValoresActivos,nBlocksActivos,N) devuelve una matriz
%   donde la fila i contiene las portadoras asociadas al bloque i.
%   NValoresActivos es el número de portadoras activas, nBlocksActivos es el
%   número de bloques que puede haber activos a la vez y N es el número de
```

```
%   portadoras.

%   CREATEBLOCKS Returns the matrix where the row i has the indices of the
%   carriers asociated to the block i.
%   createBlocks(NumberActiveSamples,NumberActiveBlocks,N) Returns the
%   matrix where the row i has the indices of the carriers asociated to the
%   block i. NumberActiveSamples is the total number of non-zero samples,
%   NumberActiveBlocks is the number of not null blocks and N is the length
%   of the signal.

    size_blocks=NumberActiveSamples/NumberActiveBlocks; %Size of the blocks
    jumps=size_blocks; %Difference between one block and the next one

    % NOTE: jumps and size_blocks can be different if there are
    % intersection between different groups.

    numBlocks=ceil((N-size_blocks+1)/jumps); %Number of different blocks

    %Creates the matrix
    D=zeros(numBlocks,size_blocks);
    D(1,:)=1:size_blocks;
    for i=2:size(D,1)
        D(i,:)=D(i-1,:)+jumps;
    end
end
```

5.3.3 CREACIÓN DE LA MATRIZ

Se trata de como crear una matriz que sea adecuada para realizar el proceso de muestreo.

```
function X=createMatrix(N,NFFT,seed,carriers,Fs_Ny,Fs_CS,D)
% CREARMATRIX: crea una matriz para el muestreo de señales dispersas
%   X=crearMatrix(N,NFFT,seed,portadoras,Fs,Fs_CS,D) devuelve la matriz de
%   muestreo. N es el número de muestras de la señal, NFFT es el número de
%   portadoras, seed es la semilla usada para la generación de números
%   aleatorios, portadoras es un vector con las portadoras que se usarán
%   en OFDM,Fs es la frecuencia de muestro según Nyquist, Fs_CS es la
%   frecuencia de muestreo del receptor de CS y D es la matriz con las
%   portadoras de cada uno de los bloques.

%   CREATEMATRIX: creates a matrix for sampling disperse signals
%   X=createMatrix(N,NFFT,seed,carriers,Fs_Ny,Fs_CS,D) returns the sampling
%   matrix. N is the number of samples of the signal, NFFT is the number
%   of samples that enter the FFT, seed is the seed used for the random
%   generator, carriers is the vector of the carriers that are used for
%   data in OFDM, Fs_Ny is the sampling frequency according to Nyquist,
%   Fs_CS is the sampling frequency in the receiver and D is the matrix
```

```
%   with the carriers in each one of the blocks.

    NumberCarriers=length(carriers); %Number of carriers that will be used

    %The matrix will be random gaussian variables for the positions of the
    %carriers that are used and  in the others.
    PREM_MOD_MATRIX=zeros((NFFT-2)/2+2,N);

    rng(seed); %Initiliziates the generator of random numbers

    % Places random complex values to the carriers

PREM_MOD_MATRIX(carriers,:)=(randn(NumberCarriers,N)+randn(NumberCarriers,N)*1i);

    rng('shuffle') %Frees the random generator

    %The matrix must be complex conjugated
    PREM_MOD_MATRIX=orth([PREM_MOD_MATRIX;flip(conj(PREM_MOD_MATRIX(2:end-
1,:)))]);

    % Matrix of the IDFT
    iF=(dftmtx(size(PREM_MOD_MATRIX,1))/sqrt(size(PREM_MOD_MATRIX,1)))^(-1);

    % Puts the two matrices together
    X=iF*PREM_MOD_MATRIX;
    X=real(X); %They are already real, but MATLAB does not know

    %% Orthogonal Blocks
    % Vamos a forzar que despues del diezmado la matriz sea ortogonales para
    % cada bloque
    % We will force that after the down-sampling the matrix is orthogonal
    % for each block

    indices=1:Fs_Ny/Fs_CS:size(X,1); %Indices of the samples that are taken in
the receiver

    % For each block
    for i=1:size(D,1)

        % A is the submatrix of the block i
        A=X(:,D(i,:));

        % M has the rows of the samples that are taken
        M=A(indices,:);

        % V is the matrix that will make A*V orthogonal for those rows
        [~,~,V]=svd(M);

        A=A*V; % Does the transformation
```

```
        X(:,D(i,:))=A; % It is inserted in the matrix
    end
end
```

Para asegurar que los bloques sean ortogonales se utiliza la descomposición espectral de valores singulares. Este método permite factorizar una matriz como:

$$A = U * \Sigma * \mathrm{V}^{\mathrm{T}}$$

Donde U, V son matrices unitarias y Σ es una matriz diagonal. Aunque no tiene importancia para este desarrollo, U es la matriz de autovectores de $A^T * A$, V es la matriz de autovectores de $A * A^T$ y los valores en la diagonal de Σ son las raíces de los autovalores de $A * A^T$ y $A^T * A$ (se puede demostrar que ambas matrices tienen los mismos autovalores no nulos).

$$X = A * V = U * \Sigma$$

Teniendo en cuenta las propiedades de las matrices, es fácil comprobar que X es una matriz ortogonal.

Si tenemos una matriz A formada por bloques $A = [A_1, A_2, A_3, \dots, A_N]$, la matriz que la multiplica por la derecha y convierte los bloques en ortogonales es:

$$V = \begin{pmatrix} V_1 & 0 & 0 & 0 \\ 0 & V_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & V_N \end{pmatrix}$$

Donde V_k es la matriz que hace $A_k * V_k$ ortogonal.

5.3.4 GENERACIÓN DE SÍMBOLO DISPERSO

A partir de unos bits, se trata de generar una señal modulada dispersa del modo que se explicó anteriormente. A continuación se muestra el código en Matlab que hace esta función.

```
function
[symbols,sent_bits,Nbits_position,offset,combinations]=createSymbols(N,nActiveBlo
cks,D,combinations)
%   CREATESYMBOLS creates symbols in a disperse signal
%   createSymbols(N,nBlocksActivos,D) creates the symbols in a vector of N
%   samples, with nActiveBlocks non-null blocks. D shows the samples that
%   belong to each block. It returns too the number of modulated bits,
%   number of modulated bits in the position, the offset vector and the
%   combinations of the number of blocks.

    [numBlocks,size_blocks]=size(D);

    NValoresActivos=size(D,2)*nActiveBlocks; %Number of non-null values

    % The number of bits that are in the position depends on the number of
    % combinations for the blocks
    Nbits_position=floor(log2(nCombinaciones(numBlocks,nActiveBlocks)));

    % The number of bits that are in the shape
    Nbits_shape=NValoresActivos;

    bits_per_symbol=Nbits_position+Nbits_shape;

    % Genera los bits
    sent_bits=round(rand(1,bits_per_symbol));

    % Gets the list of combinations (if it is not already computed)
    if(isempty(combinations))
        combinations=listCombinations(1:numBlocks,nActiveBlocks);
    end

    % The bits in the position say which blocks are activated
    indices_seleccionados=combinations(bi2de(sent_bits(1:Nbits_position))+1,:);
    active_blocks=D(indices_seleccionados,:);
    active_blocks=active_blocks(:);

    % The values are 1 if the bit is 0 and 2 if it is 1
    non_null_values=sent_bits(Nbits_position+1:end)+1;

    symbols=zeros(1,N);
    symbols(active_blocks)=non_null_values; %Disperse vector
```

```
    % Computes the optimum offset
    offset= size_blocks/(3*NValoresActivos-2*N)*ones(1,N);


    symbols=symbols+offset;
end

function res=nCombinaciones(N,k)
    % Returns the number of way of placing N elements in groups of K elements
    % without any repetition and no importance to the order
    a=(N-k+1):N;
    b=1:k;

    res=prod(a)/prod(b);
end

function y=listCombinations(list, K)
    %Gives the combination of the elements in list taken by K elements
    if size(list,1)==1
        list=list';
    end
    N=length(list);


    y=[];
    if K>1
        for i=1:N
            X=listCombinations(list(i+1:end),K-1);
            Y=zeros(size(X,1),K);
            Y(:,2:end)=X;
            Y(:,1)=ones(1,size(Y,1))*list(i);
            y=[y;Y];
        end
    else
        y=list;
    end
end
```

5.3.5 CANAL

Se considera un canal AWGN, por lo que la señal recibida es la transmitida más un vector de ruido blanco gausiano. La calidad de la recepción viene determinada por el SNR (Signal

to Noise Ratio), que se define como la potencia de la señal dividida entre la potencia del ruido del canal. Solamente se considera el ruido añadido en la banda considerada, ya que el resto se puede eliminar en el receptor mediante el filtrado. La siguiente función en código Matlab simula el canal de comunicaciones:

```
function y_cont=channel(x_cont,SNR,Fscont,Fini,Fend)
%   CHANNEL: simulates a communication channel as a AWGN channel
%   channel(x_cont,SNR,Fscont,Fini,Fend) returns the results of adding
%   white gaussian noise to x_cont. Fscont is the sampling frecuency used
%   to simulate the continous signal, Fini is the beggining frequency used
%   and Fend is the last one. SNR is the Signal-to-Noise ratio of the
%   channel in dBs of power.


    power_signal=norm(x_cont)^2; %Power of the transmitted channel
    power_noise=10^(-SNR/10)*power_signal; %Power of the noise that must be added

    % Crease a vector of noise in the desired band
    noise_limited=BandLimitedNoise(length(x_cont),Fscont,Fini,Fend);

    % Changes the power of the vector
    noise_limited=noise_limited/norm(noise_limited)*sqrt(power_noise);

    % Adds the noise to the signal
    y_cont=x_cont+noise_limited';
end

function noise=BandLimitedNoise(N,Fs,Fini,Fend)
% This function returns a vector of N positions with gaussian noise between
% the frecuencies Fini and Fend. Fs is the sampling frecuency of the
% signal.
    f=Fs*(0:N-1)/N; %Vector of frequencies

    % Creates the noise in the frequency domain for the desired band
    noise=zeros(1,(N-2)/2);
    for i=1:length(noise)
        if(f(i+1)>Fini && f(i+1)<Fend)
            noise(i)=randn(1,1)+randn(1,1)*j;
        end
    end

    %Makes the signal complex conjugated
    noise=[0,0,noise,flip(conj(noise))];

    %Converts it to the time domain
    noise=real(ifft(noise));
```

```
end
```

5.3.6 DETECCIÓN DE LOS BLOQUES NO NULOS

Este módulo trata de detectar cuales de los bloques son los que tienen los valores no nulos. La función para ello es la siguiente:

```
function I=DetectionModule(x,D,A,nActiveBlocks)
%   DETECTIONMODULE: returns the indices of the non-null blocks
%   DetectionModule(y,D,A,nActiveBlocks) returns the indices of the
%   non-null blocks from the signal x, where D is the matrix whose rows
%   show which samples contain each block, A is the matrix used for
%   sampling and nActiveBlocks is the number of non-null blocks

    %Gets the matrix which will sum the correlation of the coloumns which
    %belong to the same block
    SumingMatrix=zeros(size(D,1),size(A,2));
    for i=1:size(SumingMatrix,1)
        SumingMatrix(i,D(i,:))=1;
    end

    % Multiplies both matrices to convert it into an unique operation
    S=SumingMatrix*A';

    % Computes the score of each block
    score_of_blocks=S*x;

    % Selects the indices of the blocks which score higher
    I=zeros(1,nActiveBlocks);
    for j=1:nActiveBlocks
        [~,i]=max(score_of_blocks);
        I(j)=[I,i];
        score_of_blocks(i)=-inf;
    end

end
```

En ella se usa la conclusión de la expresión E. 2, que permite juntar los dos productos de matrices en uno solo ahorrando de esta forma operaciones.

5.3.7 ESTIMACIÓN DE LOS VALORES NO NULOS

Una vez conocidas las posiciones de los valores no nulos, estos son calculados usando la pseudoinversa tal y como se indica en la ecuación E. 1.

```
function x=ValuesEstimation(y,A,indices)
%   VALUESESTIMATION: return the non-cero values of the solution of the CS
%   problem.
%   ValuesEstimation(y,A,indices) returns the non-cero values of the
%   solution of the CS problem from the y signal using the matrix A knowing
%   those values are found in the positions in the vector indices

    M=A(:,indices); %Takes the coloumns of the non-zero values

    x=pinv(M)*y; %Computes the aproximation of the non-zero values
end
```

5.3.8 RECEPTOR DE MUESTREO COMPRESIVO

Este receptor utiliza la detección y la estimación para demodular los bits enviados a partir de la señal. La función Matlab que llama a las funciones anteriores es la siguiente:

```
function recived_bits=receiverCS(x,Fs_x,Fs_CS,Fs,A,offset,D,Comb)
%   RECEIVERCS: returns the decoded bits using a Compressive Sampling decoder
%
receiverCS(x_cont,Fscont,Fs_CS,Fs,A,offset,D,numberActiveBlocks,Comb,numberBitsIn
Position)
%   returns the bits decoded with the Compressive Sampling demodulator from
%   the signal x, sampled with frequency Fs_x. Fs_CS is the sampling
%   frequency in the receiver, Fs is the sampling frequency used in the
%   transmitter, A is the matrix used to modulate the sparse signal, offset
%   is the vector that is sum to the disperse signal before being modulated
%   D is the matrix which contains the indices of the samples that belong
%   to each block, Comb is the list of combinations of the blocks
%   that is used to modulate in the phase.

numberActiveBlocks=size(Comb,2); %Number of blocks that can be actived
numberBitsInPosition= floor(log2(size(Comb,1))); %Number of bits modulated in the
phase

indices=1:Fs/Fs_CS:size(A,1); %Indices of the rows of A that corresponde to the
picked samples
```

```
x_CS=changeFs(x,Fs_x,Fs_CS); %Downsamples the vector x to the frequency of the CS
receiver

A=A(indices,:); % Takes the rows of the matrix of the picked samples

x_CS=x_CS-A*offset'; % Undo the effects of the offset

% Selects the indeces of the blocks that are not null
indices_of_blocks=DetectionModule(x_CS,D,normc(A),numberActiveBlocks);
indices_of_blocks=unique(indices_of_blocks); %Sorts the indices

%Gets the indices of the samples that correspond to the non-null samples
indices_of_samples=D(indices_of_blocks,:);
indices_of_samples=indices_of_samples(:);

%Symbols are estimeted
recived_symbols=ValuesEstimation(x_CS,A,indices_of_samples);

% Symbols are rounded to the closest possiblity
round_symbols=round(recived_symbols);
round_symbols(round_symbols<1.5)=1;
round_symbols(round_symbols>1.5)=2;

% Bits from the form are extracted
bits_in_form=round_symbols-1;

% From the indices of the blocks, the bits are extracted using the list of
% combinations
indices_of_blocks=ismember(Comb,indices_of_blocks);
indices_of_blocks=sum(indices_of_blocks,2);
[~,indices_of_blocks]=max(indices_of_blocks);

try
    % Bits are exptracted from the phase
    bits_in_phase=de2bi(indices_of_blocks-1,numberBitsInPosition);
catch
    % There are some combinations of blocks that are not possible, in this
    % case the bits are set to 0
    bits_in_phase=zeros(1,numberBitsInPosition);
end

recived_bits=[bits_in_phase,bits_in_form'];
end
```

Capítulo 6. IMPLEMENTACIÓN DEL SISTEMA

El sistema fue descrito en forma de circuito digital usando lenguaje Verilog. En esta sección se dan los detalles de esta implementación.

Todo el código se realizó de forma parametrizada para el número de muestras, el número de bloques diferentes, el tamaño de los bloques y el número de bits usado para representar cada número. Este primer paso se calculó en Matlab mediante una simulación de implementación de punto fijo para distintos parámetros.

El lenguaje Verilog permite manejar estructuras complejas como matrices de datos, pero estas no pueden ser entradas o salidas de un módulo. Por ello las matrices son convertidas a listas unidireccionales a partir de sus filas de izquierda a derecha. Por ejemplo, la matriz:

$$\begin{pmatrix} 00 & 01 & 10 \\ 11 & 00 & 11 \end{pmatrix}$$

Se escribiría en forma de array como:

$$(000110110011)$$

A partir de un vector como este, puede reconstruirse una matriz de forma que puedan aprovecharse las ventajas de la representación en forma matricial.

La siguiente figura muestra el esquema de la implementación diseñada:

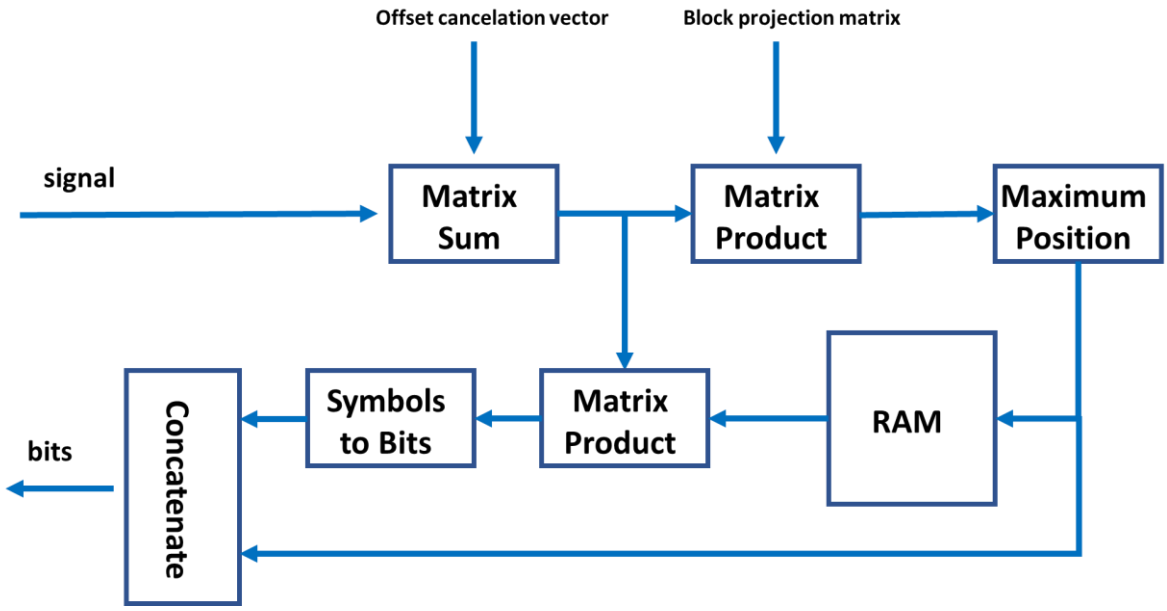

Figura 8 Diseño de la implementación del algoritmo OPBOMP

Existía la posibilidad de plantear un diseño secuencial, que pudiese utilizar el mismo multiplicador de matrices para las dos operaciones y por tanto simplificase notablemente el área y la cantidad de recursos a utilizar. Incluso el producto matricial puede plantearse a partir de un solo producto interior de vectores, con el que se calcula los elementos de la matriz resultado uno a uno. Esto permitiría usar un reloj mucho más rápido, pero el número de ciclos sería mucho mayor.

Frente a esta posibilidad se decidió partir de un diseño plenamente combinacional. Esta opción es la más rápida, ya que puede ejecutarse un gran número de operaciones en paralelo. Además, otra gran ventaja de este diseño es que puede adaptarse para un sistema de pipeline muy fácilmente: se puede hacer simplemente añadiendo registros antes y después de cada módulo. Esto provocaría que el sistema pueda ser usado con tiempos de símbolo más cortos sin que el sistema se vuelva inestable, ya que será suficiente con que el tiempo de símbolo sea mayor que el tiempo del camino combinacional más largo.

De hecho, el diseño planteado es en realidad un pipeline de dos etapas. Esto se debe a que la memoria RAM usada necesita un ciclo de reloj para realizar la lectura (esta señal no se ha representado en el esquema para mayor simplicidad). Ya que es necesario dos ciclos de reloj para terminar la ejecución, se consideró conveniente añadir un registro flip flop (tampoco representado en el esquema) que guarda la señal de entrada durante un ciclo de reloj. Esto permite convertir el diseño en un pipeline, pudiendo entrar una nueva señal cada ciclo de reloj. Además, se añadió otro flip flop a la entrada y otro a la salida, con el fin de tener controlado el momento en el que se puede introducir una nueva entrada o leer una salida.

El esquema anterior se describe en el siguiente código Verilog:

```
1.  module OPBOMP
2.          #(
3.                  parameter SIZE_BLOCKS   =3,
4.                  parameter N             =24,
5.                  parameter DATA_WIDTH    =16,
6.                  parameter NUM_BLOCKS    =32,
7.                  parameter LOG_NUM_BLOCKS =$clog2(NUM_BLOCKS)
8.          )(
9.          input [N*DATA_WIDTH-1:0] x, //received signal
10.         input clk, //clock signal, needed for the RAM
11.         output [SIZE_BLOCKS+LOG_NUM_BLOCKS-1:0] output_bits //decoded bits
12.          );
13.
14.         wire [N*DATA_WIDTH-1:0] x_input,x_middle;
15.         wire [DATA_WIDTH*N*NUM_BLOCKS-1:0] sumByBlocks; //Matrix to do the
    projection
16.         wire [DATA_WIDTH*NUM_BLOCKS-1:0]projections; //Projections on each block
17.         wire [LOG_NUM_BLOCKS-1:0] posmax; //Position of the maximum projection
18.         wire [DATA_WIDTH*N*SIZE_BLOCKS-1:0] selectedPseudoinverse; //Output of the
    RAM
19.         wire  [N*DATA_WIDTH-1:0] offset; //Offset cancelation constant
20.         wire  [N*DATA_WIDTH-1:0] signal_without_offset; //Signal after offset
    removal
21.         wire [DATA_WIDTH*SIZE_BLOCKS-1:0]estimatedvalues; //Estimated coefficients
22.         wire [SIZE_BLOCKS-1:0] bits;
23.
24.         FlipFlop #(
25.                 .N(N*DATA_WIDTH)
26.         ) inst_FlipFlop_input_x (
27.                 .clk    (clk),
28.                 .data   (x),
29.                 .enable (1'b1),
30.                 .reset  (1'b1),
31.                 .clear  (1'b0),
32.                 .out    (x_input)
33.         );
34.
35.                 FlipFlop #(
36.                 .N(N*DATA_WIDTH)
```

```
        ) inst_FlipFlop_middle_x (
                .clk    (clk),
                .data   (signal_without_offset),
                .enable (1'b1),
                .reset  (1'b1),
                .clear  (1'b0),
                .out    (x_middle)
        );



         // Returns the offset constant
        offset #(.N(N*DATA_WIDTH))offsetvector (
    .out(offset)
    );

        // Cancels the offset
        MatrixAdder #(.M(N),.N(1),.DATA_WIDTH(DATA_WIDTH))offset_elimination (
    .a(x_input),
    .b(offset),
    .res(signal_without_offset)
    );

        // Outputs the matrix which does the projection
         MatrixSumByMatrix #(.N(DATA_WIDTH*N*NUM_BLOCKS)) sumbyblockmatrix (
    .out(sumByBlocks)
    );

        //Performs the projection
        MatrixMultiplier #(.M(NUM_BLOCKS),.N(N),.O(1))projectionsCalculation (
    .x1(sumByBlocks),
    .x2(signal_without_offset),
    .y(projections)
    );

         // Outputs the position of the maximum projection
         comparator #(.DATA_WIDTH(DATA_WIDTH),.N(NUM_BLOCKS),.BITS_FOR_POSITION(LOG_
NUM_BLOCKS)) comparation (
    .values(projections),
    .pos_max(posmax)
    );


         // Keeps all the different pseudoinverses
         RAM PseudoinversesInRAM (
          .clka(clk),
          .addra(posmax),
          .douta(selectedPseudoinverse)
        );


        // Estimates the coeficients
        MatrixMultiplier #(.M(SIZE_BLOCKS),.N(N),.O(1))valueestimation (
    .x1(selectedPseudoinverse),
    .x2(x_middle),
    .y(estimatedvalues)
    );

         // Computes the bits from the coeficients
         Symbols2Bits #(.N(SIZE_BLOCKS),.DATA_WIDTH(DATA_WIDTH))s2b (
    .x(estimatedvalues),
    .output_bits(bits)
    );

                FlipFlop #(
```

```
101.                        .N(SIZE_BLOCKS+LOG_NUM_BLOCKS)
102.                ) inst_FlipFlop (
103.                        .clk    (clk),
104.                        .data   ({bits,posmax}),
105.                        .enable (1'b1),
106.                        .reset  (1'b1),
107.                        .clear  (1'b0),
108.                        .out    (output_bits)
109.                );
110.
111.
112.
113.        endmodule
114.
```

6.1 SIMULACIÓN DE LA IMPLEMENTACIÓN DE PUNTO FIJO

El primer paso fue calcular cual es un buen valor para representar los valores en un circuito digital. Para simplificar las operaciones, esta representación se hizo en punto fijo. En esta representación, se guarda un bit para el signo y una cantidad fija de bits para la parte decimal del número. Además, se usa complemento a dos para representar los números negativos simplificando de esta forma las restas.

Esta simulación se hizo en Matlab, usando las herramientas de simulaciones de punto fijo que se proporcionan. Solamente las operaciones asociadas al receptor son consideradas como fuentes de ruido de redondeo.

6.2 OPERACIONES PREDEFINIDAS

Se han usado para la implementación algunas operaciones de módulos ya definidas en el core de la FPGA. Estas operaciones son los multiplicadores, sumadores y memoria RAM usada para guardar cada una de las pseudoinversas.

6.3 *PRODUCTO MATRICIAL*

El producto de dos matrices se puede entender como el producto escalar de las filas de la primera y las columnas de la segunda. Por eso, el bloque de producto matricial se crea como la instanciación de varios productos escalares. El siguiente diagrama muestra el producto matricial de una matriz de 3 filas y un vector vertical.

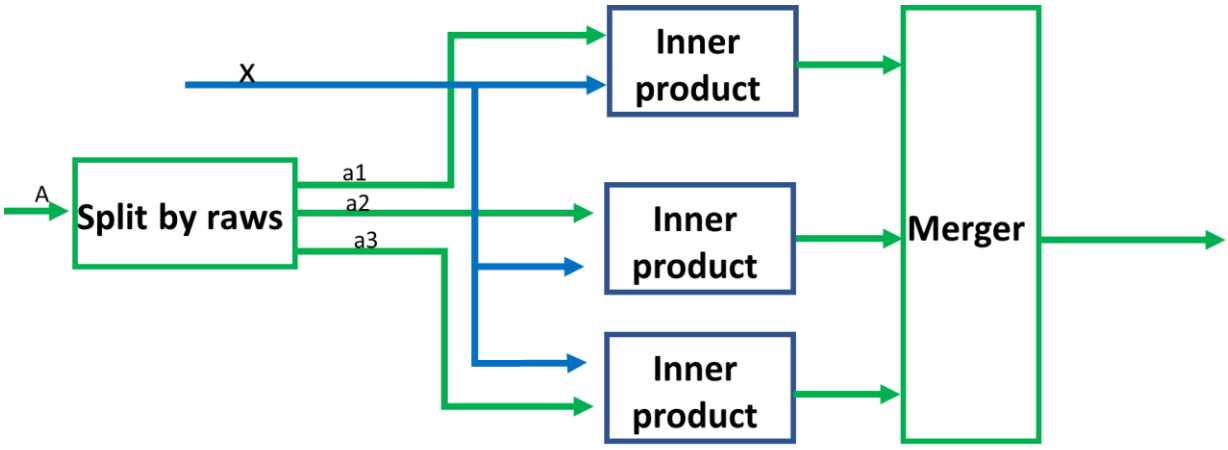

Figura 9 Diagrama de producto de matriz y vector

```
1.  module MatrixMultiplier #(      parameter M=2, parameter N=2,parameter O=1,parameter
     DATA_WIDTH=16)
2.  (
3.          input [M*N*DATA_WIDTH-1:0]x1, // First matrix
4.          input [N*O*DATA_WIDTH-1:0]x2, // Second matrix
5.          output [M*O*DATA_WIDTH-1:0]y // Result
6.  );
7.
8.
9.          wire [N*O*DATA_WIDTH-1:0]X2trans; // Transpose of matrix x2
10.         wire [DATA_WIDTH-1:0]resmat[0:M-1][0:O-1]; // Matrix form of the result
11.
12.         genvar i,j;
13.
14.
15.         // Transposes the matrix x2 in order to more easily select the rows
16.         transpose #(.M(N),.N(O), .DATA_WIDTH(DATA_WIDTH)) trans (
17.                 .a(x2),
18.                 .b(X2trans)
19.                 );
20.
21.         // Each position of the resulting matrix is got as an inner product
```

```
22.         generate
23.                 for (i = 0; i < M; i = i + 1) begin: loop5
24.                         for (j = 0; j < O; j = j + 1) begin: loop6
25.                                 VectorProduct #(.N(N), .DATA_WIDTH(DATA_WIDTH)) inne
    rproduct (
26.                                         .a(x1[M*N*DATA_WIDTH-i*N*DATA_WIDTH-
    1:M*N*DATA_WIDTH-i*N*DATA_WIDTH-N*DATA_WIDTH]),
27.                                         .b(X2trans[O*N*DATA_WIDTH-j*N*DATA_WIDTH-
    1:O*N*DATA_WIDTH-j*N*DATA_WIDTH-N*DATA_WIDTH]),
28.                                         .res(resmat[i][j])
29.                                     );
30.                         end
31.                 end
32.         endgenerate
33.
34.         // Converts the matrix form of the result to a vector representation
35.         generate
36.                 for (i = 0; i < M; i = i + 1) begin: loop7
37.                         for (j = 0; j < O; j = j + 1) begin: loop8
38.                                 assign y[M*O*DATA_WIDTH-i*O*DATA_WIDTH-j*DATA_WIDTH-
    1:M*O*DATA_WIDTH-i*O*DATA_WIDTH-j*DATA_WIDTH-DATA_WIDTH]=resmat[i][j];
39.                         end
40.                 end
41.         endgenerate
42. endmodule
```

6.3.1 TRASPUESTA DE UNA MATRIZ

Este módulo únicamente se utiliza ya que, al utilizar una representación por filas de las matrices, la selección de columnas resulta complicada. Al hacer la transpuesta, las columnas son filas y facilita escribir el código. Es importante destacar que esta operación únicamente es una relocalización de los bits, por lo que después del proceso de sintetizar debería tener un coste nulo.

```
1.  module transpose #( parameter M=2, parameter N=2,   parameter nBits=2)
2.     (
3.          input  [N*M*nBits-1:0]a, //input matrix
4.          output [N*M*nBits-1:0]b //transposed matrix
5.      );
6.
7.
8.
9.      wire [nBits-1:0]Amat[0:M-1][0:N-1]; //Matrix form of the input
10.     wire [nBits-1:0]Bmat[0:N-1][0:M-1]; //Matrix form of the output
11.
12.     genvar i,j;
13.
14.     //Converts the input to a matrix form
15.     generate
16.         for (i = 0; i < M; i = i + 1) begin: loop1
17.             for (j = 0; j < N; j = j + 1) begin: loop2
18.                     assign Amat[i][j]=a[M*N*nBits-N*i*nBits-j*nBits-1:M*N*nBits-
    N*i*nBits-j*nBits-nBits];
```

```
19.             end
20.         end
21.     endgenerate
22.
23.     // Computes the transpose
24.     generate
25.         for (i = 0; i < M; i = i + 1) begin: loop3
26.             for (j = 0; j < N; j = j + 1) begin: loop4
27.                     assign Bmat[j][i]=Amat[i][j];
28.             end
29.         end
30.     endgenerate
31.
32.     // Converts the transpose to a matrix form
33.     generate
34.         for (i = 0; i < N; i = i + 1) begin: loop5
35.             for (j = 0; j < M; j = j + 1) begin: loop6
36.                     assign b[M*N*nBits-M*i*nBits-j*nBits-1:M*N*nBits-M*i*nBits-
    j*nBits-nBits]=Bmat[i][j];
37.             end
38.         end
39.     endgenerate
40.
41.
42. endmodule
```

6.3.2 PRODUCTO ESCALAR DE VECTORES

Esta operación se usa para la implementación del producto matricial, ya que el producto de dos matrices se puede descomponer en el producto escalar de las filas de la primera y las columnas de la segunda. La estructura de este bloque para dos vectores de tamaño 4 es la siguiente:

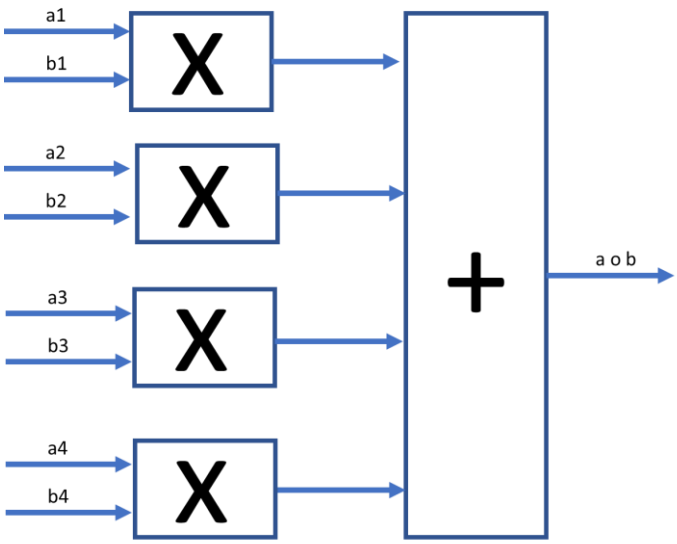

Figura 10 Estructura de un producto escalar de dos vectores

El código Verilog parametrizado que describe esta función es el siguiente.

```
1.  module VectorProduct #(  parameter N=2, parameter DATA_WIDTH=32)
2.  (
3.           input [N*DATA_WIDTH-1:0] a, // First vector
4.           input [N*DATA_WIDTH-1:0] b, // Second vector
5.           output [DATA_WIDTH-1:0]res // Result
6.   );
7.
8.
9.          wire [DATA_WIDTH-1:0]Amat[0:N-1]; // Matrix representation of the vector A
10.         wire [DATA_WIDTH-1:0]Bmat[0:N-1]; // Matrix representation of the vector B
11.         wire [N*DATA_WIDTH-1:0] prods; // To keep the element-by-element products
12.         wire [DATA_WIDTH-1:0]prods_matrix[0:N-1]; // To keep the element-by-element
    products in matrix form
13.
14.         genvar i;
15.
16.         // Converts the vectors to a matrix form
17.         generate
18.                 for (i = 0; i < N; i = i + 1) begin: loop1
19.                                 assign Amat[i]=a[N*DATA_WIDTH-1-
    i*DATA_WIDTH:N*DATA_WIDTH-i*DATA_WIDTH-DATA_WIDTH];
20.                                 assign Bmat[i]=b[N*DATA_WIDTH-1-
    i*DATA_WIDTH:N*DATA_WIDTH-i*DATA_WIDTH-DATA_WIDTH];
21.                 end
22.         endgenerate
23.
24.         // Implements the multiplications
```

```
25.         generate
26.                 for (i = 0; i < N; i = i + 1) begin: loop2
27.                         multiplier mult (
28.                                 .a(Amat[i][DATA_WIDTH-1:0]),
29.                                 .b(Bmat[i][DATA_WIDTH-1:0]),
30.                                 .p(prods_matrix[i][DATA_WIDTH-1:0]));
31.
32.                 end
33.         endgenerate
34.
35.
36.         // Converts the matrix form of multiplications to a vector form
37.         generate
38.                 for (i = 0; i < N; i = i + 1) begin: loop4
39.                         assign prods[N*DATA_WIDTH-1-i*DATA_WIDTH:N*DATA_WIDTH-
    i*DATA_WIDTH-DATA_WIDTH]=prods_matrix[i];
40.                 end
41.         endgenerate
42.
43.
44.         // Sums the elements of the vector
45.         SumElements #(.N(N),.DATA_WIDTH(DATA_WIDTH)) sums (
46.     .a(prods),
47.     .b(res)
48.     );
49.
50.
51. endmodule
```

6.3.3 SUMA DE LOS VALORES DE UN VECTOR

Este módulo suma los valores de un vector. Esta operación se implementa con una estructura en cascada. Por ejemplo, la suma de 8 valores se haría con una estructura como la siguiente.

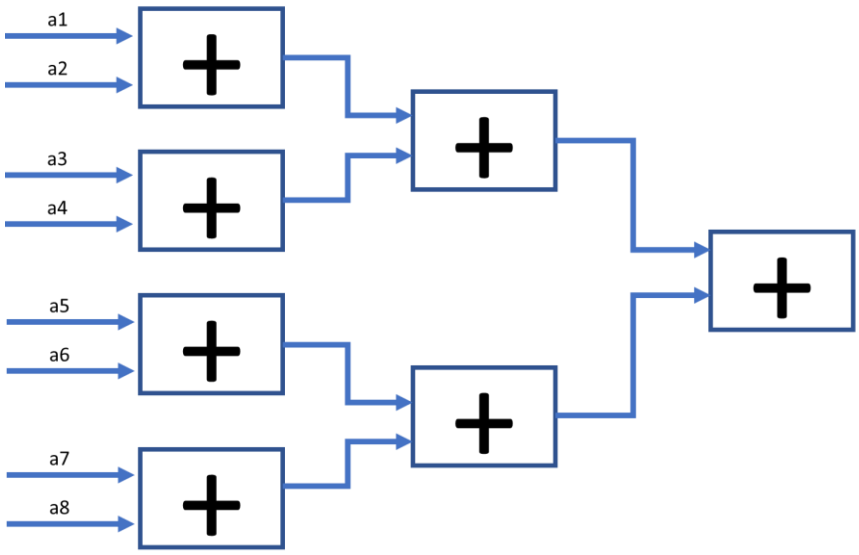

Figura 11 Suma de los valores de un vector en forma de cascada

Esta estructura escala en tiempo con $\log_2 N$ y en espacio de forma lineal, mientras que la suma acumulativa la hace de forma lineal en ambas por lo que es una estructura mucho más rápida.

El código que implementa esta función es el siguiente:

```
1.  module SumElements #(parameter N=2, parameter DATA_WIDTH=16)(
2.          input [N*DATA_WIDTH-1:0] a, //vector
3.          output [DATA_WIDTH-1:0]b // result
4.      );
5.
6.
7.          genvar i,j;
8.          parameter log=$clog2(N)+1; //Number of layers of the structure
9.
10.         wire [DATA_WIDTH-1:0]sums[0:2**log-1][0:log]; //Matrix to keep the partial
    values
11.
12.         //Initializes the first layer of the structure
13.         generate
14.                 for (i = 0; i < 2**log; i = i + 1) begin: loop1
15.                         if(i<N) begin
16.                                 assign sums[i][0] = a[N*DATA_WIDTH-i*DATA_WIDTH-
    1:N*DATA WIDTH-i*DATA WIDTH-DATA WIDTH];
17.                         end else begin
18.                                 assign sums[i][0] = 0;
19.                         end
20.                 end
21.         endgenerate
```

```
22.
23.         //Adds adders between each layer and the next one
24.         generate
25.                 for (i = 1; i <= log; i = i + 1) begin: loop2
26.                         for (j = 0; j <2**log/(2**i) ; j = j + 1) begin: loop3
27.                         adder add (
28.                                 .a(sums[2*j][i-1]),
29.                                 .b(sums[2*j+1][i-1]),
30.                                 .s(sums[j][i])
31.                                 );
32.                         end
33.                 end
34.         endgenerate
35.
36.         // The result can be found in the last layer
37.         assign b=sums[0][log];
38. endmodule
```

En el proceso de síntesis los espacios de la matriz que no se usarán deberían ser eliminados, por lo que el resultado debería ser el mismo que el mostrado en Figura 11.

6.4 SELECCIÓN DEL MÁXIMO ELEMENTO DE UN VECTOR

La búsqueda del índice de la posición del mayor elemento en el vector puede hacerse con una estructura muy similar a la suma de elementos de un vector, utilizando comparadores de dos elementos al mismo tiempo. Para el caso de 8 elementos, la estructura será la siguiente:

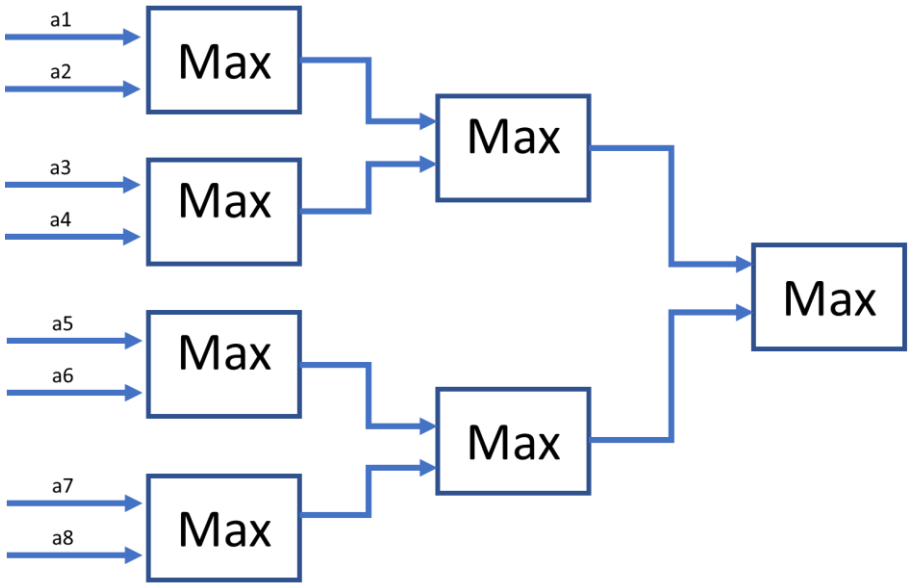

Figura 12 Selección del máximo de los valores de un vector en forma de cascada

Esta estructura se implementa mediante el siguiente código:

```
1.  module comparator
2.  #(
3.          parameter DATA_WIDTH        = 16,
4.          parameter N                 = 8 ,
5.          parameter BITS_FOR_POSITION = 3
6.  ) (
7.          input  [DATA_WIDTH*N-1:0]               values , //Vector of values
8.          output [BITS_FOR_POSITION-1:0]  pos_max  //Position of the maximum value
9.  );
10.
11.         genvar i,j;
12.
13.         wire [DATA_WIDTH-1:0]                   partialmaxs [0:2**BITS_FOR_POSITION-
    1][0:BITS_FOR_POSITION]; //To keep the maximum values
14.         wire [BITS_FOR_POSITION-1:0]    partialpos      [0:2**BITS_FOR_POSITION-
    1][0:BITS_FOR_POSITION]; //To keep the maximum positions
15.
16.         //Initiatilizes the first layer
17.         generate
18.                 for (i = 0; i < 2**BITS_FOR_POSITION; i = i + 1) begin: loop1
19.                         if(i<N) begin
20.                                 assign partialmaxs[i][0] = values[N*DATA_WIDTH-
    i*DATA_WIDTH-1:N*DATA_WIDTH-i*DATA_WIDTH-DATA_WIDTH];
21.                                 assign partialpos[i][0]  = i;
22.                         end else begin
23.                                 assign partialmaxs[i][0] = 15'd0;
24.                                 assign partialpos[i][0]  = 15'd0;
25.                         end
26.                 end
27.         endgenerate
28.
```

```
29.         //Gets the value for each layer from the previous one
30.         generate
31.                 for (i = 1; i <= BITS_FOR_POSITION; i = i + 1) begin: loop2
32.                         for (j = 0; j <2**BITS FOR POSITION/(2**i) ; j = j + 1) begi
    n: loop3
33.                                 comparator2 #(
34.                                         .BITS_FOR_POSITION(BITS_FOR_POSITION),
35.                                         .DATA_WIDTH(DATA_WIDTH)) comp2 (
36.                                         .a          (partialmaxs[2*j][i-1]   ),
37.                                         .b          (partialmaxs[2*j+1][i-1]),
38.                                         .pos_a      (partialpos[2*j][i-1]    ),
39.                                         .pos_b      (partialpos[2*j+1][i-1] ),
40.                                         .value_max(partialmaxs[j][i]        ),
41.                                         .pos_max    (partialpos[j][i]        )
42.                                 );
43.                         end
44.                 end
45.         endgenerate
46.
47.         //Gets the result from the last layer
48.         assign pos_max = partialpos[0][BITS_FOR_POSITION];
49.
50.
51. endmodule
```

6.4.1 SELECCIÓN DEL MÁXIMO DE DOS VALORES

Cada uno de los comparadores devolverá no solo el máximo de ambos sino la posición que ocupa ese valor en el vector. De esta forma, podrá obtenerse en la última capa la posición del máximo valor.

El código en Verilog que hace esta función es el siguiente:

```
1. module comparator2#(parameter nBits=16, parameter bits_for_position
   s=4)(
2.     input signed [nBits-1:0] a,b, //The two values to compare
3.     input [bits_for_positions-1:0] pos_a, pos_b, // The positions
   of each of those two values
4.     output reg [bits_for_positions-1:0] pos_max, //The position of
   the maximum value
5.     output reg [nBits-1:0] value_max); //The maximum value
6.
7.     always @* begin
8.         if(a>b) begin
9.             pos_max=pos_a;
10.                 value_max=a;
11.              end else begin
12.                 pos_max=pos_b;
13.                 value_max=b;
14.             end
```

```
15.           end
16.
17.       endmodule
```

Al sintetizar, esta estructura se implementará mediante un comparador y dos multiplexores, tal y como muestra la siguiente figura.

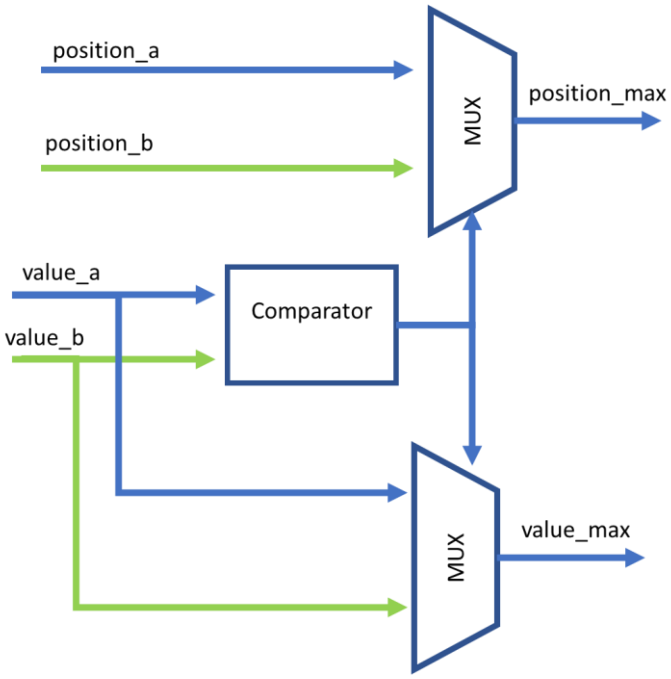

Figura 13 Selección del máximo entre dos valores

6.5 SUMA DE MATRICES

Este módulo se ha usado para realizar la cancelación del offset introducido en el emisor, por lo que funciona como una suma de dos vectores. Consiste únicamente en la suma de los elementos de ambos vectores que comparten índices. El código en Verilog para este módulo es el siguiente:

```
1. module MatrixAdder #(parameter N=2, parameter M=2, parameter nBits=
   16)(
2.     input [N*M*nBits-1:0]a, // First matrix
3.     input [N*M*nBits-1:0]b, // Second matrix
4.     output [N*M*nBits-1:0]res //Result
5.     );
6.
7.
8.     genvar i,j;
9.
10.          generate
11.              for (i = 0; i < N; i = i + 1) begin: loop3
12.                  for (j = 0; j < M; j = j + 1) begin: loop4
13.                      adder addition(
14.                        .a(a[M*N*nBits-i*N*nBits-j*nBits-
   1:M*N*nBits-i*N*nBits-j*nBits-nBits]), // input [31 : 0] a
15.                        .b(b[M*N*nBits-i*N*nBits-j*nBits-
   1:M*N*nBits-i*N*nBits-j*nBits-nBits]), // input [31 : 0] b
16.                        .s(res[M*N*nBits-i*N*nBits-j*nBits-
   1:M*N*nBits-i*N*nBits-j*nBits-nBits]) // output [31 : 0] s
17.                      );
18.                  end
19.              end
20.          endgenerate
21.      endmodule
```

6.6 CONVERSIÓN DE LOS SÍMBOLOS A BITS

A partir de los símbolos se extraen los bits transmitidos mediante la comparación con el valor 1.5 (valor medio entre 1 y 2, los cuales se usan para modular la información). Esto se ha implementado mediante la comparación con el valor 1.5 de los coeficientes estimados.

```
1. module Symbols2Bits
2.     #(
3.         parameter N=1,
4.         parameter DATA_WIDTH=16
5.     )(
6.         input [DATA_WIDTH*N-1:0] x, //Vector of coeficients
7.         output [N-1:0] output_bits //Decoded bits
8.     );
9.
10.
11.          genvar i;
12.
```

```
13.          //Substracts 1.5 of the coefficients and takes the bit of
   the sign
14.          generate
15.              for (i = 0; i < N; i = i + 1) begin: loop2
16.
17.                  comparator2 #(
18.                      .BITS_FOR_POSITION(1),
19.                      .DATA_WIDTH(DATA_WIDTH)) comparator_with_cons
   tant (
20.                      //.a        (vector_of_symbols[i]),
21.                      .a        (x[N*DATA_WIDTH-i*DATA_WIDTH-
   1:N*DATA_WIDTH-i*DATA_WIDTH-DATA_WIDTH]),
22.                      .b        (16'd192),
23.                      .pos_a    (1'b1),
24.                      .pos_b    (1'b0),
25.                      .pos_max  (output_bits[i]),
26.                      .value_max()
27.                      );
28.              end
29.          endgenerate
30.
31.
32.
33.      endmodule
```

Capítulo 7. ANÁLISIS DE RESULTADOS

En esta sección se muestran y analizan los resultados obtenidos en los distintos experimentos desarrollados en el trabajo. Estos resultados comprenden los obtenidos en las simulaciones usando lenguaje Matlab así como la simulación de la implementación del sistema en lenguaje Verilog.

7.1 SIMULACIONES DE LA PROBABILIDAD DE ERROR

Son varios los parámetros del sistema que afectan al comportamiento de este, lo cual complica conocer reglas que permitan conocer fácilmente los mejores parámetros para cada una de las condiciones. Distintas simulaciones se han realizado con el objetivo de conocer cómo afectan a la probabilidad de error de los bits en la transmisión.

Se consideró como banda de comunicaciones la comprendida entre 2 y 3 GHz. Los símbolos considerados son de 100 ns, lo que provoca un total de 100 portadoras. Se simuló el sistema para distintos números de bloques y tamaño de los mismos, utilizando canales con diferentes SNRs. Además de la probabilidad de error, para cada uno de los sistemas simulados se calculó cual es la capacidad del enlace.

La frecuencia de muestreo del receptor CS también tomó varios valores en las distintas simulaciones. Las siguientes figuras muestran los resultados obtenidos.

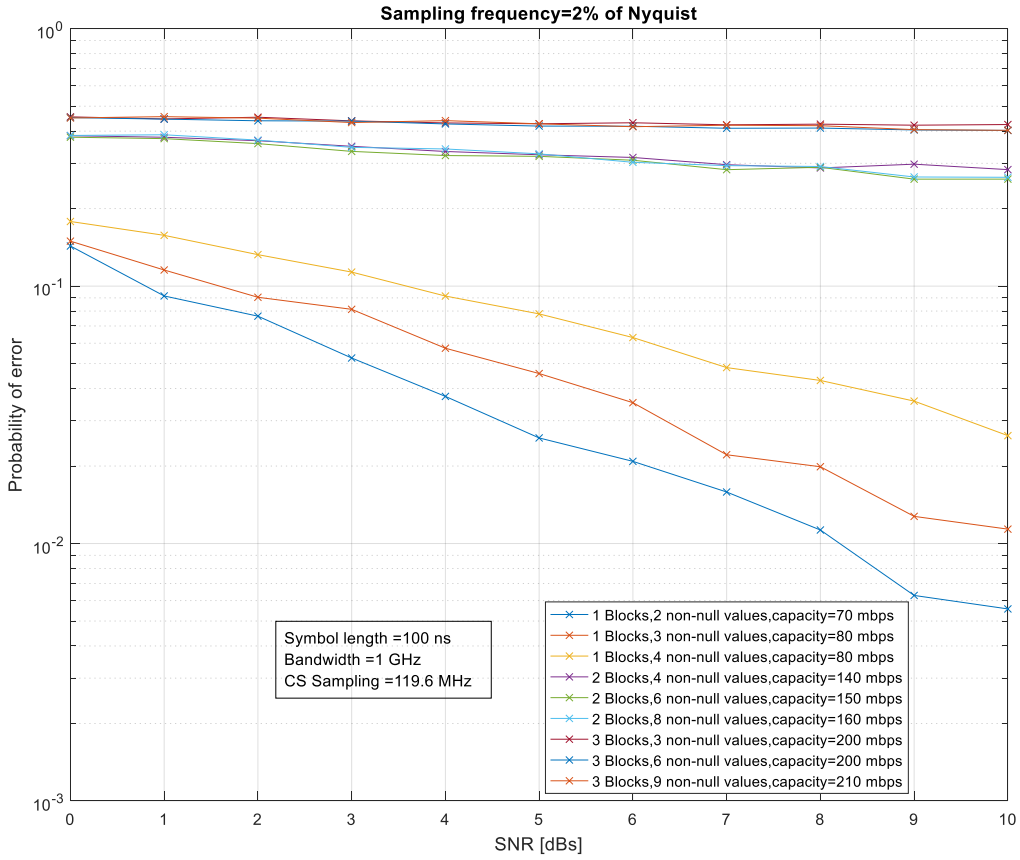

Figura 14 Simulación con una frecuencia de muestreo del 2% de Nyquist

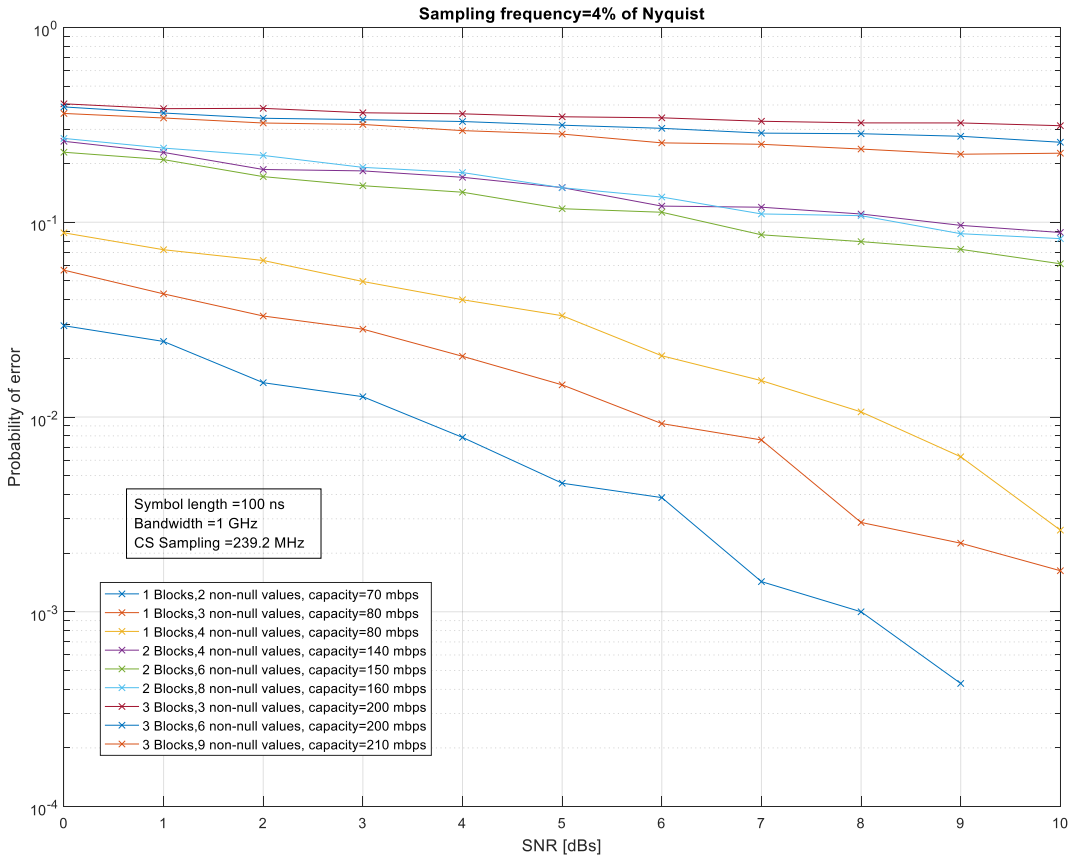

Figura 15 Simulación con una frecuencia de muestreo del 4% de Nyquist

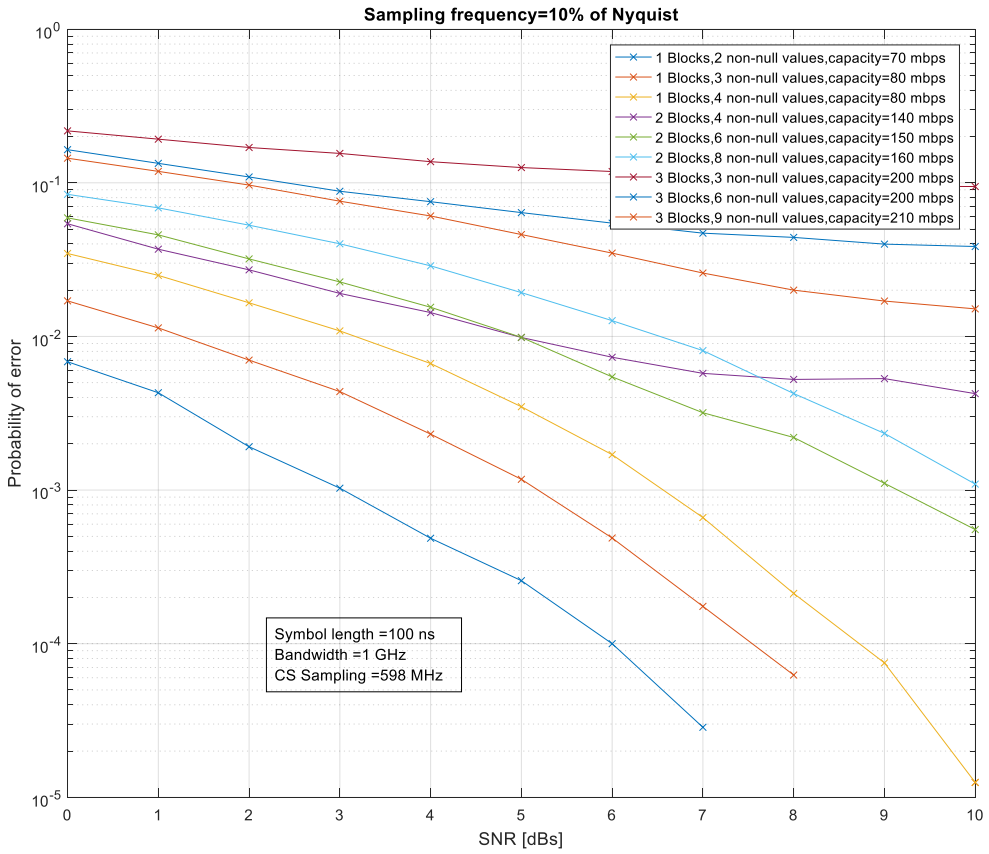

Figura 16 Simulación con una frecuencia de muestreo del 10% de Nyquist

Como era de esperar, el hecho de aumentar la frecuencia de muestreo aumenta la resistencia de la comunicación frente al ruido. Esta ventaja es obtenida a costa de usar un muestreador más caro y con un mayor consumo, además de que el hecho de recoger más muestras también provoca que el procesado y demodulación se vuelve más complejo al requerir productos de matrices más grandes.

Por otro lado, el hecho de aumentar el número de bloques activos provoca un aumento importante en la probabilidad de error. Esto se debe a que es imposible asegurar que las

matrices de cada par/trío de columnas sea ortonormal, por lo que no es posible obtener esta ventaja. Aún así, para un muestreador del 10% de Nyquist se obtuvieron probabilidades de error relativamente bajas, por lo que esta técnica puede ser empleada en algunos casos.

De cara a la implementación, se consideró el caso de un bloque activo con tres posiciones y un 4% de la frecuencia de Nyquist.

7.2 SIMULACIÓN EN PUNTO FIJO

Con el objetivo de conocer cuales son los mejores parámetros para la representación en punto fijo. Para simplificar el proceso de selección, se supuso que de una palabra de N bits uno se usa para el signo y del resto de bits la mitad son para la parte entera y la otra mitad parte decimal. Además, se supone un número entero de bytes para cada una de las palabras.

La simulación se hizo con la librería de punto dijo de Matlab, como ya se ha mencionado. Se midió la probabilidad de error de bit para cada conjunto de parámetros elegido, con el fin de observar cual es el tamaño a partir del cual no merece la pena usar más bits. Los resultados se muestran en la siguiente gráfica:

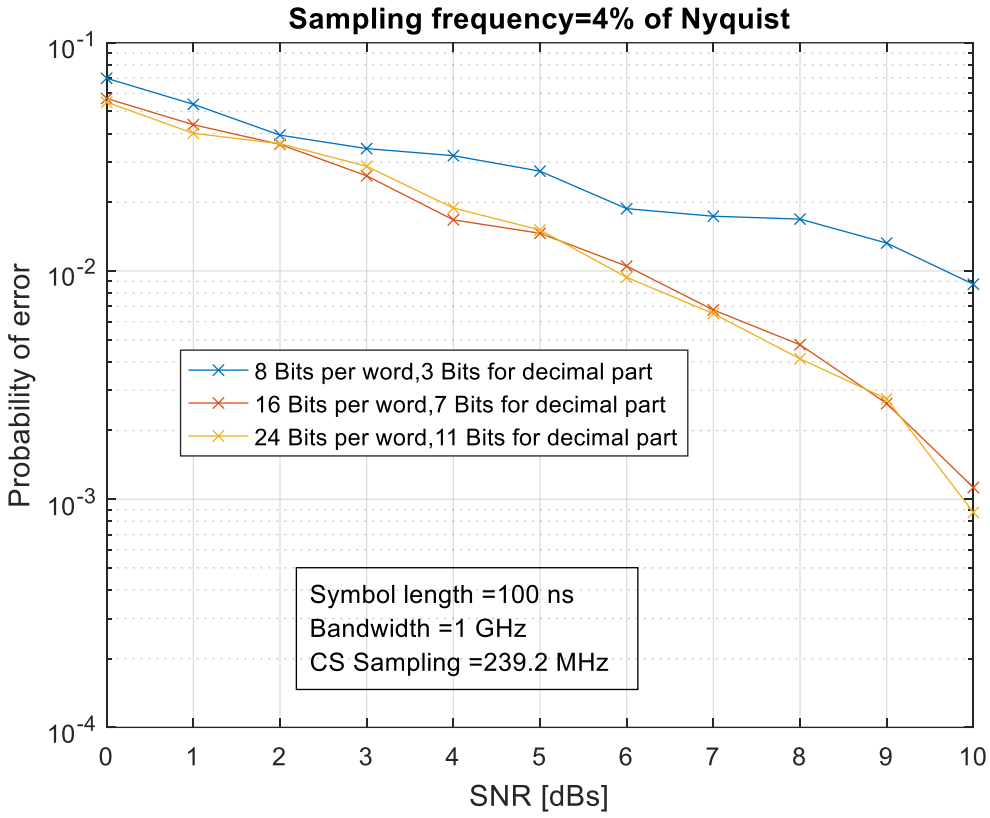

Figura 17 Resultado de la simulación en punto fijo para distintas longitudes de palabras.

Se observa que, mientras que sí que hay una diferencia notable entre usar 8 y 16 bits, apenas la hay al usar 16 o 24 bits. Es por esto que se decidió usar palabras de 16 bits con 7 bits para la parte decimal.

Por lo tanto, los valores positivos se representarán como un array de bits con un 0 en la primera posición, seguido por la representación binaria de la parte entera en 8 bits y la representación binaria en 7 bits de la parte fraccional multiplicada por $2^7 = 128$. Por ejemplo, el número 14,492 será representado por:

$$bin(14) = 00001110$$

$$bin(EnteroMásCercano(0.492 * 2^7)) = bin(63) = 0111111$$

$$FixedPoint_{16,7}(14{,}492) = \underbrace{0}_{signo}\ \underbrace{000011100}_{parte\ entera}\ \underbrace{0111111}_{parte\ fraccional}$$

Por otro lado, los números negativos se representan en complemento a 2, ya que esto facilita las operaciones aritméticas.

En esta representación, el valor más positivo que se puede representar es 255.9922:

$$FixedPoint_{16,7}(511.9844) = \underbrace{0}_{signo}\ \underbrace{11111111}_{parte\ entera}\ \underbrace{1111111}_{parte\ fraccional}$$

El número más negativo que acepta esta representación es:

$$FixedPoint_{16,7}(-512) = \underbrace{1}_{signo}\ \underbrace{00000000}_{parte\ entera}\ \underbrace{0000000}_{parte\ fraccional}$$

La sensibilidad que acepta la representación es $2^{-7} = 0.0078$.

7.3 IMPLEMENTACIÓN

La siguiente imagen muestra la visualización del resultado de sintetizar el código en Verilog.

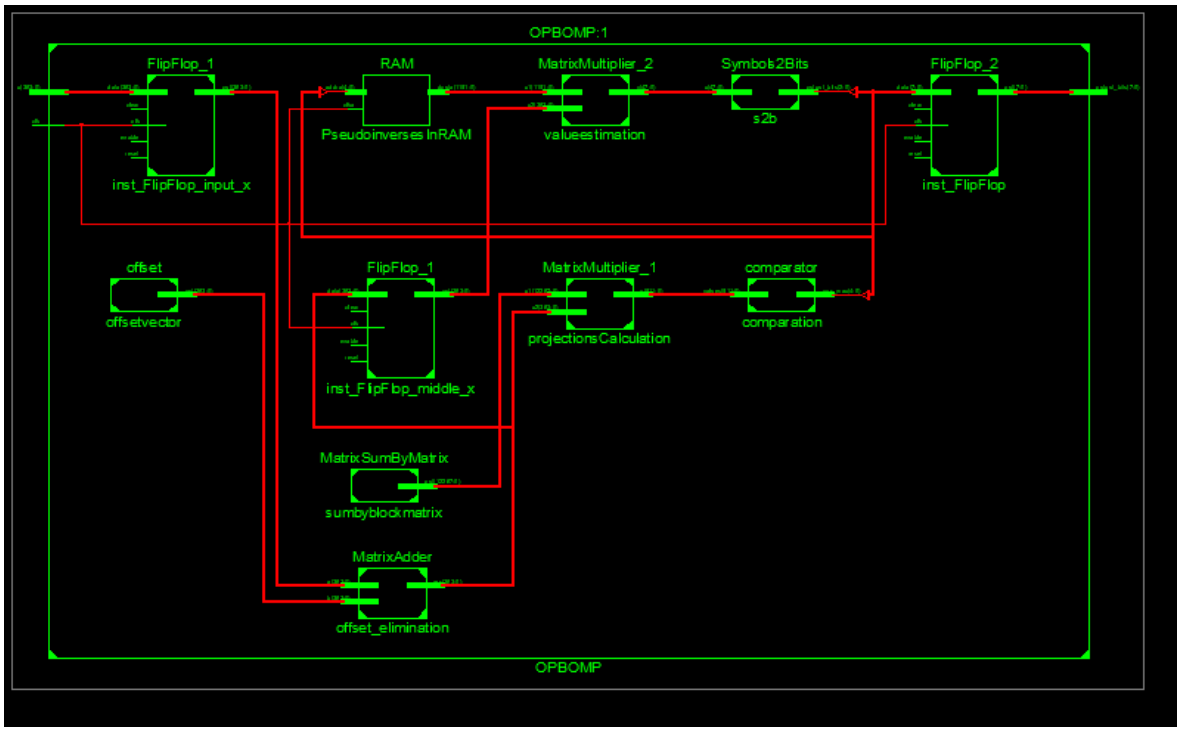

Figura 18 Esquemático RTL del circuito sintetizado

El circuito mostrado en la figura coincide con el del diseño, por lo que el resultado de la síntesis es el correcto.

Se simuló el correcto funcionamiento del circuito mediante el software ISIM. Para ello, se prepararon pruebas unitarias para cada uno de los bloques diseñados y ya mencionados. El código usado para estos bancos de prueba no se muestra aquí debido a su simplicidad, ya que simplemente se componen de unos estímulos y se observa la salida para comprobar que es la esperada. Se comprobó cada bloque con varias entradas distintas y distintos parámetros (tamaños de las matrices, longitudes de los vectores, etc …) y se comprobó que la salida siempre era la esperada, pudiendo confirmar que el circuito descrito realizaba la función deseada. Las siguientes figuras muestran estos resultados.

Para el producto de matrices, se probó con la operación:

$$\begin{pmatrix} 0x0000 & 0x0080 \\ 0x0100 & 0x0180 \end{pmatrix} x \begin{pmatrix} 0x0280 \\ 0x0300 \end{pmatrix} = \begin{pmatrix} 0x0300 \\ 0x0E00 \end{pmatrix}$$

Teniendo en cuenta que la operación se hace en punto fijo con 7 posiciones para los decimales. Se puede observar en la siguiente simulación que el resultado es el adecuado:

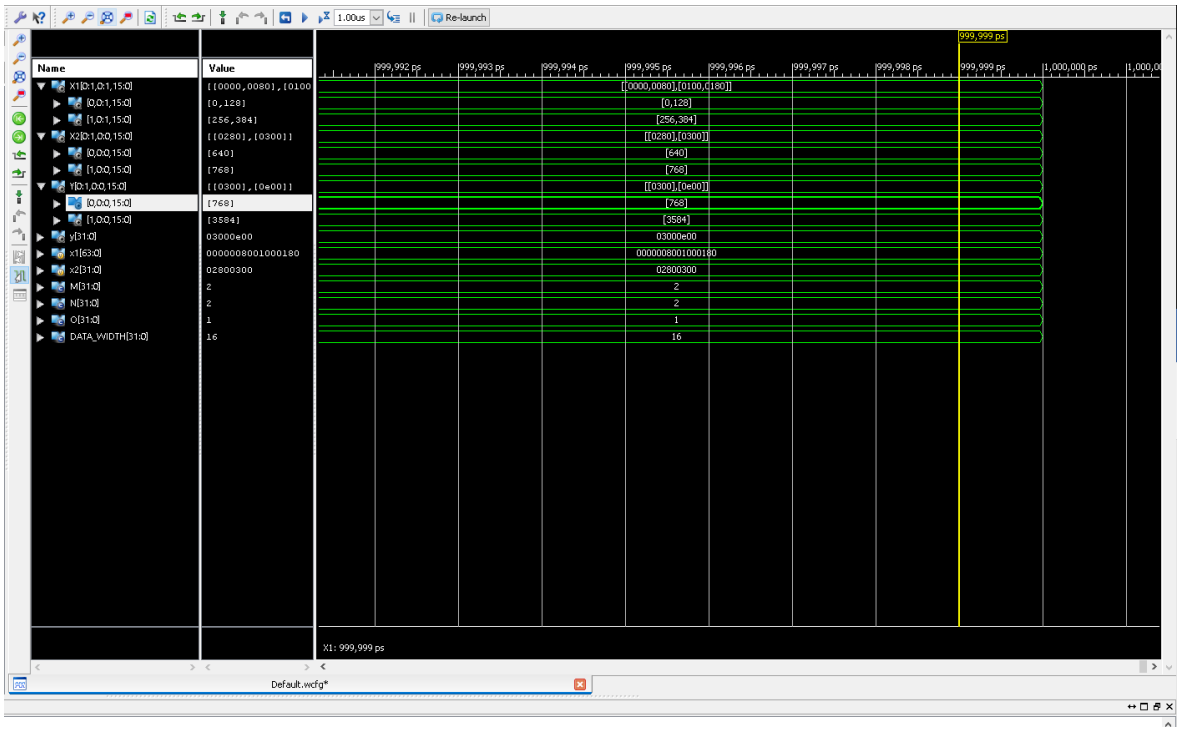

Figura 19 Simulación de la multiplicación de matrices

Del mismo modo, si adaptamos el módulo para operar con una matriz 3x3, obtenemos la siguiente simulación:

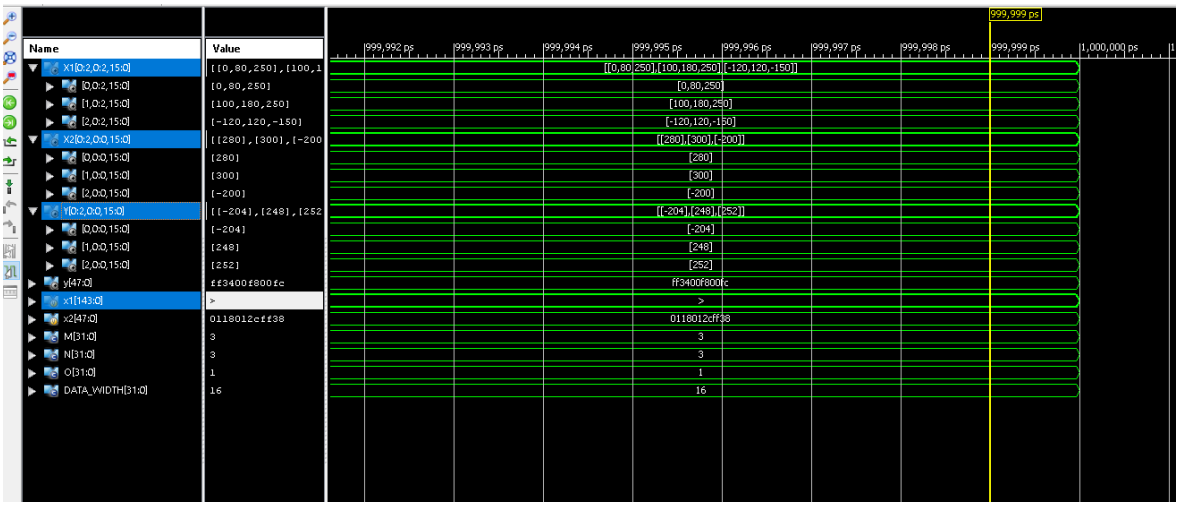

Figura 20 Simulación de producto de matriz 3x3

Dado que funciona correctamente para distintos parámetros, podemos inducir que el comportamiento es el adecuado.

Para el comparador que selecciona la posición del mayor valor, se usó el siguiente testbench:

```
initial begin
                    // Initialize Inputs

                    // Wait 100 ns for global reset to finish
                    #100;

                    // Add stimulus here
                    vectorOfValues[0]=0;
                    vectorOfValues[1]=1;
                          vectorOfValues[2]=2;
                          vectorOfValues[3]=3;
                          vectorOfValues[4]=4;
                          vectorOfValues[5]=5;
                          vectorOfValues[6]=6;
                          vectorOfValues[7]=7;
                          values=values_aux;
                          #50
                          if(pos_max!=7) begin
                                  $display("First test failed");
                                  $finish;
                          end

```

```
23.                     #100
24.                     vectorOfValues[3]=10;
25.                     #10
26.                     values=values_aux;
27.                     #50
28.                     if(pos_max!=3) begin
29.                             $display("Second test failed");
30.                             $finish;
31.                     end
32.
33.                     #100
34.                     vectorOfValues[2]=-25;
35.                     #10
36.                     values=values_aux;
37.                     #50
38.                     if(pos_max!=3) begin
39.                             $display("Third test failed");
40.                             $finish;
41.                     end
42.
43.
44.                     #100
45.                     vectorOfValues[4]=100;
46.                     #10
47.                     values=values_aux;
48.                     #50
49.                     if(pos_max!=4) begin
50.                             $display("Fourth test failed");
51.                             $finish;
52.                     end
53.
54.                     $display("All tests passed");
55.             end
```

Que prueba distintas posibilidades de valores en el vector, mezclando números positivos y negativos, y comprueba que el funcionamiento es el correcto. Todos los tests pasaron satisfactoriamente. La siguiente figura muestra el resultado de la simulación:

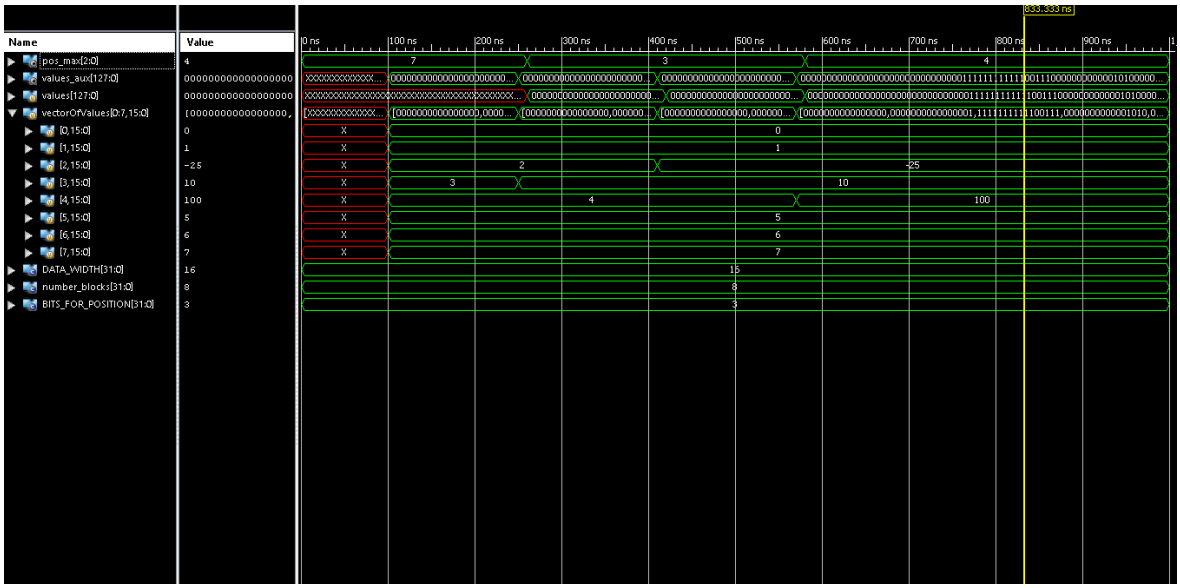

Figura 21 Simulación del seleccionador de la posición con valor más alto en un vector.

Otro test similar es el de la obtención de los bits a partir de los símbolos demodulados. El funcionamiento de este test es muy similar al anterior: se introducen distintos valores y se comparan con el valor frontera (192) para conocer si el resultado es el adecuado. Todas las pruebas pasaron con éxito.

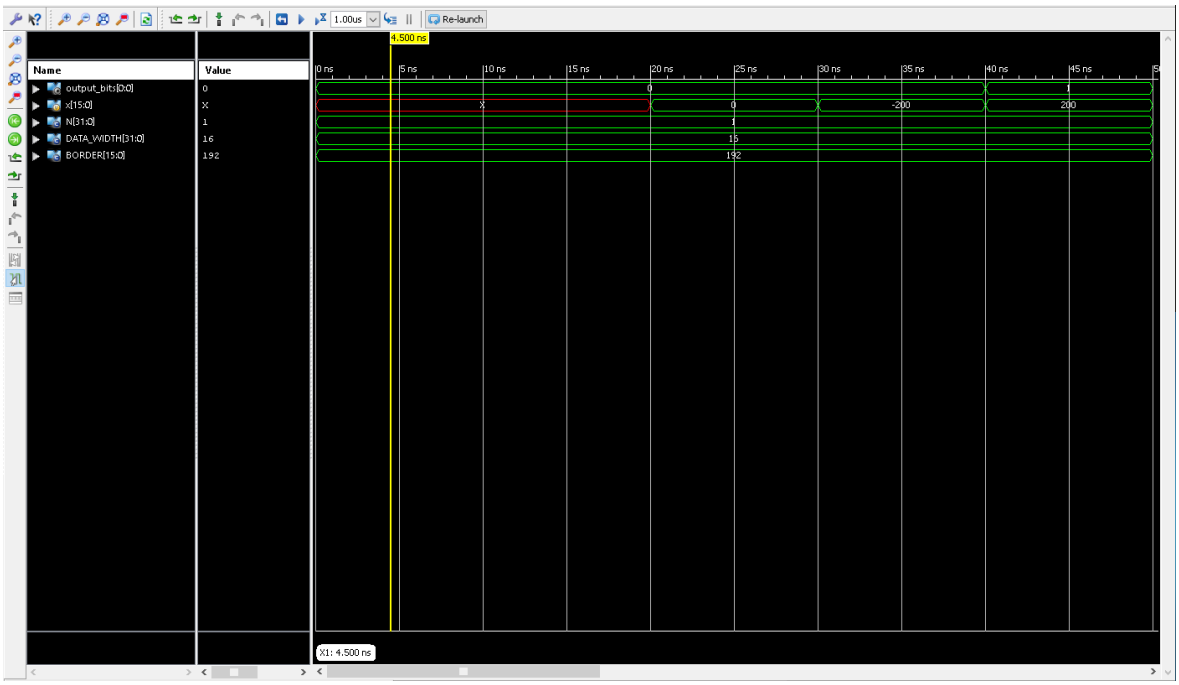

Figura 22 Pruebas del demodulador de bits

También se realizó una simulación del módulo completo. Para ello, se simuló una señal a la entrada y se comprobó que la salida era la misma que la obtenida con la simulación de punto fijo en Matlab. La siguiente figura muestra el resultado de la simulación:

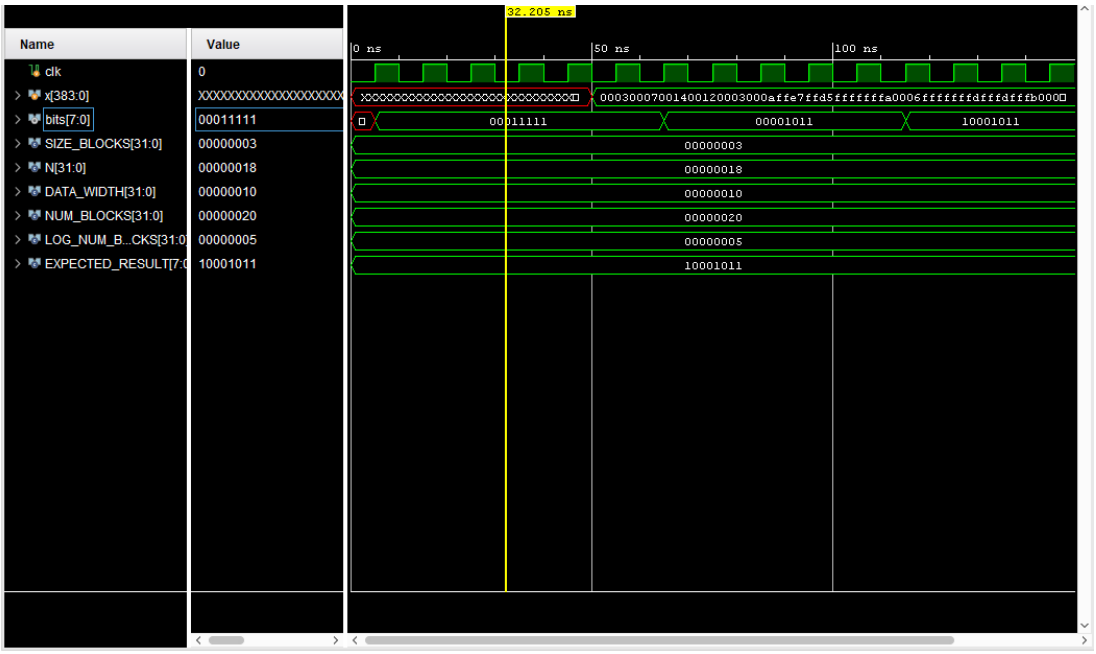

Figura 23 Resultado de la simulación de módulo desarrollado completo.

La secuencia de bits obtenida es la misma que la esperada, como se observa en la simulación. Podemos observar que esta salida se obtiene en el segundo flanco de reloj después de que se lea la entrada, ya que se ha simulado usando una RAM síncrona y por tanto es necesario un ciclo para leer la pseudoinversa a usar. El periodo de reloj usado no es representativo, ya que se usó uno alto para favorecer la velocidad de ejecución de la simulación sin tener efectos en el funcionamiento del circuito (no se tiene en cuenta en este tipo de simulación).

El proceso de síntesis muestra algunos datos de interés sobre el circuito diseñado. El informe indica una aproximación al tiempo correspondiente al camino combinacional más largo presente en el circuito. Este camino es de 41.901 ns, lo cual condiciona la máxima frecuencia de reloj a la que puede funcionar el circuito que es de 23.866 MHz. Por lo tanto, dado que el circuito diseñado funciona en dos ciclos de reloj podemos concluir que el tiempo de decodificación de la señal ronda los 80 ns. Gracias al registro intermedio añadido entre la etapa de detección y la de estimación, es posible hacer funcionar el circuito como un

pipeline, por lo que podemos introducir nuevas señales en cada ciclo de reloj. Esto hace que pueda funcionar con señales de cómo poco 30 ns sin volverse un sistema inestable.

El informe de síntesis también muestra los elementos que ha sintetizado a partir del código. Esto se muestra en la siguiente tabla:

Elemento	*Cantidad*
Registros Flip Flop	776
Comparadores 16 bits	34
Mutliplexadores 1 bit 2 a 1	3
Multiplexadores 16 bits 2 a 1	34
Multiplexadores 5 bits 2 a 1	31

Tabla 3 Clasificación de los elementos detectados por el sintetizador

Del mismo modo, podemos ver los resultados de porcentaje de utilización de los recursos de la FPGA.

Elemento	*Utilización*
Slice Register	6%
Slice Look Up Tables	15%
Inputs/Outputs	385%

Block RAM	50%
BUFG/BUFGCTRL/BUFHCEs	6%

Tabla 4 Porcentaje de utilización de los recursos

Solamente el número de entradas y salidas sale mayor del 100%, lo cual es normal ya que en la práctica las muestras llegarán al dispositivo de una en una y no todas al mismo tiempo por lo que el número real de entradas y salidas será distinto.

Por último, la memoria RAM usada par las pseudoinversas tiene 5 bits para la dirección y lee 1152 bits a la salida (144 bytes), lo que hace un total de 36864 bytes de memoria. En la FPGA, esto se implementa mediante 16 bloques de memoria de 18 Kbytes de memoria cada una.

Capítulo 8. CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se ha podido desarrollar un sistema de comunicaciones de banda ancha que permite transmitir en una banda de frecuencias que en otras circunstancias sería imposible debido al elevado coste de un muestreador suficientemente rápido como para transmitir en una modulación común. Para ello, se creó una forma de modular los bits que permite generar señales dispersas, las cuales permiten aprovecharse de las ventajas de las técnicas de muestreo compresivo.

Debido a la alta complejidad del problema a resolver, inicialmente era complejo resolver el problema en tiempo real por lo que difícilmente podría ser aplicado a un sistema de comunicaciones. Por ello, se realizaron algunas suposiciones que permitían reducir la complejidad del algoritmo. Como fruto de esto, se pudo resolver la resolución del problema notablemente, provocando que sea posible implementarlo sin tener que recurrir a costes muy altos.

Las simulaciones mostraron que es posible usar la modulación desarrollada en un sistema de comunicaciones en un canal que añada ruido gaussiano. Las probabilidades de error para algunos parámetros del sistema son suficientes para garantizar una buena calidad en la transmisión.

En la modulación desarrollada, los cálculos en el receptor son notablemente más caros y complejos que en el transmisor. Para probar que el sistema desarrollado puede ser puesta en práctica en un sistema real, se implementó el procesado de la señal en el receptor en un circuito digital usando lenguaje Verilog. Tras las simulaciones, se comprobó su correcto funcionamiento demostrando por tanto que puede ser implementado en una cantidad limitada de recursos sin un coste muy alto.

Por lo tanto, se demostró que las ventajas de la modulación diseñada pueden aprovecharse para un sistema de comunicaciones. Además de los beneficios asociados a la baja velocidad del muestreador ya mencionadas, cabe destacar que aparecen otras ventajas secundarias como puede ser la seguridad, ya que es imposible la obtención de los bits a partir de la señal si no se conoce la matriz con la que estos fueron modulados. Esta matriz puede generarse de forma aleatoria a partir de una semilla, por lo que si emisor y receptor acuerdan previamente a la transmisión en esta semilla puede lograrse una capa de seguridad a la comunicación.

De cara a futuros trabajos, existe la posibilidad de continuar el análisis de sensibilidad del sistema en circunstancias no estudiadas en este trabajo. Como ejemplo, sería interesante conocer cuál es el efecto de transmitir en un canal que presente multicamino, por lo que el canal puede ser modelado como un filtro de convolución. Al no realizar el procesado en el dominio de la frecuencia, el proceso de ecualización no es tan sencillo como puede ser en un sistema OFDM.

Otro tema interesante para futuros trabajos consiste en el estudio de las propiedades óptimas de la matriz de muestreo. En este trabajo se generó de forma aleatoria a partir de una semilla, pero resulta lógico pensar que pueden existir semillas que generen matrices mejores que otras. Una buena selección de esta semilla puede provocar que haya diferencias en la calidad de la recepción.

Capítulo 9. BIBLIOGRAFÍA

[1] X. Crespo, «planetachatbot.com,» 18 Enero 2017. [En línea]. Available: https://planetachatbot.com/qu%C3%A9-es-una-fpga-y-por-qu%C3%A9-jugar%C3%A1n-un-papel-clave-en-el-futuro-e76667dbce3e. [Último acceso: 5 Junio 2018].

[2] R. M. B. W. I. M. N. a. S. L. S. Qaisar, «Compressive sensing: From theory to applications, a survey,,» *Journal of Communications and Networks,* vol. 15, pp. 443-456, 2013.

[3] H. J. K. M. a. P. W. G. Huang, «Lensless imaging by compressive sensing.,» *IEEE International Conference on Image Processing,* 2013.

[4] D. J. Brady, C. Kerkil, D. L. Marks, R. Horisaki y S. Lim, «Compressice holography,» *Opt. Express,* 2009.

[5] D. D. a. J. M. P. Michael Lustig, «Sparse mri: The application of compressed sensing for rapid mr imaging,» *Magnetic Resonance in Medicine,* 2007.

[6] M. H. Firooz y S. Roy, «Network tomography via compressed sensing,» *EEE Global Telecommunications Conference GLOBECOM 2010,* 2010.

[7] J. H. G. R. a. R. N. Waheed U. Bajwa, «Compressed Channel Sensing,» *Information Sciences and Systems,* 2008.

[8] Z. H. R. C. Q. B. M. S. Peng Zhang, «A Compressed Sensing Based Ultra-Wideband Communication System,» *2009 IEEE International Conference on Communications*, 2008.

[9] A. N. a. A. T. rne Svensson, «Ultra-Wideband Communication Systems: Technology and Applications,» *EURASIP Journal onWireless Communications and Networking*, 2006.

[10] S. G. M. a. Z. Zhang, «Matching pursuits with time-frequency dictionaries,» *IEEE Transactions on Signal Processing*, pp. 3397-3415, 1993.

[11] R. R. a. P. S. K. Y. C. Pati, «Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition,» *Proceedings of 27th Asilomar Conference*, pp. 40-44, 1993.

[12] Y. L. J. H. X. C. X. Z. Weijing Shi, «An Extensible and Real-time Compressive Sensing Reconstruction Hardware for WBANs using OMP,» *2013 IEEE 10th International Conference on ASIC, Shenzhen*, pp. 1-4, 2013.

[13] M. Elad, Sparse and Redundant Representations From Theory to Applications in Signal and Image Processing, 2010.