Síntesis de alto nivel de algoritmo CoSaMP para reconstrucción de señales a partir de muestras incompletas

por

Cristhian Alfonso Figueroa Figueroa

Tesis sometida como requisito parcial para obtener el grado de

Maestro en Ciencias en el Área de Electrónica

en el

Instituto Nacional de Astrofísica, Óptica y Electrónica
Tonantzintla, Puebla

Supervisada por:

Dr. Juan Manuel Ramírez Cortés, INAOE

©INAOE 2017
El autor otorga al INAOE el permiso de reproducir y distribuir copias en su totalidad o en parte de esta tesis

Síntesis de alto nivel de algoritmo CoSaMP para reconstrucción de señales a partir de muestras incompletas

Tesis de Maestría

Por:

Ing. Cristhian Alfonso Figueroa Figueroa

Asesor:

Dr. Juan Manuel Ramírez Cortés

Instituto Nacional de Astrofísica Óptica y Electrónica
Coordinación de Electrónica

Tonantzintla, Puebla. Noviembre 2017

Agradecimientos

Al Instituto Nacional de Astrofísica Óptica y Electrónica, por darme la oportunidad de prepararme mejor académicamente.

Al Dr. Juan Manuel Ramírez Cortés por el tiempo que dedicó para guiarme durante mi estadía en el instituto y con este trabajo de tesis.

A mis profesores del INAOE, por sus enseñanzas durante estos dos años.

A mis sinodales: Dr. José Rangel, Dr. Jorge Martínez Carballido y Dr. Israel Cruz Vega, por sus correcciones.

A mi madre y hermana por su apoyo incondicional.

A mis amigos y compañeros del INAOE: Carlos, Cesar, Miguel, Jorge, Betty, Balam, Sergio, Dania y los que faltan por su ayuda y consejos.

A CONACyT por la beca otorgada.

Resumen

El sensado comprimido es una técnica desarrollada en años recientes cuyo objetivo es la adquisición de una señal con representación dispersa o comprimible en determinado espacio vectorial, con una cantidad de muestras por debajo del límite establecido por el Teorema de Nyquist. En esta tesis se presenta el desarrollo del algoritmo de reconstrucción de señales CoSaMP a través de síntesis de alto nivel, haciendo uso de la herramienta Vivado HLS de la empresa Xilinx para una posterior implementación en un FPGA. Se analiza robustez del algoritmo bajo experimentos de reconstrucción de señales sintéticas limpias, señales con ruido y finalmente señales reales con aplicación al sensado de señales EEG. Se presentan resultados comparativos relacionados con la fiabilidad de las señales reconstruidas, el uso de recursos de hardware y la latencia con un FPGA Virtex-7 de la empresa Xilinx.

Abstract

Compressed sensing is a novel technique recently developed whose objective is the recovery of a signal with sparse representation or compressed on a vector space with much less samples stablished by the sampling theorem. On this thesis, the development of the CoSaMP algorithm for signal reconstruction is presented, and making use of the tool Vivado HLS from Xilinx, different arquitectures are proposed for a future implementation on a FPGA. The robustness of the algorithm is tested under reconstruction experiments using synthetic clean signals, noisy signals and real signals with aplication on the sensing of EEG signals. Comparative results on reliability, hardware utilization and latency are presented using a Virtex-7 FPGA from Xilinx.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1
Introducción

En el campo de procesamiento de señales es una tarea común querer reconstruir una señal con una serie de mediciones de esta. En general, esta tarea es imposible porque no hay forma de reconstruir una señal durante las veces en que la señal no es medida. Sin embargo, con conocimiento previo o asumiendo información acerca de la señal, es posible reconstruir perfectamente la señal con una serie de mediciones.
La teoría clásica de muestreo dice que para muestrear una señal es necesario usar una frecuencia de al menos dos veces la frecuencia máxima la señal [1]. Sin embargo, se ha observado que las señales no cubren uniformemente el espacio en el cual viven ni tampoco se limitan a un subespacio del mismo; las señales pueden expresarse como una combinación lineal de unos pocos elementos de una base conocida. Esta observación surge básicamente de realizar una transformación especial de la señal, por ejemplo utilizando una base tipo Fourier y, observar que los coeficientes obtenidos concentran su energía en unos pocos coeficientes. En este sentido se dice que la señal en cuestión tiene una representación dispersa en la base de Fourier.

El sensado comprimido es una técnica novedosa la cual permite el muestreo de señales dispersas por debajo del nivel de Nyquist al utilizar algoritmos de reconstrucción basados en encontrar la solución dispersa a sistemas indeterminados.
Fue por el 2004 cuando Emmanuel Candès, Terence Tao y David Donoho probaron que dado cierto conocimiento de la dispersidad de la señal, la señal podía ser reconstruida con incluso menos muestras que las mínimas necesarias por el teorema de muestreo[2, 3]. En sensado comprimido típicamente se empieza tomando muestras comprimidas en una base diferente en donde la señal se sabe que es dispersa.

Los resultados obtenidos por Emmanuel Candès, Terence Tao y David Donoho, muestran que el número de esas mediciones comprimidas pueden ser pequeñas y de todas formas contener la información necesaria para poder reconstruir la señal.

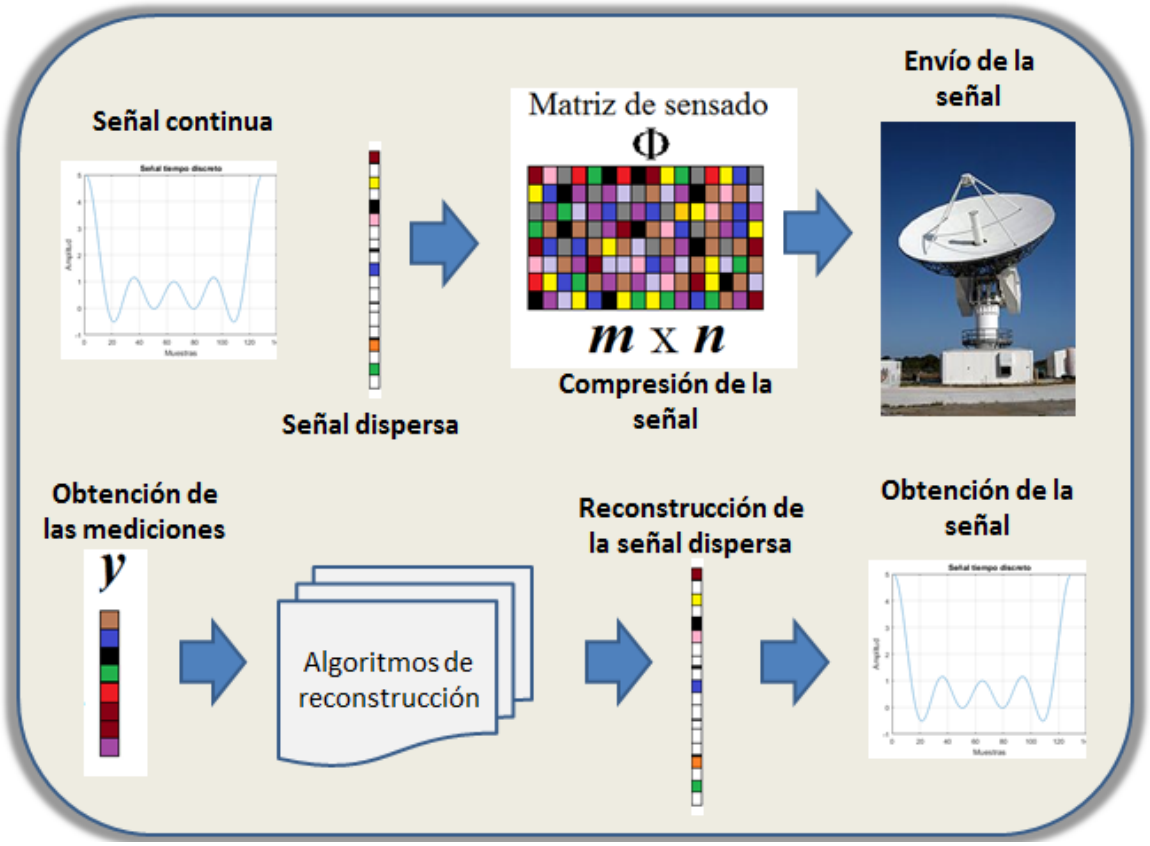

Figura 1.1: Introducción al Sensado comprimido

1.1. Problemática

La reconstrucción de señales son llevadas a cabo por algoritmos de reconstrucción los cuales consumen demasiado tiempo en implementaciones de software debido a que requieren gran cantidad de multiplicación de matrices. La complejidad computacional no es mayor problema para aplicaciones fuera de linea (ej. reconstrucción de resonancias magnéticas[4]), sin embargo, el problema se vuelve mas complejo cuando se requiere procesamiento en tiempo real(ej. radar[5], detección facial[6], procesamiento de imágenes[7],etc). Para cumplir con requerimientos

de alta velocidad y/o consumo de potencia, se pueden utilizar implementaciones en hardware utilizando FPGA's, ya que los algoritmos se ejecutan en paralelo. Sin embargo, los algoritmos de reconstrucción de señales utilizados en sensado comprimido requieren de un esfuerzo computacional elevado, por eso es que aún se buscan algoritmos que reconstruyan señales a alta velocidad y que puedan ser implementados eficientemente en FPGA's de cualquier tipo. Se propone realizar arquitecturas para un FPGA que cumpla con los requerimientos de velocidad y precisión en la señal reconstruida y que pueda ser utilizado en aplicaciones que requieran un procesamiento en tiempo real.

1.2. Solución propuesta

1.2.1. Objetivo general

Mediante la utilización de síntesis de alto nivel, lograr diferentes arquitecturas para un FPGA del algoritmo CoSaMP, que pueda ser utilizado para reconstruir señales en tiempo real.

1.2.2. Objetivos específicos

- Obtener resultados de latencia y de recursos utilizados para cada diferente arquitectura presentada.
- Las arquitecturas obtenidas serán comparables o mas eficientes en en cuanto a utilización de recursos y/o latencia que las ya presentadas anteriormente.
- Reconstruir señales en donde las muestras puedan estar contaminadas con algún tipo de ruido.
- Las arquitecturas presentadas serán capaces de reconstruir señales reales.

1.3. Organización de la tesis

La tesis esta organizada en cinco capítulos. El primer capítulo es la parte introductoria de la tesis. Se presenta el marco general sobre el sensado comprimido y el enfoque de la tesis. Se repasan algunos de los campos en los cuales se están

desarrollando aplicaciones utilizando el esquema de sensado comprimido. En el capítulo 2 se presentan las bases teóricas necesarias para entender el problema y las herramientas para resolverlo. En el capitulo 3 se presenta el método y la forma en que se abordo el problema para cumplir con los diferentes objetivos planteados. También se muestran trabajos planteados anteriormente en el contexto de sensado comprimido. En el capítulo 4 se presentan los resultados obtenidos, latencia y utilización de recursos necesarios en cada etapa resultante de la programación del algoritmo. En el capítulo 5 se muestran conclusiones y discusiones sobre el algoritmo implementado y sobre como mejorar el desempeño para posible trabajo futuro.

Capítulo 2
Marco Teórico

2.1. Teoremas de muestreo

El muestreo de señales es fundamental para representar y recuperar señales continuas en el campo de procesamiento de señales. Normalmente utilizamos el teorema de muestreo de Shannon - Nyquist el cual dice que la frecuencia de muestreo debe ser al menos dos veces la frecuencia máxima de la señal en cuestión.[1]

$$f_{sampling} \geqq 2f_{signal} \tag{2.1.1}$$

Hoy en día se procesan grandes cantidades de información, por lo tanto se buscan técnicas las cuales reduzcan el tiempo de procesado de dichas señales. La compresión de datos es una herramienta muy útil, ya que permite representar la información de una forma mas compacta.

El sensado comprimido es una técnica de procesamiento de señales en la cual se adquieren y reconstruyen señales buscando soluciones a sistemas lineales indeterminados. Se requiere tener conocimiento previo de la dispersidad de la señal para recuperarla con muchas menos muestras que las requeridas por el teorema de Shannon-Nyquist[8].

Para hacer sensado comprimido se deben cumplir dos condiciones para obtener una reconstrucción exitosa; la primera es que la señal tenga una representación dispersa en cualquier dominio y la segunda es sensar con una matriz adecuada la cual deberá satisfacer la propiedad de isometría restringida(RIP)[9].

A primera vista pareciera ser que el sensado comprimido violase el teorema de muestreo porque depende de la dispersidad de la señal y no de su máxima frecuencia; sin embargo, es una idea equívoca debido a que el teorema de muestreo garantiza una reconstrucción perfecta dadas suficientes y no necesarias condiciones. Algunas señales pueden ser muestreadas muy por debajo del nivel del teorema de muestreo si y sólo si son dispersas en alguna base conocida usando sensado comprimido.

Un sistema de ecuaciones lineales indeterminado tiene más variables que ecuaciones y generalmente tienen infinidad de soluciones. El sistema de ecuaciones esta dado por:

$$y = \Phi\Psi s = \Phi x \tag{2.1.2}$$

Donde:

y son las mediciones tomadas, $y \in R^m$

Φ es la matriz de sensado, $\Phi \in R^{mxn}$

Ψ es la matriz base, $\Psi \in R^{mxn}$

s señal a ser comprimida, $x \in R^n$

x es la representación dispersa de la señal, $x \in R^n$

Una representación gráfica se puede ver en la figura 2.1.

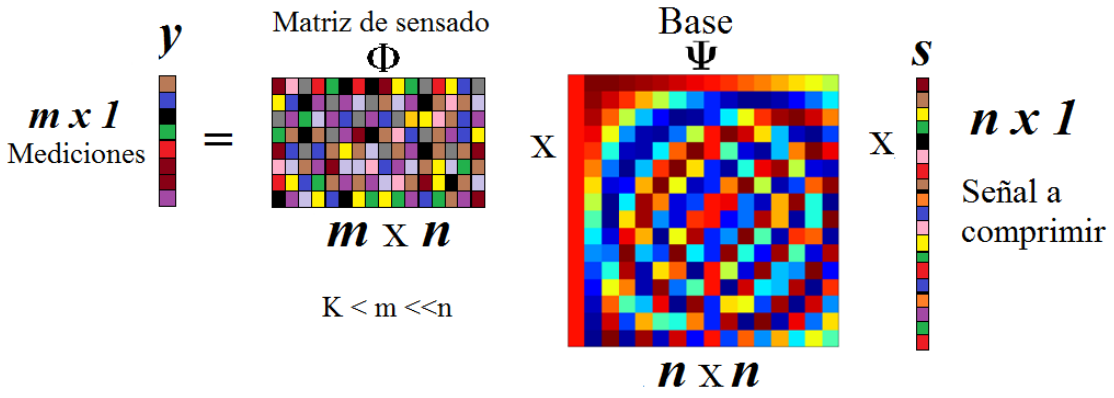

Figura 2.1: Sensado comprimido.

Requerimos solución para x que es la representación dispersa de la señal que queremos comprimir. Para obtener dicha solución se imponen ciertas condiciones o restricciones que se explicarán en las siguientes secciones. Cabe decir que no todos los sistemas indeterminados de ecuaciones lineales tienen una solución dispersa; sin embargo, si hay una única solución dispersa al sistema indeterminado, entonces el sensado comprimido permite la reconstrucción de esa solución[10].

2.2. Bases y dispersidad

Una de las técnicas más populares para la compresión de señales es conocida como "codificación por transformada", y trata de buscar una base o estructura la cual provea una representación dispersa o compresible para las señales de interés[11, 12, 13]. Por representación dispersa queremos decir que para una señal de tamaño n, podemos representarla con $k << n$ coeficientes diferentes de cero; por representación compresible queremos decir que la señal está bien aproximada por una señal con solo k coeficientes diferentes de cero. Los dos tipos de señales, dispersa o compresible pueden ser representadas con gran fidelidad al preservar solamente los valores y posición de los coeficientes más grandes de una señal. Este proceso se llama aproximación dispersa, y es la base de los esquemas de codificación por transformada que explota la dispersidad y la compresibilidad de las señales.

Un conjunto de $\{\psi_i\}_{i=1}^{n}$ es llamada una base para R^n si los vectores forman un arreglo R^{nxn} y son linealmente independientes. Específicamente para cualquier $x \in R^n$, existen coeficientes $\{s_i\}_{i=1}^{n}$ que nos dan una representación:

$$x = \sum_{i=i}^{n} s_i \psi_i \tag{2.2.1}$$

Denotamos como Ψ una matriz de tamaño nxn con columnas dadas por ψ_i y ponemos como s el vector de tamaño n, entonces representamos la ecuación de una forma mas compacta como:

$$x = \Psi s \tag{2.2.2}$$

Normalmente podemos representar señales por una combinación lineal de solo unos cuantos elementos de una base conocida o diccionario. Matemáticamente decimos que una señal x es k-dispersa cuando tiene a lo mucho k valores no ceros. Denotamos como:

$$\Sigma_k = \{x : \|x\|_0 \leq k\} \tag{2.2.3}$$

el conjunto de todas las señales k-dispersas. Normalmente se trabaja con señales que en sí no son dispersas, sin embargo permiten una representación dispersa en alguna base Ψ.
Una base ortonormal tiene la ventaja de que los coeficientes c pueden ser fácilmente calculados como :

$$c_i = \langle x, \psi_i \rangle \tag{2.2.4}$$

ó

$$c = \Psi^T x \tag{2.2.5}$$

en notación matricial. Puede ser fácilmente comprobable debido a la ortonormalidad de las columnas de Ψ tenemos que $\Psi^T \Psi = I$, donde I es la matriz identidad de tamaño nxn[10].

2.3. La transformada como base en sensado comprimido

Algunas de las transformadas más utilizadas en procesamiento de señales nos permiten obtener representaciones dispersas de ciertas señales permitiéndonos usarlas como base en un sistema de sensado comprimido. A continuación se presentan tres de las mas utilizadas.

2.3.1. Transformada de Fourier discreta (DFT)

En matemáticas, la transformada discreta de Fourier o DFT es un tipo de transformada discreta utilizada en el análisis de Fourier. Transforma una función matemática en otra, obteniendo una representación en el dominio de la frecuencia, siendo la función original una función en el dominio del tiempo. Pero la DFT requiere que la función de entrada sea una secuencia discreta y de duración finita.

Dichas secuencias se suelen generar a partir del muestreo de una función continua. La secuencia de N números complejos de entrada se transforma en la secuencia de N números complejos $X_0, X_1, \ldots, X_{n-1}$ mediante la DFT con la formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi i}{N} kn} \quad k = 0, ..., N-1 \tag{2.3.1}$$

Donde i es la unidad imaginaria y $e^{\frac{-2\pi i}{N}}$ es la N-ésima raíz de la unidad. Esta expresión se puede escribir también en términos de una matriz DFT; cuando se escala de forma apropiada se convierte en una matriz unitaria y X_k puede ser entonces interpretado como los coeficientes de x en una base ortonormal.

La transformada inversa de Fourier discreta (IDFT) viene dada por :

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \quad n = 0, ..., N-1 \tag{2.3.2}$$

Los números complejos X_k representan la amplitud y fase de diferentes componentes sinusoidales de la señal de entrada x_n. La DFT calcula X_k a partir de x_n, mientras que la IDFT muestra como calcular x_n como la suma de componentes sinusoidales[14].

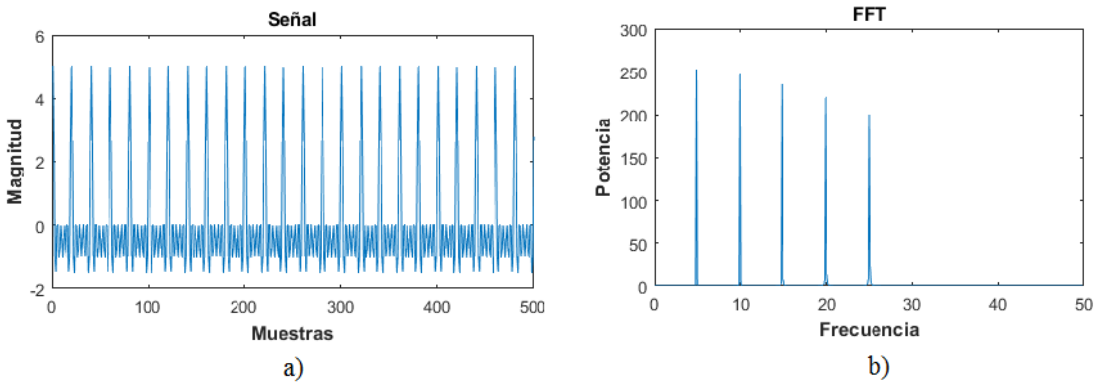

Figura 2.2: a)Señal en tiempo discreto. b)DFT de la señal (Representación de la señal en la frecuencia con pocos coeficientes).

2.3.2. Transformada coseno discreta (DCT)

En la mayoría de aplicaciones los datos son valores reales. Por esta razón, puede ser beneficioso usar la DFT de una forma en la que solo obtenemos valores reales. Esto se puede lograr al extender la secuencia de los datos X_m exhibiendo así la simetría necesaria para que la DFT X_k de valores reales. La transformada coseno discreta (DCT) es una transformada basada en la transformada de Fourier discreta, pero solo utiliza números reales[15]. La DCT de una secuencia de datos $X(m)$, $m = 0, 1, \ldots, (M-1)$ se define como:

$$\begin{aligned} G_x(0) &= \frac{\sqrt{2}}{M} \sum_{m=0}^{M-1} X(m) \\ G_x(k) &= \frac{2}{M} \sum_{m=0}^{M-1} X(m) \cos \frac{(2m+1)k\pi}{2M} \quad k = 1, 2, ..., (M-1) \end{aligned} \tag{2.3.3}$$

Donde $G_x(k)$ es el k-ésimo coeficiente.

La transformada inversa discreta coseno (IDCT) se define como :

$$X_m = \frac{1}{\sqrt{2}} G_x(0) + \sum_{k=1}^{M-1} G_x(k) \cos \frac{(2m+1)k\pi}{2M} \quad m = 0, 1, ..., (M-1) \tag{2.3.4}$$

Si la ecuación anterior se escribe en su forma matricial A de tamaño M x M que describe la transformación coseno, entonces la propiedad de ortogonalidad puede ser expresada como:

$$A^T A = \frac{M}{2} I \tag{2.3.5}$$

Donde A^T es la matriz transpuesta de A e I es la matriz identidad[16].

2.3.3. Transformada wavelet discreta (DWT)

La transformada wavelet discreta consiste en dividir recursivamente la señal en sus componentes de baja y alta frecuencia. Los componentes de baja frecuencia nos dan una aproximación de la señal, mientras que los componentes de alta frecuencia dan el detalle a la señal[10]. La DWT de una señal x es calculada al pasarla por una serie de filtros. Primero, las muestras se pasan por un filtro pasa-bajas con

respuesta al impulso g dando como resultado una convolución de las dos.

$$A_n = (x * g)(n) = \sum_{k=-\infty}^{\infty} x[k]g[n-k] \tag{2.3.6}$$

La señal también se descompone simultáneamente utilizando un filtro pasa-altas con respuesta al impulso h.

$$D_n = (x * h)(n) = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{2.3.7}$$

La salida representan los coeficientes de detalle (pasa-altas) y los coeficientes de aproximación (pasa-bajas). Es importante que los dos filtros estén relacionados uno con otro y son conocidos como filtros espejo en cuadratura[17]. Sin embargo, como la mitad de las frecuencias de han sido removidas, la mitad de las muestras se pueden descartar de acuerdo al teorema de Shannon-Nyquist [1]. La salida del filtro pasa-bajas es submuestreado por un factor de 2 y lo mismo con el filtro pasa-altas. Se puede seguir procesando la señal al pasar el número de veces necesarios por los filtros con la mitad de la frecuencia de corte de los anteriores.
Esta descomposición divide a la mitad la resolución en tiempo porque solo la mitad de cada filtro caracteriza a la señal. Sin embargo, cada salida tiene la mitad de la banda de frecuencia de la entrada y por lo tanto, la resolución en la frecuencia es el doble[18].

Para la transformada inversa es casi el mismo procedimiento, con la diferencia que agregamos ceros a los coeficientes de detalle y aproximación cada factor de 2 y se hace convolución con los filtros de reconstrucción.

2.4. Matriz de sensado

Para hacer sensado comprimido debemos tener en cuenta dos cuestiones principales; la primera es como diseñar la matriz de sensado Φ para que mantenga la información de la señal x, y la segunda es como recuperar x con y mediciones. En el caso donde la información es dispersa o compresible, veremos que se pueden diseñar matrices Φ con $m << n$ que aseguren que podremos reconstruir la señal original[10].

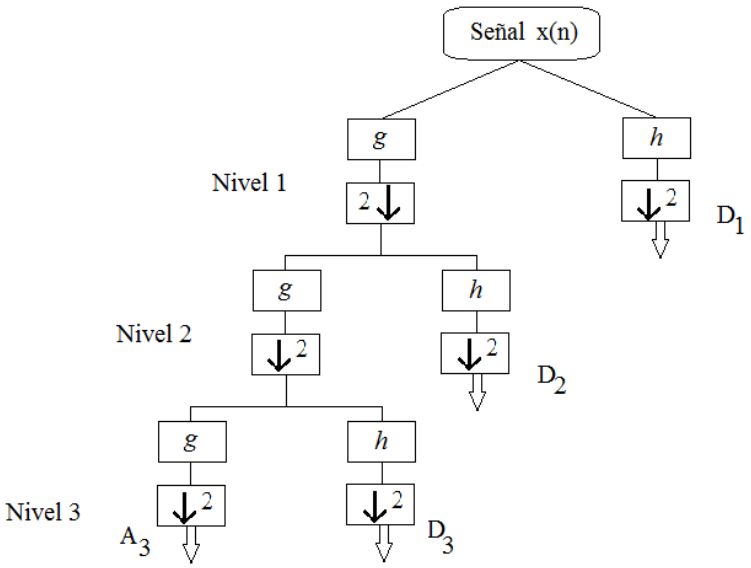

Figura 2.3: Descomposición wavelet a tres niveles

En esta sección se ven las condiciones que las matrices de sensado deben cumplir y como construirlas para que la reconstrucción de la señal sea satisfactoria.

2.4.1. Propiedad de espacio nulo (NSP)

Una propiedad clave que necesitamos conocer es la spark de una matriz Φ. La spark de una matriz Φ es el número más pequeño posible de columnas que son linealmente dependientes.

El espacio nulo de una matriz Φ, se define como:

$$\mathcal{N}(\Phi) = \{z : \Phi z = 0\} \tag{2.4.1}$$

Si queremos poder reconstruir todas las señales dispersas x, entonces queda claro que para cualquier par de vectores distintos x y $x' \in \Sigma_k$, debemos tener que $\Phi x \neq \Phi x'$, porque de lo contrario sería imposible distinguir x de x' basándonos solo en las mediciones de y. Al observar si $\Phi x = \Phi x'$ entonces $\Phi(x - x') = 0$, con $x - x' \in \Sigma_{2k}$, vemos que Φ únicamente representa todas las $x \in \Sigma_k$, si y sólo si

$\mathcal{N}(\Phi)$ no contiene vectores en Σ_{2k} [10]. Mientras hay muchas formas equivalentes de caracterizar esta propiedad, una de las más comunes es conocida como la spark. Esta definición nos permite obtener la siguiente garantía:

Para cualquier vector $y \in R^m$, existe a lo mucho una señal $x \in \Sigma_k$ que hace que $y = \Phi x$ si y sólo si la $spark(\Phi) > 2k$[19].

A partir de aquí es fácil ver entonces que $m \geq 2k$.
Si todas las columnas son linealmente independientes, la spark de una matriz Φ es infinito. Cuando tratamos con vectores dispersos exactos, la spark nos da una caracterización completa de cuando la reconstrucción dispersa es posible. Sin embargo, si trabajamos con señales dispersas aproximadas debemos considerar condiciones mas restrictivas en el espacio nulo de Φ [20].

2.4.2. Propiedad de isometría restringida(RIP)

Mientras que la propiedad de espacio nulo es necesaria y suficiente para darnos garantías de reconstrucción de señales, estas no toman en cuenta el caso del ruido. Cuando las mediciones están contaminadas con ruido o han sido modificadas por algún error por ejemplo de cuantización, será necesario considerar condiciones más fuertes. Candès y Tao introdujeron la siguiente condición de isometría en matrices Φ y establecieron su rol importante en sensado comprimido[21]. Una matriz Φ satisface la propiedad de isometría restringida (RIP) de orden k si existe un $\delta_k \in (0, 1)$ que

$$(1 - \delta_k) \parallel x \parallel_2^2 \quad \leq \quad \parallel \Phi x \parallel_2^2 \quad \leq \quad (1 + \delta_k) \parallel x \parallel_2^2 \tag{2.4.2}$$

Se mantiene para toda $x \in \Sigma_k$. Si una matriz Φ satisface la RIP de orden $2k$ entonces podemos interpretar la ecuación, diciendo que Φ aproximadamente preserva la distancia entre cualquier par de vectores k-dispersos. Esto claramente tiene implicaciones fundamentales en robustez al ruido. Finalmente, vemos que si una matriz satisface la RIP, entonces también satisface la propiedad de espacio nulo. Por lo tanto, la RIP es una condición más fuerte que la NSP.

2.4.3. Construcción de la matriz de sensado

Ahora que se han definido las propiedades que una matriz de sensado debe satisfacer, se presentan algunas técnicas de construcción de matrices de sensado. Para empezar una matriz Vandermonde V construida de m escalares distintos tiene una $Spark(V) = m+1$ [20]. Desafortunadamente, esas matrices están pobremente acondicionadas para grandes valores de n, haciendo el problema de la reconstrucción numéricamente inestable. También es posible construir matrices de manera determinista de tamaño $m\ x\ n$ que satisfagan la RIP de orden k, pero dichas construcciones requieren que m sea relativamente grande[22, 23, 24, 25]. Afortunadamente, esas limitaciones pueden sobrellevarse al construir matrices aleatorias. Usar matrices aleatorias Φ tiene muchos beneficios adicionales. Primero, uno puede ver que para construcciones aleatorias las mediciones son democráticas, es decir que es posible recuperar una señal usando cualquier subconjunto suficientemente largo de mediciones [26, 27]. Por lo tanto al usar matrices aleatorias, el sistema es más robusto a la pérdida de información o valores erráticos cuando se usa una pequeña fracción de mediciones. Segundo, en la práctica uno está más interesado en tomar una base Ψ donde x sea dispersa. En este caso lo que realmente queremos es que el producto de $\Phi\Psi$ satisfaga la RIP y si usamos una matriz aleatoria asegurándonos de tener una m suficientemente grande, la RIP se satisface con una alta probabilidad. Por ejemplo si usamos una matriz Φ con una distribución Gaussiana y Ψ es una base ortonormal, entonces fácilmente vemos que $\Phi\Psi$ será también una distribución Gaussiana. También vemos que obtenemos resultados similares cuando usamos matrices con distribuciones sub-gaussianas[28].
La parte mala de usar matrices aleatorias es que son matrices densas y por lo general ocupan mucho espacio en memoria.

2.5. Pseudoinversa de Moore-Penrose

La pseudoinversa $A^{\dagger}$ de una matriz A de tamaño $m\ x\ n$ es una generalización de la matriz inversa [29]. La pseudoinversa de A de tamaño $n\ x\ m$ deberá satisfacer los siguientes criterios:

1. $AA^{\dagger}A = A$

2. $A^{\dagger}AA^{\dagger} = A^{\dagger}$
3. $AA^{\dagger} = (AA^{\dagger})^T$ es una proyección ortogonal en R^m
4. $A^{\dagger}A = (A^{\dagger}A)^T$ es una proyección ortogonal en R^m

2.5.1. Propiedades

1. Si tenemos una matriz A de tamaño $m\ x\ n$ y sus columnas son linealmente independientes, tenemos que A^TA es invertible ya que es de rango n, la pseudoinversa A^+ se calcula de la siguiente forma:

$$A^{\dagger} = (A^TA)^{-1}A^T \tag{2.5.1}$$

Donde, $A^{\dagger}A = I$.

2. Si tenemos una matriz A de tamaño $m\ x\ n$ y sus filas son linealmente independientes, tenemos que AA^T es invertible ya que es de rango m. La pseudoinversa $A^{\dagger}$ se calcula de la siguiente forma:

$$A^{\dagger} = A^T(AA^T)^{-1} \tag{2.5.2}$$

Donde, $AA^{\dagger} = I$ [30].

Las siguientes propiedades son de especial ayuda para resolver la pseudoinversa de una matriz, ya que se limita el problema a resolver una matriz inversa de tamaño nxn.

2.6. Métodos de inversión de matrices

Una matriz cuadrada A de tamaño $n\ x\ n$ es invertible si existe una matriz cuadrada $n\ x\ n\ B$ tal que

$$AB = BA = I \tag{2.6.1}$$

Donde I denota la matriz identidad de tamaño $n\ x\ n$ y B es determinada a partir de A y es llamada la inversa de A. Una matriz que no es invertible es una matriz singular. Una matriz es singular cuando su determinante es 0 [31].
Como se vio en la sección anterior, debemos resolver el problema de la matriz

inversa dadas por las dos propiedades mencionadas. Se presentan algunos métodos de inversión de matrices a continuación.

2.6.1. Eliminación Gauss-Jordan

La eliminación Gaussiana es un algoritmo para resolver sistemas de ecuaciones lineales. Este método puede también ser usado para encontrar el rango de una matriz, calcular el determinante y para calcular la inversa de una matriz. Para resolver la matriz uno usa una secuencia de operaciones elementales en las filas hasta que obtengamos puros ceros en la matriz triangular inferior. Hay tres tipos de operaciones que podemos realizar:

1.- Intercambiar dos filas.

2.- Multiplicar una fila por un número diferente de cero.

3.- Sumar un múltiplo de una fila a otra fila.

Para encontrar la inversa de una matriz A, una variante de la eliminación Gaussiana puede ser usada. La llamada eliminación Gauss–Jordan busca solución para una matriz de tamaño nxn haciendo ceros las dos matrices triangulares [30]. Para encontrar la matriz inversa formamos una matriz de tamaño $nx2n$ al juntar la matriz A con una matriz identidad a la derecha. Buscamos entonces convertir el lado izquierdo a una matriz identidad, usando las operaciones mencionadas anteriormente, y el resultado o la matriz inversa, sería la matriz de nxn de lado derecho. Ejemplo:

$$[A|I] = \left[\begin{array}{ccc|ccc} 2 & 7 & 5 & 1 & 0 & 0 \\ 2 & 3 & 1 & 0 & 1 & 0 \\ 9 & 1 & 5 & 0 & 0 & 1 \end{array}\right].$$

Al hacer la eliminación de lado izquierdo:

$$[I|B] = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & \frac{-7}{52} & \frac{15}{52} & \frac{1}{13} \\ 0 & 1 & 0 & \frac{1}{104} & \frac{35}{104} & \frac{-1}{13} \\ 0 & 0 & 1 & \frac{25}{104} & \frac{-61}{104} & \frac{1}{13} \end{array}\right].$$

De la definición de una matriz inversa tenemos $BA = I$, y por lo tanto, $B = A^{-1}$. A la derecha tenemos $BI = B$, la cual es la matriz inversa.

2.6.2. Descomposición QR

En álgebra lineal, la descomposición QR de una matriz es una descomposición de una matriz A en un producto de dos matrices QR, donde Q es una matriz ortogonal y R una matriz triangular superior. Normalmente se usa para resolver el problema de mínimos cuadrados.
Comparado a obtener la matriz inversa directa, la solución a la inversa usando descomposición QR es numéricamente mas estable debido a su reducida condición numérica[32].
Hay varios metodos para calcular la descomposicion QR, una de ellas y quizas la mas usada es con el proceso de Gram-Schmidt[33].
Consideramos el proceso aplicado a las columnas de la matriz $A = [a_1, ..., a_n]$ de rango completo, y el producto punto $\langle v, w\rangle = v^T w$.
Se define la proyección como:

$$proj_e a = \frac{\langle e, a\rangle}{\langle e, e\rangle} e$$

obtenemos:

$$u_1 = a_1$$

$$u_2 = a_2 - proj_{u_1} a_2$$

$$u_k = a_k - \sum_{j=1}^{k-1} proj_{u_j} a_k$$

y

$$e_1 = \frac{u_1}{\|u_1\|}$$

$$e_2 = \frac{u_2}{\|u_2\|}$$

$$e_k = \frac{u_k}{\|u_k\|}$$

Re-expresamos ahora a las a_i sobre una nueva base ortonormal:

$$a_1 = \langle e_1, a_1\rangle e_1$$

$$a_2 = \langle e_1, a_2\rangle e_1 + \langle e_2, a_2\rangle e_2$$

$$a_k = \sum_{j=1}^{k} \langle e_j, a_k \rangle e_j$$

donde $\langle e_i, a_i \rangle = \|u_i\|$. En forma matricial:

$$A = QR$$

donde:

$$Q = [e_1, e_2, ..., e_n]$$

y

$$R = \begin{bmatrix} \langle e_1, a_1 \rangle & \langle e_1, a_2 \rangle & \cdots & \langle e_1, a_i \rangle \\ 0 & \langle e_2, a_2 \rangle & \cdots & \langle e_2, a_i \rangle \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \langle e_i, a_i \rangle \end{bmatrix}.$$

La inversa de una matriz cuadrada es fácil obtenerla debido a: $A = QR$ y por lo tanto $A^{-1} = R^{-1}Q^{-1} = R^{-1}Q^T$ y R^{-1} es fácil de obtenerla porque R es triangular. Para resolver un sistema lineal indeterminado $Ax = y$, donde la matriz A tiene dimension mxn y de rango m, primero se encuentra la descomposición QR de la transpuesta de A: $A^T = QR$, donde Q es una matriz ortogonal ($Q^T = Q^{-1}$), y R tiene la forma de $R = \begin{bmatrix} \mathbf{R}1 \\ 0 \end{bmatrix}$. $R1$ tiene forma cuadrada, es decir mxm y es una matriz triangular superior, la matriz de ceros tiene tamaño $(n-m)xm$. La solución a la inversa esta dada por $x = Q \begin{bmatrix} \{R1^T\}^{-1}y \\ 0 \end{bmatrix}$ y para resolver usamos sustitución directa o eliminación Gaussiana[29].

2.6.3. Descomposición Cholesky

En álgebra lineal, la descomposición Cholesky es la descomposición de una matriz hermitiana y definida positiva, en el producto de una matriz triangular inferior y su transpuesta conjugada.
Una matriz es definida positiva si para todos los vectores no nulos $z \in C^n$ tenemos que:

$$z^* M z > 0$$

La descomposición Cholesky de una matriz hermitiana definida positiva A, es una descomposición de la forma:

$$A = LL^*$$

donde L es una matriz triangular inferior con entradas positivas en la diagonal, y L^* es la transpuesta conjugada de L. Todas las matrices hermitianas definidas positivas tienen una descomposición Cholesky única[34].
Cuando A tiene puros valores reales, L tiene también valores reales y la descomposición puede escribirse como $A = LL^T$ [34]. Para calcular las matrices tenemos lo siguiente:

$$A = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}.$$

$$A = LL^T = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}.$$

Para la diagonal tenemos:

$$l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2}$$

Y para los elementos debajo de la diagonal(donde $i > k$):

$$l_{ik} = \frac{1}{l_{kk}}(a_{ik} - \sum_{j=1}^{k-1} l_{ij} l_{kj})$$

Para obtener la matriz inversa: $A^{-1} = (L^{-1})^T L^{-1}$.

2.6.4. Descomposición en valores propios(SVD)

La descomposición en valores propios de una matriz real o compleja A de tamaño mxn es una factorización de la forma $A = U\Sigma V^*$, donde U es una matriz real o compleja unitaria de tamaño mxm, Σ es una matriz rectangular de tamaño mxn con números reales no negativos en su diagonal y V es una matriz real o

compleja unitaria de tamaño nxn. Una matriz unitaria X es aquella en la cual su transpuesta conjugada X^* es también su inversa, es decir $X^*X = XX^* = I$. Los valores de la diagonal de Σ o σ_i son los eigen valores de M. Las columnas de U y las columnas de V son los eigen vectores izquierdos y eigen vectores derechos de M respectivamente[29].
Algunas aplicaciones que utilizan SVD es el computo de la pseudoinversa, problema de mínimos cuadrados, control multi-variable, aproximación de matrices, determinación del rango y espacio nulo de una matriz.

Para calcular la pseudoinversa de una matriz M usando SVD:

$$M^+ = V\Sigma^+U^*$$

donde Σ^+ es la pseudoinversa de Σ, la cual se forma al remplazar la diagonal por su recíproco y al trasponer la matriz resultante[35].

2.6.5. Método de Chebyshev iterativo

El método de Chebyshev nos permite obtener una secuencia de aproximaciones N_m para calcular la inversa de una matriz A[36], está dada por:

$$N_{m+1} = N_m(3I - AN_m(3I - AN_m)), \quad m = 0, 1... \tag{2.6.2}$$

o bien:

$$N_{m+1} = (3I - N_mA(3I - N_mA))N_m, \quad m = 0, 1... \tag{2.6.3}$$

Donde:

N_{m+1} es la aproximación siguiente.

N_m es la aproximación anterior.

I es la matriz identidad del tamaño de N_m

A es la matriz a ser invertida.

El método de Chebyshev muestra al menos una convergencia de orden tres para ecuaciones no lineales. Para que el método converja siempre con la misma

razón se presenta en [37] un valor inicial N_0 el cual esta dado por:

$$N_0 = \frac{A^T}{\| A \|_1 \| A \|_\infty}$$

Donde:

A^T es la matriz transpuesta de A.

$\| A \|_1$ es el valor máximo de entre todas las columnas de la sumatoria de todos sus valores.

$\| A \|_\infty$ es el valor máximo de entre todas las filas de la sumatoria de todos sus valores.

2.7. Algoritmos de reconstrucción

Pasamos ahora a la discutir algunos de los algoritmos para resolver el problema de la reconstrucción de señales en sensado comprimido. Hay una variedad de algoritmos que han sido usados en aplicaciones de aproximaciones dispersas, estadística, geofísica y ciencias de la computación teórica que fueron desarrollados para explotar la dispersidad de las señales y por lo tanto pueden ser utilizados en el problema de reconstrucción en sensado comprimido. Notese que los algoritmos aquí presentados son para reconstruir la señal x, sin embargo en algunas aplicaciones se utilizan estos para para resolver problemas de detección, clasificación o estimación de parámetros en donde todo caso, no es necesaria una reconstrucción completa de la señal[38, 39, 40, 41, 42, 43].

2.7.1. Minimización de la norma l_1

Dadas mediciones y y con el conocimiento que la señal original x es dispersa o compresible, es natural el querer recuperar x al resolver un problema de optimización de la forma:

$$\hat{x} = arg_z min \|z\|_0 \quad z \in B(y) \tag{2.7.1}$$

Donde $B(y)$ nos asegura que $\hat{x}$ es consistente con las mediciones y. Por ejemplo, en el caso donde las mediciones son exactas y sin ruido, tenemos $B(y) = \{z : Az = y\}$.

Cuando las mediciones han sido contaminadas con ruido, se puede considerar $B(y) = \{z : \|Az - y\|_2 \leq \epsilon\}$. En los dos casos, el problema de optimización busca la x mas dispersa que es consistente con las mediciones y. En el caso en que x no es dispersa, tomamos a $x = \Psi s$ y podemos modificar fácilmente la aproximación y considerar:

$$\hat{s} = arg_z min \|z\|_0 \quad z \in B(y) \tag{2.7.2}$$

Donde $B(y) = \{z : A\Psi z = y\}$ ó $B(y) = \{z : \|A\Psi z - y\|_2 \leq \epsilon\}$. En el caso en que $A = \Phi\Psi$ es exactamente lo mismo. Sin embargo no se sigue esta estrategia ya que la norma l_0 no es convexa y por lo tanto es difícil de resolver[44]. En vez de la norma l_0 se reemplaza por su aproximación convexa o la norma l_1. Específicamente se considera:

$$\hat{x} = arg_z min \|z\|_1 \quad z \in B(y). \tag{2.7.3}$$

Dado que $B(y)$ es convexo , es factible computacionalmente. El problema puede ser resuelto como un programa lineal[45].

2.7.2. Algoritmos greedy

Mientras que las técnicas de optimización convexas son métodos buenos para obtener representaciones dispersas, tambien hay una gran variedad de metodos "greedy" iterativos para resolver dichos problemas[46, 47, 48, 49, 50, 51]. Se les llama "greedy" por que muestran ciertas propiedades que pueden mejorar el desempeño de los algoritmos convexos en ciertas áreas, como la velocidad, el almacenamiento, facilidad de implementación, flexibilidad etc. Los algoritmos "greedy"se basan en la aproximación iterativa de los coeficientes de la señal y el soporte, también en identificar iterativamente el soporte de la señal hasta que se cumpla con un criterio de convergencia, ó alternativamente, al obtener un estimado de la señal dispersa en cada iteración que intente dar con el desajuste de la información medida.

Básicamente estos algoritmos tienen dos pasos fundamentales:

1.- Selección del elemento(Columna de la matriz de sensado).

2.- Actualización de los coeficientes.

Los algoritmos greedy se dividen en dos tipos:

Greedy pursuits

Normalmente se inician con un estimado de la señal igual a cero $\hat{x}^{[0]} = 0$. Con esta inicialización, el error residual es $r^{[0]} = y - A\hat{x}^{[0]} = y$ y el soporte (indices de los elementos no cero) de la primera estimación $\hat{x}^{[0]}$ es $T = \emptyset$. En cada iteración se actualizan esas cantidades al agregar elementos adicionales (columnas del diccionario) al soporte T y al actualizar el estimado de la señal $\hat{x}$, se decrementa el error residual r.
En la tabla 2.1 se muestra el marco general para los algoritmos greedy.

Tabla 2.1: Marco general de algoritmo greedy

Marco general greedy pursuits.
Entrada: y,A,k
for: i=1; i:=i+1 hasta que se cumpla un criterio de parada
Calcular $g^{[i]} = A^T r^{[i]}$ y seleccionar columnas de A en base a la magnitud de los elementos de $g^{[i]}$.
Calcular el estimado para $\hat{x}^{[i]}$ (y por lo tanto $\hat{y}^{[i]}$) al decrementar la función de costo: $F(\hat{x}^{[i]}) = \|y - A\hat{x}^{[i]}\|_2^2$
end for
Salida: $r^{[i]}$ y $\hat{x}^{[i]}$

Uno de los algoritmos greedy persuits mas utilizados es el Orthogonal matching pursuit (OMP por sus siglas en ingles), el cual aproxima a x en cada iteración al proyectar y ortogonalmente en las columnas de A asociadas con el soporte actual calculado $T^{[i]}$. Se puede ver el algoritmo OMP en la tabla 2.2.

Thresholding algorithms

En general los algoritmos "greedy persuits" son fáciles de implementar y pueden ser muy rápidos. Sin embargo , no ofrecen garantías de reconstrucción tan fuerte como la de los algoritmos convexos. Para solucionar ese problema se utilizan los algoritmos de tipo de umbral ó "thresholding algorithms" en ingles. Siguen

Tabla 2.2: Pseudocodigo de algoritmo OMP

Orthogonal matching pursuit.

Entrada: y,A,k
Iniciar: $r^{[0]} = y$, $\hat{x}^{[0]} = 0$, $T^{[0]}$
for: $i = 1$; $i := i + 1$ hasta que se cumpla un criterio de parada

$g^{[i]} = A^T r^{[i-1]}$
$j^{[i]} = argmax_j |g_j^{[i]}| / \|A_j\|_2$
$T^{[i]} = T^{[i-1]} \cup j^{[i]}$
$\hat{x}^{[i]} = A_T^{\dagger} y$
$r^{[i]} = y - A\hat{x}^{[i]}$

end for
Salida: $r^{[i]}$ y $\hat{x}^{[i]}$

siendo relativamente fáciles de implementar y pueden ser extremadamente rápidos.
En la tabla 2.3 se presenta el algoritmo "Compressive Sampling Matching Persuit"(CoSaMP) utilizado en esta tesis, el cual fue introducido por Needell y Tropp [52]. El algoritmo es inicializado con una aproximación trivial. En cada iteración, el algoritmo realiza 5 pasos principales:

1.- Se calcula el producto punto del residuo con la matriz de sensado y se identifican los componentes mas grandes.

2.- El conjunto de elementos encontrados se unen con el conjunto de componentes que aparecen en la aproximación actual.

3.- El algoritmo soluciona el problema de mínimos cuadrados para aproximar la señal con el soporte unido.

4.- El algoritmo produce una nueva aproximación al retener solamente los elementos mas grandes de la aproximación de mínimos cuadrados.

5.- Las muestras son actualizadas y se calcula el residuo.

Tabla 2.3: Pseudocodigo de algoritmo CoSaMP

Compressive Sampling Matching Persuit.

Entrada: y,A,k
Iniciar: $r^{[0]} = y$, $\hat{x}^{[0]} = 0$
for: $i = 1$; $i := i + 1$ hasta que se cumpla un criterio de parada

$g^{[i]} = A^T r$
$\Omega = supp(g_{2k}^{[i]})$
$T^{[i]} = \Omega \cup supp(\hat{x}^{[i-1]})$
$b|_T = A_T^{\dagger} y$
$b|_T^c = 0$
$\hat{x}^{[i]} = b_k$
$r^{[i]} = y - A\hat{x}^{[i]}$

end for
Salida: $r^{[i]}$ y $\hat{x}^{[i]}$

2.7.3. Algoritmos combinacionales

En adición a los algoritmos greedy y a la minimización de la norma l_1, existe otra clase importante de algoritmos de recuperación de señales dispersas, llamados algoritmos combinacionales. Estos algoritmos fueron desarrollados en el contexto de pruebas de grupo combinacional[53, 54]. En este problema suponemos que hay n elementos totales y k elementos anómalos que deseamos encontrar. Nuestra meta es diseñar un grupo de pruebas que nos permitan identificar el soporte(y posiblemente el valor) de x mientras minimizamos el número de las pruebas realizadas. En la práctica normalmente esas pruebas son representadas por una matriz binaria A en la cual sus entradas a_{ij} son igual a 1 si y sólo si el elemento j-ésimo es utilizado en la i-ésima prueba. Si la salida es lineal con respecto a las entradas, entonces el problema de recuperar el vector x es esencialmente el mismo que el que resolvemos en sensado comprimido. Algunas ventajas que muestran estos tipos de algoritmos es que el almacenamiento necesario es reducido, ya que las matrices de sensado son matrices no densas[55]. La velocidad de los algoritmos generalmente es mas rápida que los presentados anteriormente.

2.8. Trabajos relacionados

Los autores de [56] ejecutaron un algoritmo de reconstrucción(OMP) en MATLAB en un procesador Intel Core Duo a 2.8GHz. Les tomó 606μS reconstruir una señal de longitud 128 y dispersidad k=5. En [57] proponen un método de sensado comprimido para una arquitectura de varios núcleos. Utilizan un procesador Intel i7 a 3GHz para reconstruir señales de longitud 512 y dispersidad k=12, logran una precisión de 1,24$e-03$ y el tiempo de reconstrucción es de 25mS. En el artículo [58] proponen una arquitectura en paralelo del algoritmo de reconstrucción OMP basado en la GPU GTX480 de NVIDIA. Para reconstruir una señal de longitud 8192 con dispersidad k=32, le toma al algoritmo 32mS. Sin embargo, el consumo de potencia es elevado. En [59] se propone una arquitectura en paralelo en un FPGA Kintex-7, le toma 39.9μS xk reconstruir una señal. Además, el trabajo de reconstrucción también es ejecutado en una CPU y una GPU. Alternativamente en [60] se presenta una arquitectura basada en un FPGA Virtex-4 de una versión optimizada del algoritmo CoSaMP para reconstruir señales con dispersidad k=2, utilizando un diccionario de tamaño 32x255. Los autores de [61] presentan una implementación optimizada del algoritmo OMP con un tamaño de diccionario de 32x128 con nivel de dispersidad k=5 al cual le toma 24μS reconstruir una señal. En [62] se presentan dos algoritmos de reconstrucción (OMP y AMP), utilizan diccionarios de tamaño 256x1024 con un nivel de dispersidad k=12 a una frecuencia de 100MHz, presentan resultados de implementación en un FPGA Virtex-6. En la tabla 2.4 se presentan los resultados de la implementación del algoritmo OMP. En el artículo [63] los autores presentan una arquitectura basada en el algoritmo OMP con un tamaño de diccionario 256x1024 y un nivel de dispersidad $k = 36$. El modelo que proponen lo programan en MATLAB utilizando System Generator de Xilinx. El tiempo de reconstrucción de la señal es de 443μS. Finalmente en el trabajo [64] realizan reconstrucción de señales en un FPGA Virtex-7 con tamaños de diccionarios 32x128 y 512x2048 utilizando una memoria ram externa. Manejan niveles de dispersidad k=5 y k=12 respectivamente y se alcanza una frecuencia máxima de operación de 165MHz el tiempo de reconstrucción para k=5 es de 18μS.

En la tabla 2.4 se resumen algunos de los trabajos presentados anteriormente en sensado comprimido.

Tabla 2.4: Trabajos realizados en sensado comprimido

Trabajos en Sensado Comprimido						
Arquitectura	***Tamaño del diccionario***	***Dispersidad***	***Frecuencia de operación***	***Precision***	***Tipo de datos***	***Tiempo Reconstrucción***
Intel Core Duo[56]	32x128	5	2.8GHz	-	32 bits fixed point real data	$606\mu S$
Intel i7[57]	64x512	12	3.0GHz	1.24e-03	Single precision complex data	252mS
NVIDIA GTX480[58]	512x8192	64	-	-	-	15mS
Kintex-7 FPGA[59]	640x1470	<320	53.7MHz	-	Single precision real data	$39.9\mu S$ x k
Virtex-4 FPGA[60]	32x255	2	-	-	24 bits fixed point real data	-
Virtex-5 FPGA[61]	32x128	5	39MHz	-	32 bits fixed point real data	$24\mu S$
Virtex-6 FPGA[62]	256x1024	12	100MHz	-	18 bits fixed point real data	$158.7\mu S$
Virtex-6 FPGA[63]	256x1024	36	92MHz	-	18 bits fixed point real data	$443\mu S$
Virtex-7 FPGA[64]	32x128	5	165MHz	1.2e-03	Single precision complex data	$18.3\mu S$

Capítulo 3
Metodología

En este capítulo se presenta la metodología para poder completar este trabajo de tesis. Se revisan las herramientas utilizadas, y se da una revisión al algoritmo que se utilizó.

3.1. Síntesis de alto nivel

La síntesis de alto nivel o también llamado síntesis a nivel de comportamiento y arquitectura, es una tecnología de diseño para realizar sistemas de cualquier tipo enfocados a sistemas embebidos, procesadores y multi procesadores. Las ventajas de la síntesis de alto nivel se debe a que se dejan los detalles de la implementación a los algoritmos y las herramientas, incluyendo la habilidad de determinar el tiempo necesario para realizar una operación, la transferencia de datos y el almacenamiento. La síntesis de alto nivel promete ser una de las soluciones para lidiar con el significante incremento en la demanda por la productividad en el diseño mas allá del estado del arte. También nos da la posibilidad de explorar el espacio de diseño de forma eficiente al tratar con una abstracción mas elevada y una rápida implementación para probar la factibilidad de los algoritmos y permitir una optimización del desempeño del mismo.

Se utiliza la herramienta VIVADO-HLS como interfaz y el algoritmo se programa en lenguaje C++.

En la figura 3.1 se presenta la metodología seguida en este trabajo. En cuanto al uso de directivas se siguió la propuesta en [65], en donde para acelerar el proceso de multiplicación de matrices hay que desenrollar el ciclo interno completamente y aplicar Pipeline II (1) al ciclo que viene después de este. Para poder hacer esto, hay

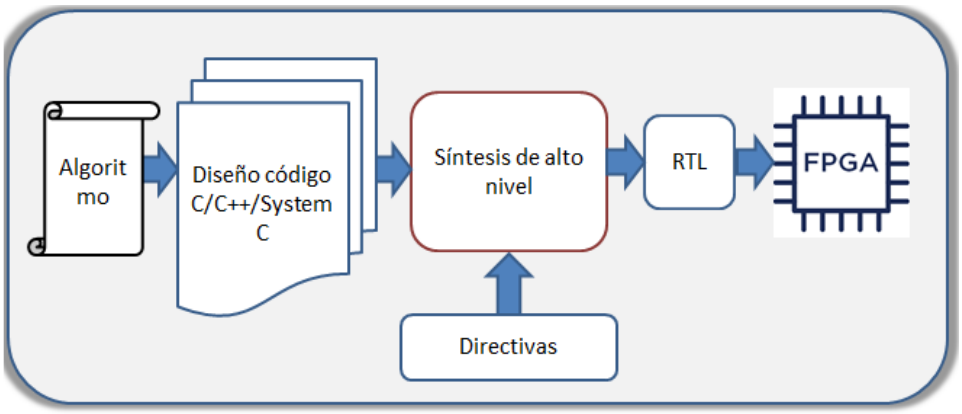

Figura 3.1: Diagrama de flujo de metodología.

que particionar los vectores utilizados, pudiendo usar un particionado completo o parcial. El esfuerzo del sintetizador por encontrar la solución al sistema planteado se dejó en default(medio), al configurarse en alto el tiempo por corrida aumenta demasiado.

3.2. Aplicación en FPGA

Hacer la implementación en FPGA tiene muchas ventajas, algunas son: bajo consumo de potencia, ejecución del código en paralelo, velocidad, etc. Se utiliza un FPGA Virtex-7 (XC7VX690T) para observar resultados de latencia y recursos utilizados en las diferentes etapas del algoritmo CoSaMP. Se decide utilizar single precision para compararse con otros trabajos propuestos, en los cuales no solo aplican FPGA's sino también GPUs y procesadores Intel. También, se trabaja con números complejos ya que se decide utilizar la transformada de Fourier como base dispersa.

3.3. Análisis del algoritmo CoSaMP

A continuación se hace un análisis del algoritmo presentado en 2.3 para luego hacer la síntesis de este.

Después de obtener las muestras y, el diccionario A y el nivel de dispersidad de la señal k, iniciamos el residuo r igual a las muestras obtenidas y. El primer paso del algoritmo es hacer la multiplicación del diccionario A por el residuo r

es decir $g = A^T r$, la cual devolverá un vector de tamaño n. Una vez obtenido el vector de tamaño n, es necesario obtener el soporte de la misma. El soporte puede ser obtenido de dos formas:

1.- Utilizando un algoritmo de ordenamiento.

2.- Obteniendo el k-máximo del proxy.

Cada una tiene sus ventajas y desventajas. Para utilizar un algoritmo de ordenamiento, uno busca que sea lo mas eficiente posible, por lo tanto se buscan algoritmos de ordenamiento que sean paralelos. La arquitectura de estos cuestan demasiados recursos y pueden ser demasiados rápidos. Si se utiliza un algoritmo menos demandante en recursos, la latencia es inexacta y pueden ser muy lentos. Para el segundo es básicamente hacer $n - 1$ comparaciones por el nivel de dispersidad k o pudiendo tomar también $2k$, lo cual también es un gasto fuerte en recursos pero menor que al hacer un acomodo de todos los datos. El soporte se obtiene para poder elegir las columnas de la matriz de sensado con las cuales se hace la recuperación de la señal. La unión del soporte actual con la del soporte anterior es una de las partes mas importantes del algoritmo y es algo por lo cual algoritmo tiene mejor garantía de reconstrucción que la de los "greedy pursuits". En la primera aproximación uno solo cuenta con el soporte obtenido cuando se forma el "proxy"de la señal, después de formar la señal dispersa se guardan los coeficientes. A partir de la segunda iteración se vuelve a calcular el "proxy"de la señal y se obtiene el soporte nuevo; ese soporte es ahora unido con el soporte anterior para formar un soporte de tamaño $2k$. Después viene el calculo de b que será un arreglo de tamaño k para la primera iteración y $2k$ a partir de la segunda. El vector b se puede calcular de diferentes formas, dependerá de que tantos recursos, rapidez y exactitud uno esté buscando, pudiendo usar métodos iterativos o métodos exactos. La señal reconstruida se formara a partir de los valores k valores mas grandes obtenidos y de los valores del soporte T que resultaron. Esos mismo soporte de x se guarda y es utilizado en la unión del siguiente soporte T. Finalmente se actualiza el residuo r haciendo $y - Ax^{[i]}$. En la tabla 3.1 se presentan los pasos necesarios para implementar el algoritmo.

Entrada:
Muestras: y
Matriz de sensado: A
Nivel de dispersidad: k
Iniciar:
Residuo r=y
Señal dispersa x =0
Durante la primera iteración
$g^{[1]} = A^T r$ (Formación del proxy)
$\Omega = supp(g_k^{[1]})$ (Soporte)
$b\|_\Omega = A_\Omega^\dagger y$ (Solución al problema)
$b\|_\Omega^c = 0$
$\hat{x}^{[1]} = b_k$ (Formar señal dispersa)
$r^{[i]} = y - A\hat{x}^{[i]}$ (Actualizar el residuo)
A partir de la segunda iteración
$g^{[i]} = A^T r$(Formación del proxy)
$\Omega = supp(g_k^{[i]})$(Soporte)
$T^{[i]} = \Omega \cup supp(\hat{x}^{[i-1]})$ (Unión del soporte actual con el anterior)
$b\|_T = A_T^\dagger y$ (Solución al problema)
$b\|_T^c = 0$
$\hat{x}^{[i]} = b_k$ (Formar señal dispersa y guardar el soporte)
$r^{[i]} = y - A\hat{x}^{[i]}$(Actualizar el residuo)
Salida:
Señal dispersa $\hat{x}$
Residuo r

Tabla 3.1: Descripción del algoritmo CoSaMP

3.3.1. Modificaciones al algoritmo original

Como se pudo apreciar la única modificación que se le hizo al algoritmo presentado en 2.3 fue utilizar un soporte Ω de tamaño k en vez de $2k$, durante las pruebas no se presento mayor cambio el utilizar un soporte de menor tamaño. El soporte unido T cambia de tamaño $3k$ a $2k$ sin mayor problema ya que al final solo se terminan agarrando k coeficientes para la reconstrucción de la señal. Para una implementación en FPGA es conveniente la reducción del tamaño del soporte de la señal ya que también se reducen las operaciones realizadas, mejorando la latencia total del algoritmo y la utilización de recursos.

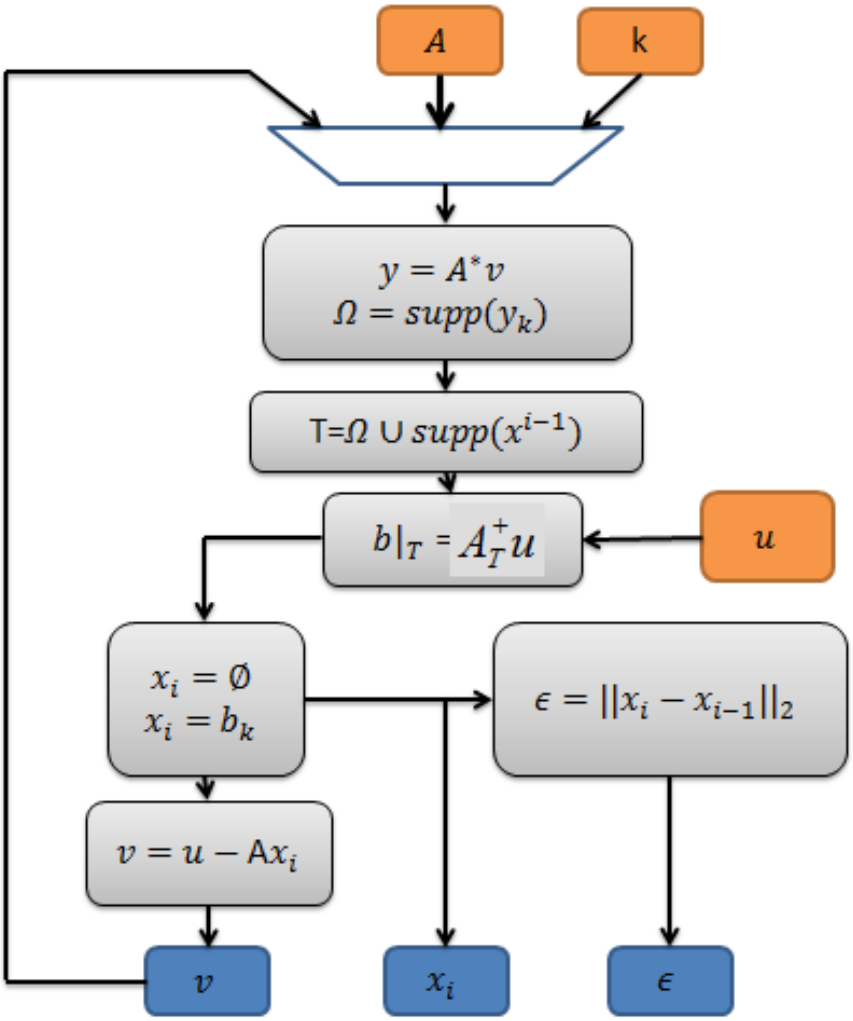

Figura 3.2: Diagrama de flujo algoritmo CoSaMP.

Capítulo 4

Resultados

En esta sección se presentan los resultados obtenidos al simular el algoritmo CoSaMP en un FPGA.
Para las pruebas se utilizó un FPGA Virtex-7 XC7VX690T a una frecuencia de 165MHz. Los resultados mostrados es utilizando "single precision", pudiendo mejorar el desempeño del algoritmo utilizando punto fijo.
Los resultados principales se obtuvieron utilizando una señal de tamaño $n = 128$ a una razón de compresión de 4 ó $m = 32$ a diferente nivel de dispersidad.

4.1. Soporte de la señal

En el algoritmo se implementó la búsqueda del soporte por medio de comparaciones para encontrar el k-máximo valor. Una vez encontrado, se guardan los coeficientes que resultaron dentro de esas comparaciones. La utilización de recursos depende del tamaño n de la señal y del nivel de dispersidad. En la tabla 4.1 se muestran los recursos necesarios para el cálculo del soporte a diferentes niveles de dispersidad y un tamaño de vector n=128.

Tabla 4.1: Recursos para el cálculo del soporte para un vector de tamaño 128, para diferentes niveles de dispersidad

Dispersidad	BRAM_18K	DSP48E	FF	LUTs
2	2	0	1473	9176
3	2	0	2322	12728
4	2	0	3593	14844
5	2	0	4902	19196
6	2	0	5017	21525
7	2	0	6757	24875

4.2. Operaciones con matrices

Al trabajar con este algoritmo es imprescindible el trabajar con operaciones matriciales. En esta sección se muestran resultados de latencia al realizar multiplicaciones entre matrices.

Como paso inicial del algoritmo CoSaMP es formar un *proxy* al multiplicar el diccionario y el residuo de la señal $y = |A^T r|$. El numero de operaciones a realizar depende del tamaño de la señal reconstruida(n) y del tamaño de la señal comprimida(m). En la figura 4.1 se muestra la latencia necesaria para diferente tamaño de señales reconstruidas a diferente razón de compresión.

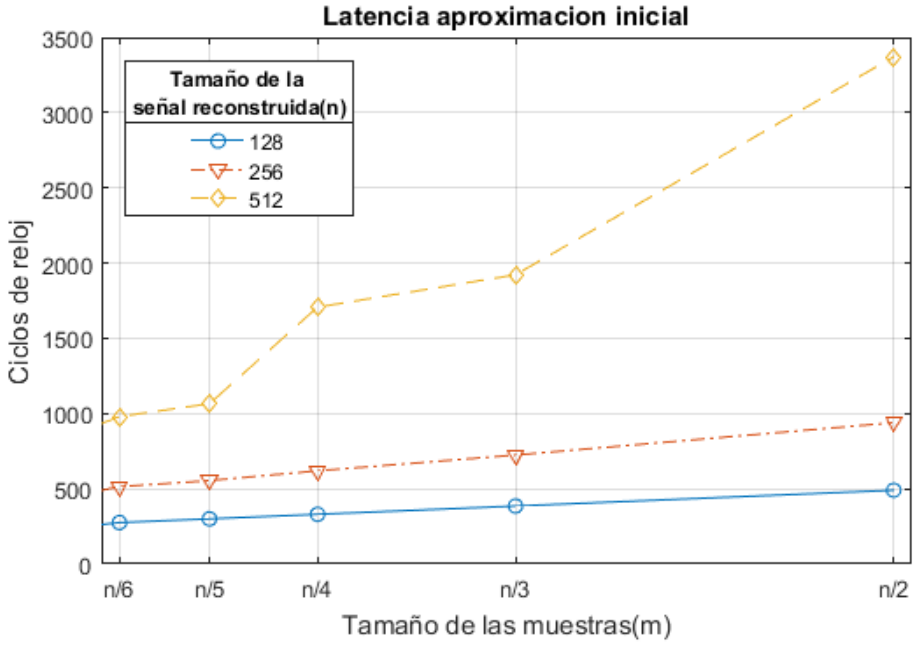

Figura 4.1: Latencia aproximación inicial.

Para vectores de longitud mayor hay que tomar en cuenta que la utilización de recursos es muy alta, por lo tanto, hay un intercambio entre velocidad contra recursos utilizados. Es necesario comprimir la señal al máximo posible si se requiere optimizar los recursos del FPGA a utilizar. En el caso del vector de longitud 512, la latencia se eleva, ya que se utilizó diferente intervalo de iniciación de pipeline para que el uso de recursos no excediera los limites del FPGA(es notorio en el cambio de pendiente). Por lo tanto, para vectores grandes se tienen que emplear otras estrategias. Una forma podría ser descomponer los vectores en otros mas pequeños

y hacerlo por partes. La utilización de recursos para una señal a reconstruir de 128 datos a diferente razón de compresión se muestra en la tabla 4.2.

Tabla 4.2: Utilización de recursos para aproximación inicial con un vector de 128 datos, usando m=n/2,n/4,n/8.

Tamaño del vector de muestras	DSP48E	FF	LUT
64	1291	109172	94057
32	651	56276	48620
16	331	29828	25905

Para poder calcular la pseudoinversa de una matriz A hacemos uso de la ecuación (2.5.1). Por lo que se requiere resolver el producto $A^T A$ y así obtener una matriz simétrica. El número de operaciones dependerá del nivel de dispersidad de la señal y del tamaño del vector de muestras(m). Se obtiene en la primera iteración una matriz de tamaño kxk donde k es el nivel de dispersidad y a partir de la segunda iteración del algoritmo, tomará un valor de $(k-2k)$x$(k-2k)$. En la figura 4.2 se muestra la latencia requerida para calcular dicha matriz a diferentes tamaños de vectores de muestras y nivel de dispersidad(o tamaño de la matriz resultante). En la tabla 4.3 se muestran los recursos necesarios para calcular la matriz utilizando una señal de 128 datos y un vector de muestras de 32 datos.

Tabla 4.3: Recursos calculo $A^T A$

Dispersidad	DSP48E	FF	LUT
2	160	15417	11388
4	160	15425	11417
8	160	15433	11445
16	160	15441	11475
32	160	13400	11504

Paso siguiente es encontrar la inversa de esa matriz(lo cual se explicará en la siguiente sección) y finalmente realizar el producto de la matriz inversa por A^T para obtener la pseudoinversa($A^\dagger$). La pseudoinversa tendrá un tamaño de kxm para la primera iteración del algoritmo y un tamaño de $(k-2k)$xm a partir de la segunda iteración. La latencia para calcular el producto de la matriz inversa por la transpuesta de A se muestra en la figura 4.3 y los recursos utilizados en la tabla 4.4.

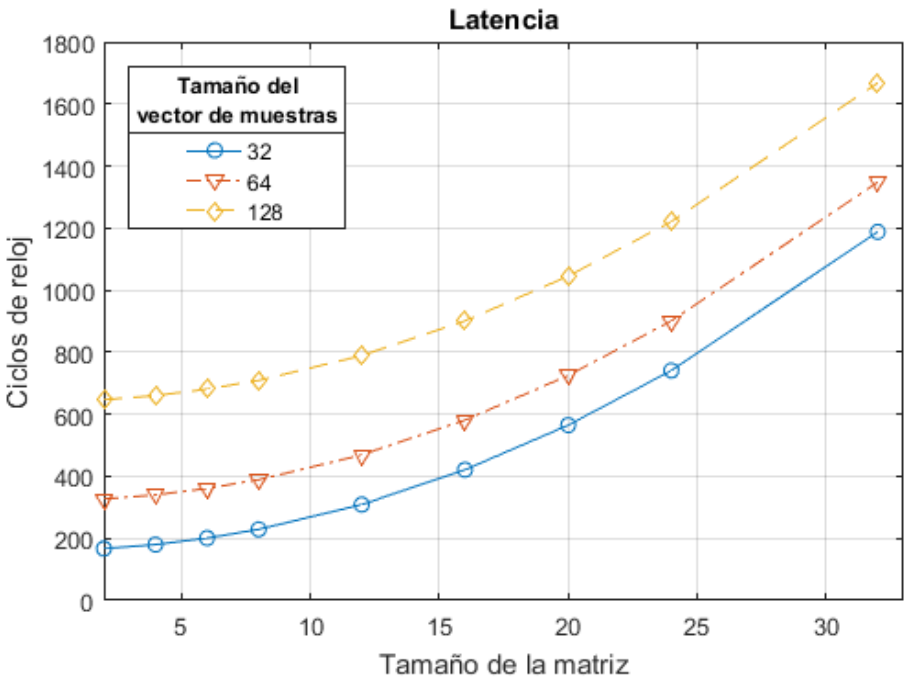

Figura 4.2: Latencia producto matriz transpuesta por si misma.

Tabla 4.4: Recursos utilizados para el producto de la matriz inversa por la transpuesta de A, para un vector de n=128 y m=32.

Dispersidad	DSP48E	FF	LUT
2	10	945	770
4	20	1847	1485
8	40	3647	2925
16	80	7243	5782
32	160	13400	11504

Para calcular la matriz inversa se realizan multiplicaciones con matrices cuadradas de tamaño kxk para la primera iteración y $(k-2k)$x$(k-2k)$ a partir de la segunda iteración del algoritmo. La latencia obtenida para diferentes tamaños de matrices se muestran en la figura 4.4 y los recursos utilizados se muestran en la tabla 4.5.

Una vez calculada la matriz pseudoinversa se procede a resolver el problema de mínimos cuadrados. En la primera iteración obtenemos un vector de tamaño k a partir de la segunda iteración obtendremos un vector en el intervalo de k a

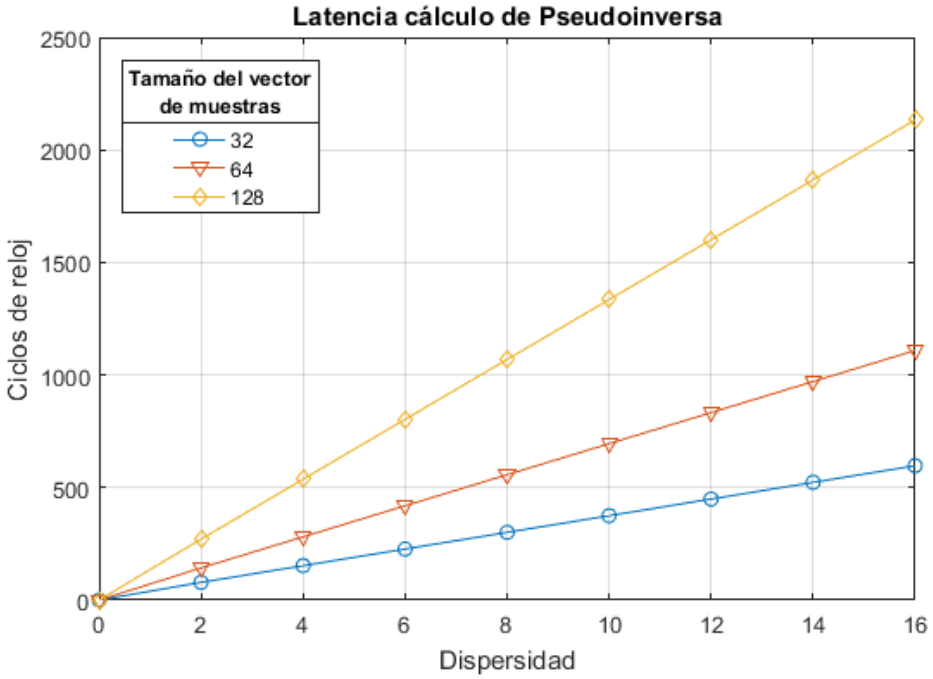

Figura 4.3: Latencia para el calculo del producto de la inversa por la transpuesta de A para diferentes tamaños de vectores de muestras.

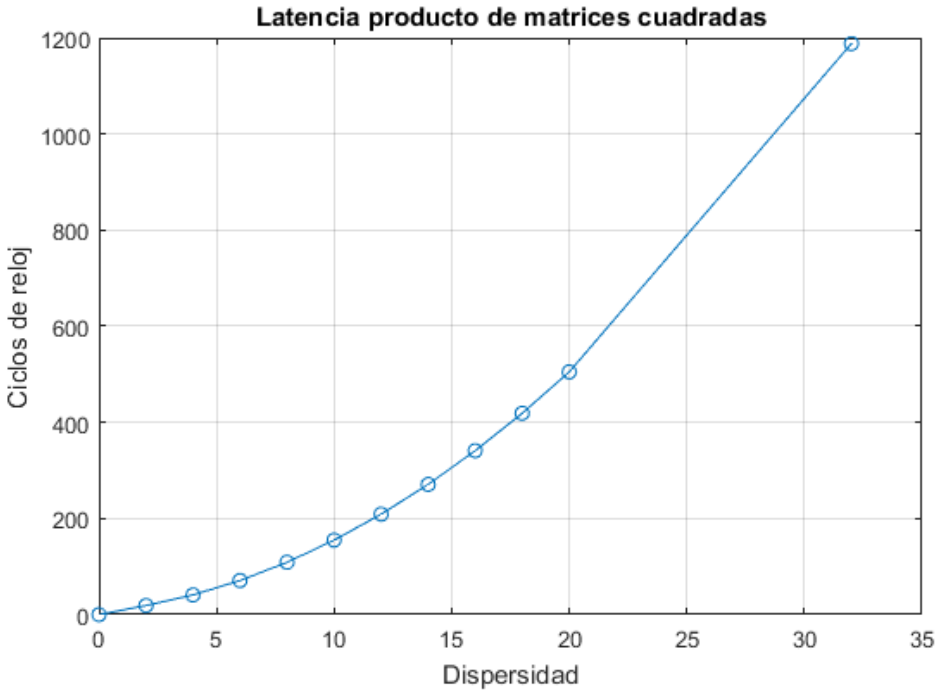

Figura 4.4: Latencia del producto entre matrices cuadradas.

Tabla 4.5: Recursos para producto de matrices cuadradas.

Dispersidad	DSP48E	FF	LUT
2	10	987	740
4	20	1957	1463
8	40	3889	2905
16	80	7745	5771
32	160	13400	11504

$2k$. El tiempo necesario para resolver el problema se muestra en la figura 4.5, la utilización de recursos para un vector de 32 datos se presenta en la tabla 4.6.

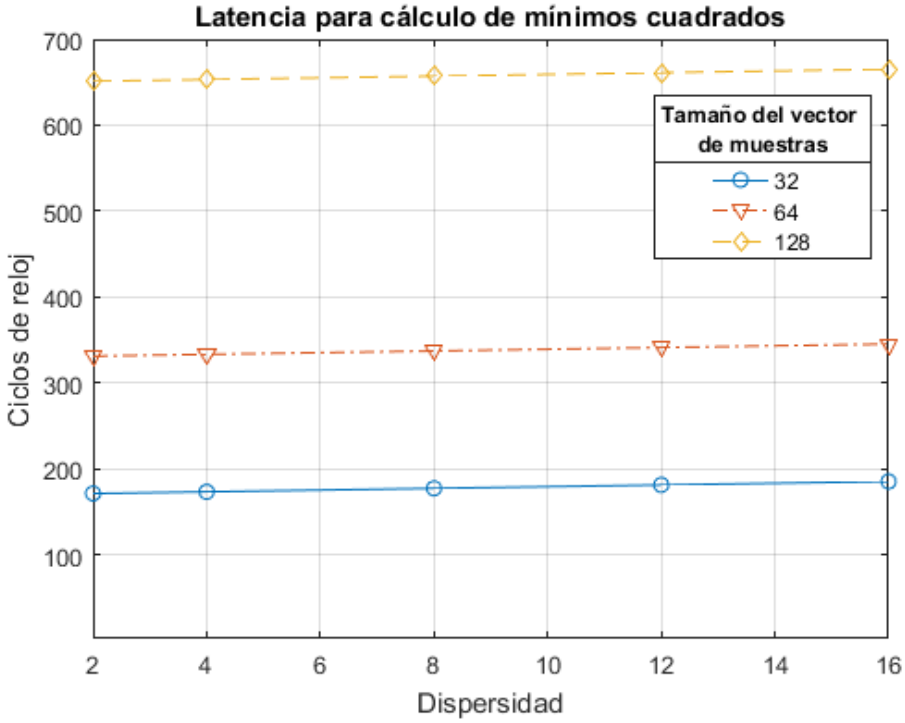

Figura 4.5: Latencia de mínimos cuadrados.

Tabla 4.6: Recursos utilizados para resolución de mínimos cuadrados para una señal con m=32.

Dispersidad	DSP48E	FF	LUT
2	640	54963	51290
4	640	54965	51298
8	640	54967	51306
16	640	54969	51315

Después de haber resuelto el problema de mínimos cuadrados y de haber formado la señal dispersa, hay que actualizar el vector de muestras. La latencia calculada para diferentes tamaños de vectores de muestras se muestra en la figura 4.6. Los recursos necesarios para actualizar un vector de muestras de tamaño 32 se muestran en la tabla 4.7.

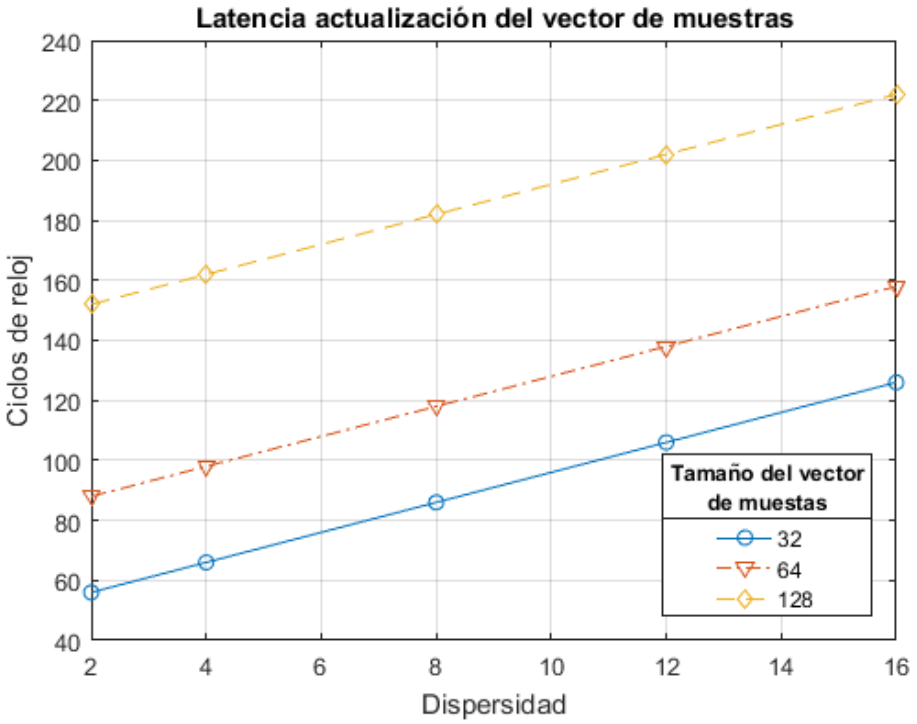

Figura 4.6: Latencia para actualizar el vector de muestras de tamaño m.

Tabla 4.7: Recursos para realizar la actualización del vector de muestras de tamaño m=32.

Dispersidad	DSP48E	FF	LUT
2	44	4003	3983
4	84	7437	7013
8	164	14305	13061
16	324	28041	25675

4.3. Inversión de matrices

Se dedica una sección al método de inversión de matrices ya que es la parte que mas recursos y latencia producen del algoritmo. Se presentan 3 métodos de

inversión de matrices:

1.- Método iterativo de Chebyshev.

2.- Descomposición QR.

3.- Descomposición Cholesky.

Para medir la latencia en la inversión de matrices, se utilizaron matrices aleatorias de diferente tamaño.

4.3.1. Método iterativo de Chebyshev

Para resolver la matriz inversa de una matriz A con el método de Chebyshev, es necesario resolver la ecuación anteriormente descrita en (2.6.2). Como es un método iterativo es necesario definir una forma de paro. Una de ellas es medir el error cuadrático medio comparando la medición anterior con la actual, otra es simplemente definir un número de iteraciones fijo, el cual se definirá mediante mediciones del error hechas con anterioridad. En la figura 4.7 se presentan algunas pruebas hechas con matrices generadas aleatoriamente, teniendo la particularidad de ser matrices simétricas. Se observa como el error se va reduciendo conforme el numero de iteraciones aumenta. Para matrices grandes o al utilizar una dispersidad elevada, el numero de iteraciones para reducir el error será mayor.

El uso de recursos para el cálculo de la matriz inversa a diferente nivel de dispersidad se muestra en la tabla 4.8.

Tabla 4.8: Recursos utilizados para el cálculo de la matriz inversa.

Dispersidad	BRAM_18K	DSP48E	FF	LUT
2	0	37	6963	9510
4	0	71	20680	27853
8	96	99	50796	74755
16	192	195	75176	126732

La latencia de la matriz inversa obtenida mediante este método dependerá de la precision que se necesite. Por lo tanto, se presenta la figura 4.8 con la latencia por cada iteración a diferente nivel de dispersidad.

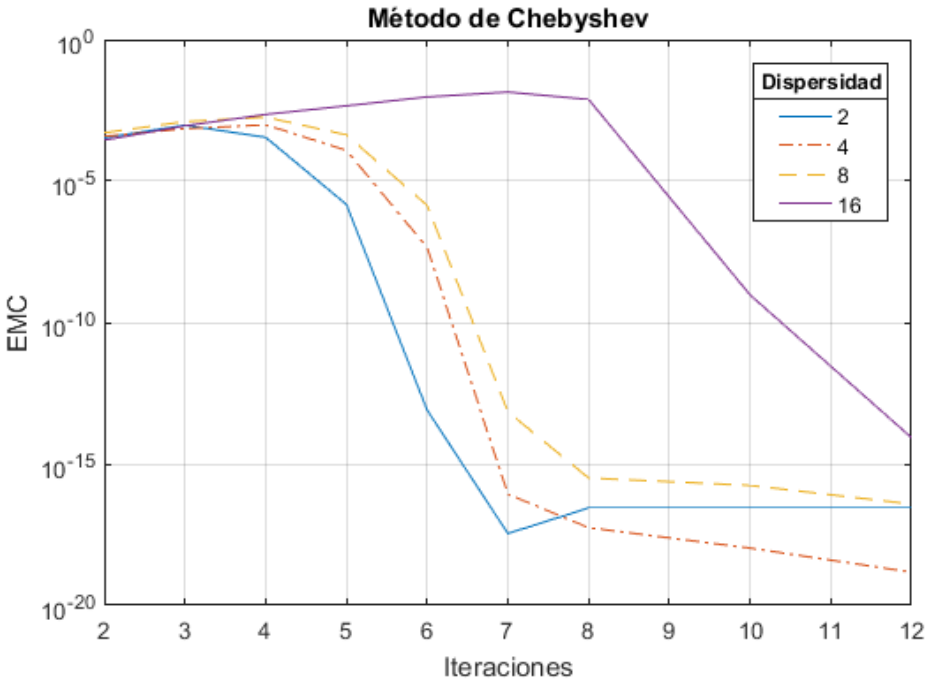

Figura 4.7: Reducción del error cuadrático medio para el método de Chebyshev.

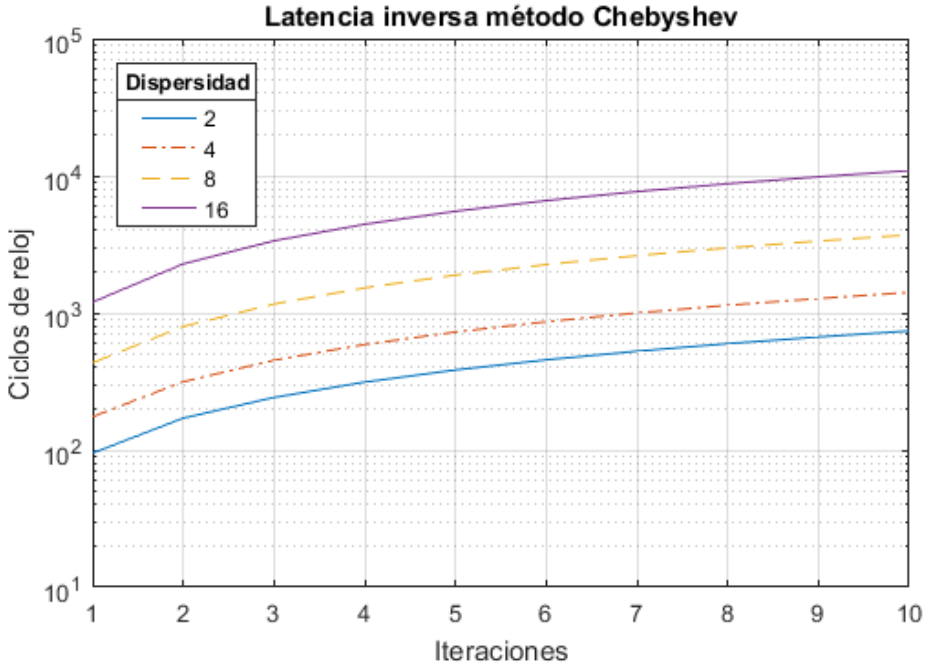

Figura 4.8: Latencia para el método iterativo de Chebyshev.

4.3.2. Descomposición QR

Otro método de inversión de matrices es la descomposición o factorización QR. Xilinx tiene soporte en álgebra lineal con una librería en la cual incluyen la factorización QR de matrices. En ella también se incluye el algoritmo para obtener la inversa de una matriz cuadrada mediante el mismo método. Es un método exacto, por lo tanto nos ahorra el problema de calcular el error y puede ser mas rápido que el método de Chebyshev si se necesita una precision elevada a niveles bajos de dispersidad.

Se muestran resultados de latencia en la figura 4.9 y de utilización de recursos en la tabla 4.9.

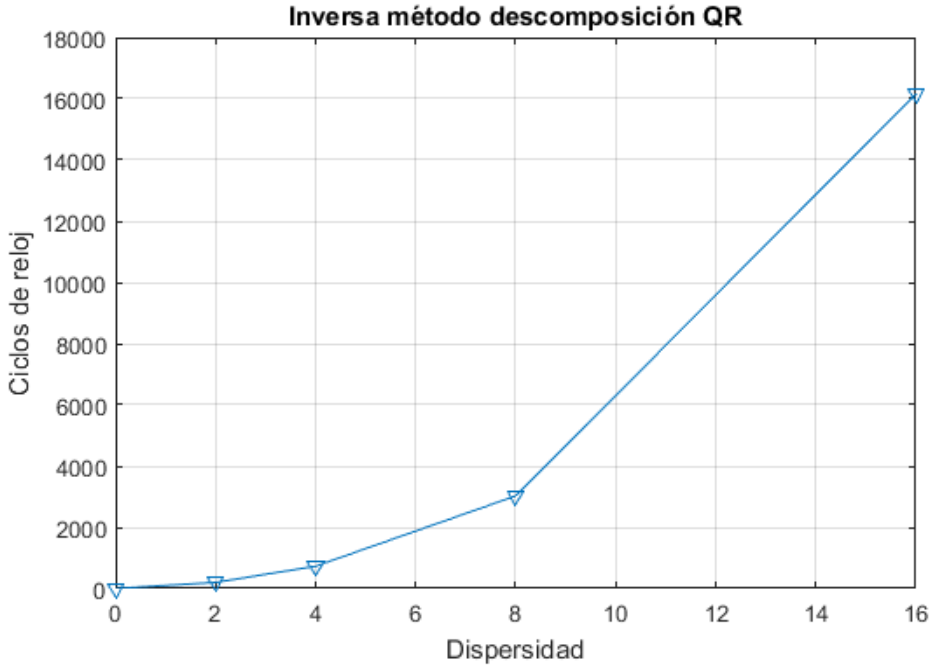

Figura 4.9: Latencia para calcular la inversa con el método de descomposición QR

Tabla 4.9: Recursos para el método QR

Dispersidad	BRAM_18K	DSP48E	FF	LUT
2	0	50	9507	12710
4	6	37	8160	9007
8	10	34	7704	8577
16	10	34	7836	8820

4.3.3. Descomposición Cholesky

Se obtuvo también la inversa de la matriz utilizando la descomposición o factorización Cholesky utilizando la librería de álgebra lineal de Xilinx. Utiliza menos recursos que el método QR. Si se requieren implementaciones de señales de baja dispersidad, es recomendable usar este método. En la figura 4.10 se muestra el comportamiento que se obtuvo en latencia para diferentes niveles de dispersidad.

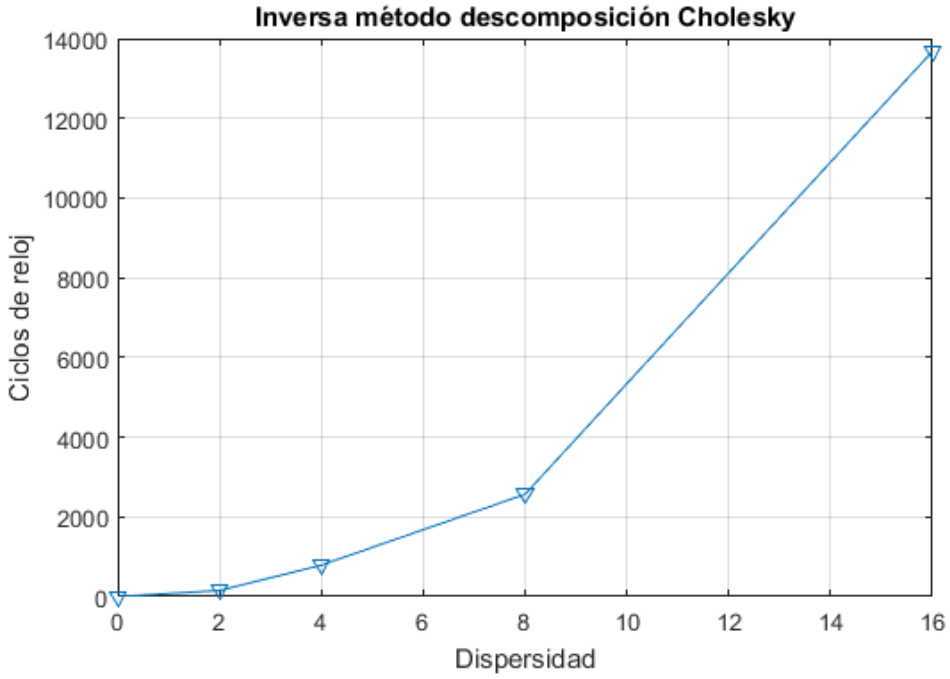

Figura 4.10: Latencia para calcular la inversa con el método de descomposición Cholesky

En la tabla 4.10 se muestran los recursos obtenidos para resolver la matriz inversa con el método de descomposición Cholesky.

Tabla 4.10: Recursos para obtener la matriz inversa mediante la descomposición Cholesky.

Dispersidad	BRAM_18K	DSP48E	FF	LUT
2	0	24	3645	5359
4	2	29	3856	4876
8	7	32	4227	4729
16	7	32	4565	4887

4.4. Latencia total de algoritmo CoSaMP

En esta sección se presenta la latencia que se obtuvo al reconstruir la señal con los diferentes métodos explicados anteriormente. Para la obtención de resultados se utilizaron las siguientes señales generadas en MATLAB utilizando single precision:

1.- $1 + cos(2\pi t)$.

2.- $1 + cos(2\pi t) + cos(4\pi t)$.

3.- $1 + cos(2\pi t) + cos(4\pi t) + cos(6\pi t)$.

4.- $1 + cos(2\pi t) + cos(4\pi t) + cos(6\pi t) + cos(8\pi t)$

5.- $1 + cos(2\pi t) + cos(4\pi t) + cos(6\pi t) + cos(8\pi t) + cos(10\pi t)$.

6.- $1 + cos(2\pi t) + cos(4\pi t) + cos(6\pi t) + cos(8\pi t) + cos(10\pi t) + cos(12\pi t)$.

Se utilizaron señales de tamaño n=128 que es exactamente un período de la señal. El muestreo de la señal se hizo utilizando un diccionario aleatorio generado en MATLAB y la transformada de Fourier discreta como base.

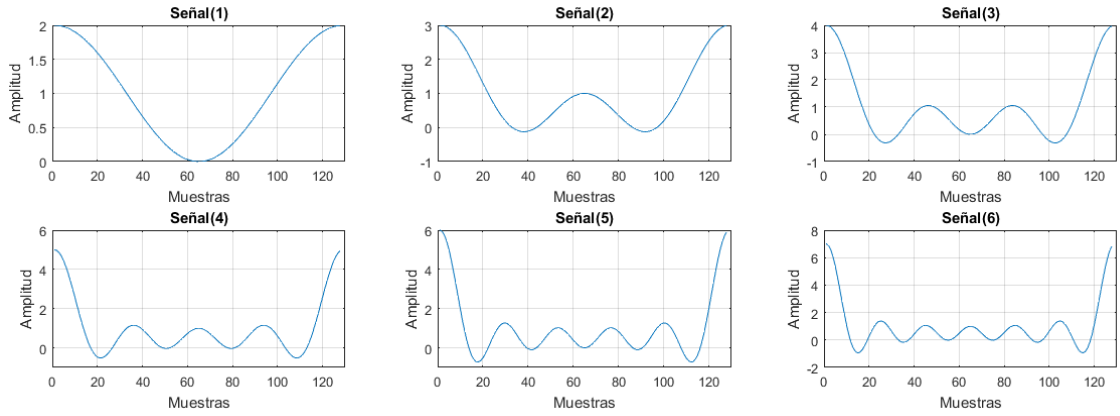

Figura 4.11: Señales prueba

Como el espectro de Fourier es simétrico solo es necesario tomar la mitad de este y hacer cero el resto de la señal. En la figura 4.12 se muestra la representación dispersa de las señales prueba en el dominio de Fourier.

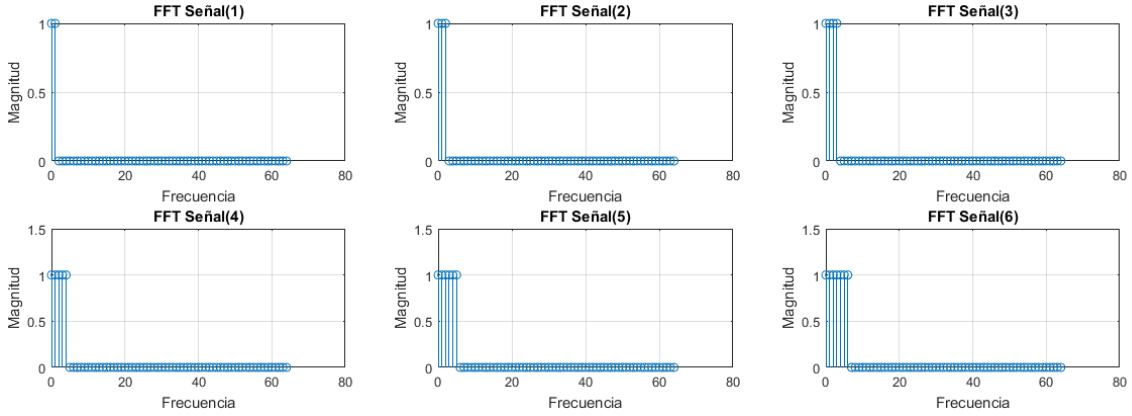

Figura 4.12: Representación dispersa de las señales prueba en el dominio de Fourier.

Tabla 4.11: Recursos para algoritmo CoSaMP utilizando el método de Chebyshev.

Método	*Chebyshev*			
Dispersidad	**BRAM_18K**	**DSP48E**	**FF**	**LUT**
2	532	747	130458	214037
3	420	747	113020	180613
4	448	747	128609	232477
5	482	747	161041	257485
6	526	747	143025	233980
7	538	747	164690	284985

Tabla 4.12: Recursos para algoritmo CoSaMP utilizando el método descomposición Cholesky.

Método	*Cholesky*			
Dispersidad	**BRAM_18K**	**DSP48E**	**FF**	**LUT**
2	286	803	126071	291420
3	339	822	115904	241414
4	342	848	125930	281584
5	343	872	117037	201923
6	356	788	113677	211322
7	357	799	119535	230030

En las tablas 4.11, 4.12 y 4.13 se muestran los recursos necesarios para que el algoritmo CoSaMP se pudiera ejecutar a una frecuencia de 165MHz.

En la figura 4.13 se muestra la latencia obtenida para las diferentes arquitecturas para cada nivel de dispersidad. Para el método de Chebyshev se utilizó un error menor a 10^{-10} pudiendo mejorar la latencia modificando el error. Las señales

Tabla 4.13: Recursos para algoritmo CoSaMP utilizando el método descomposición QR.

Método	*QR*			
Dispersidad	BRAM_18K	DSP48E	FF	LUT
2	290	837	136297	300704
3	346	849	125622	247062
4	350	864	136496	288818
5	350	876	124891	209600
6	362	815	122259	219821
7	364	815	128292	239185

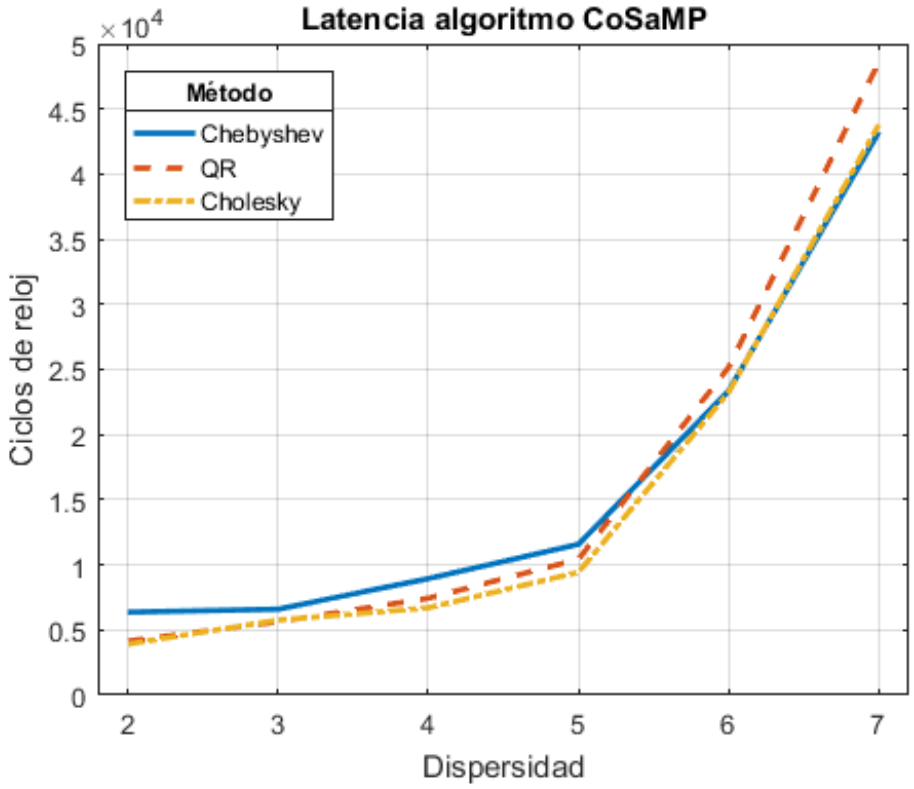

Figura 4.13: Latencia del algoritmo CoSaMP con las diferentes arquitecturas a diferente nivel de dispersidad.

de dispersidad del 2 al 5($k = 2, 3, 4, 5$) se pudieron reconstruir con dos iteraciones del algoritmo, la señal con dispersidad $k = 6$ se reconstruyó con tres iteraciones y la señal con nivel de dispersidad $k = 7$ hicieron falta 4 iteraciones.

4.5. Análisis de reconstrucciones

4.5.1. Señales y muestras ideales

Se muestran las señales reconstruidas en la figura 4.14 y el error obtenido en la tabla 4.14 de cada una de ellas. Cabe decir que la señal reconstruida es de tamaño n=128, en la figura se muestra solo la mitad del espectro de Fourier por ser simétrica y para que la frecuencia de la señal sea vista exactamente en la gráfica. El error cuadrático medio, se calculó a partir del vector de tamaño 128.

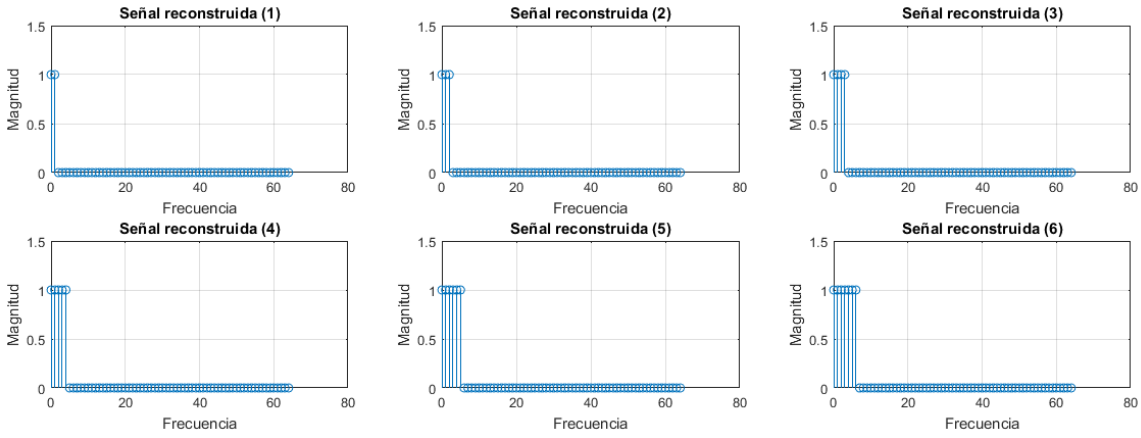

Figura 4.14: Señal dispersa reconstruida utilizando los diferentes métodos.

Tabla 4.14: Error cuadrático medio normalizado de la señal reconstruida n=128.

NMSE			
Dispersidad	***Chebyshev***	***QR***	***Cholesky***
2	7.3304e-12	5.0762e-12	7.0054e-12
3	6.9128e-12	8.8432e-12	6.9511e-12
4	1.7570e-11	1.0840e-11	1.1453e-11
5	1.2030e-11	1.8979e-11	1.2685e-11
6	1.3503e-11	7.7536e-12	1.1237e-11
7	2.9802e-12	3.4603e-12	3.6098e-12

4.5.2. Análisis de ruido

El sistema gana robustez al ruido al no comprimir demasiado la señal que se va reconstruir. En las pruebas mientras mas grande era el vector de muestras, el sistema era mas robusto al ruido. También, el sistema es mas robusto al ruido si se usa un nivel de dispersidad k suficientemente grande. El error también depende del diccionario utilizado; por lo tanto, cuando se vaya a utilizar un diccionario generado aleatoriamente, se recomienda hacer simulaciones de reconstrucciones con señales ruidosas. En esta tesis el vector de muestras es de tamaño m=32. Se optó por reducir en gran parte el numero de muestras. Para generar las muestras con ruido, se utilizo el comando awgn en MATLAB para inducir ruido blanco gaussiano a las muestras generadas anteriormente. Se utilizó el mismo diccionario y el mismo número de iteraciones(2,3 y 4) para reconstruir las señales de dispersidad k=5, k=6 y k=7. Se mide el error cuadrático medio normalizado para cada caso y cada método. En las tablas 4.15, 4.16 y 4.17 se muestran los resultados obtenidos.

Tabla 4.15: Error cuadrático medio normalizado de la señal reconstruida con k=5.

SNR (dB)	*Chebyshev*	*Cholesky*	*QR*
20	0.0139	0.0139	0.0139
30	0.0017	0.0017	0.0017
40	3.6820e-04	3.6821e-04	3.6820e-04

Tabla 4.16: Error cuadrático medio normalizado de la señal reconstruida con k=6.

SNR (dB)	*Chebyshev*	*Cholesky*	*QR*
20	0.0281	0.0281	0.0281
30	0.0055	0.0055	0.0055
40	1.1836e-04	1.1834e-04	1.1834e-04

Tabla 4.17: Error cuadrático medio normalizado de la señal reconstruida con k=7.

SNR (dB)	*Chebyshev*	*Cholesky*	*QR*
20	0.0488	0.0586	0.0586
30	0.0040	0.0040	0.0040
40	8.6602e-05	8.6544e-05	8.6562e-05

En las figuras 4.15, 4.16 y 4.17 se muestran las reconstrucciones que se hicieron

con el ruido que se le agregó a las muestras tomadas de la señal a reconstruir con nivel de dispersidad $k = 5$, $k = 6$ y $k = 7$ respectivamente.

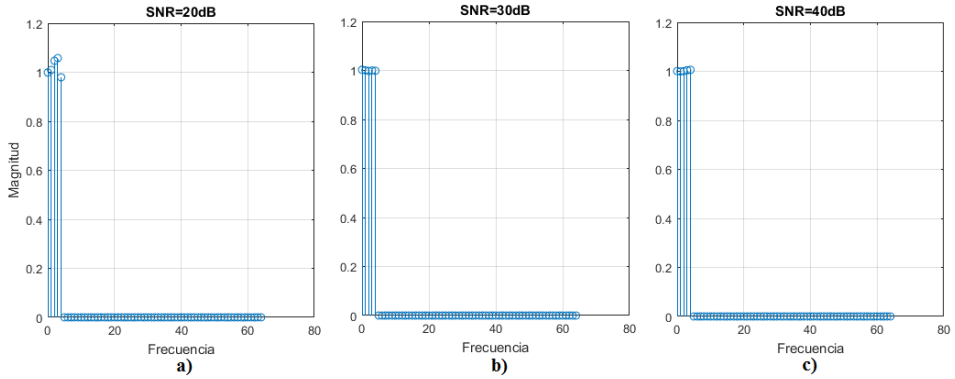

Figura 4.15: Señal reconstruida con nivel de dispersidad k=5 con: a)SNR=20dB, b) SNR=30dB y c) SNR=40dB.

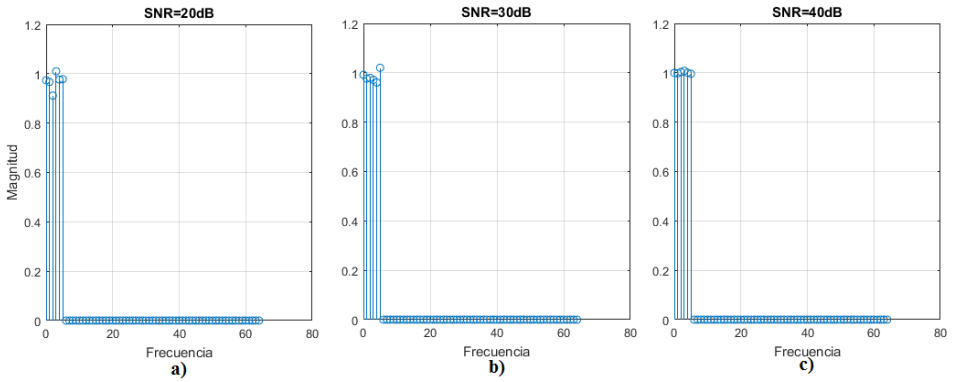

Figura 4.16: Señal reconstruida con nivel de dispersidad k=6 con: a)SNR=20dB, b) SNR=30dB y c) SNR=40dB.

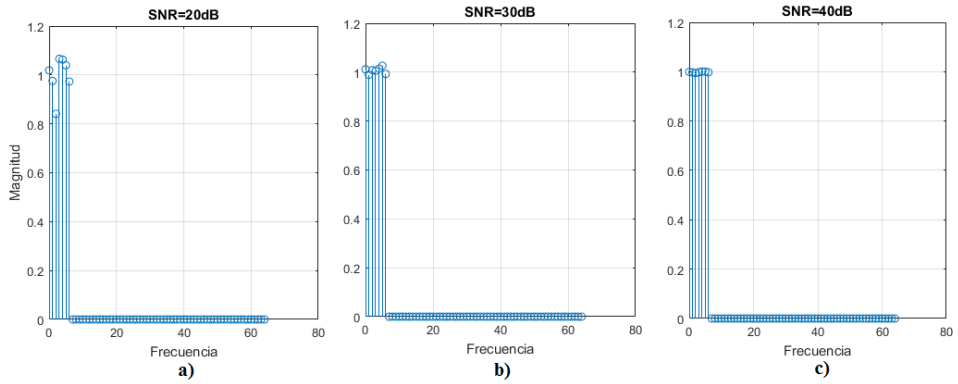

Figura 4.17: Señal reconstruida con nivel de dispersidad k=7 con: a)SNR=20dB, b) SNR=30dB y c) SNR=40dB.

En las figuras 4.18, 4.19 y 4.20 se muestra el vector de muestras(y) contaminado con ruido utilizando diferente SNRs.

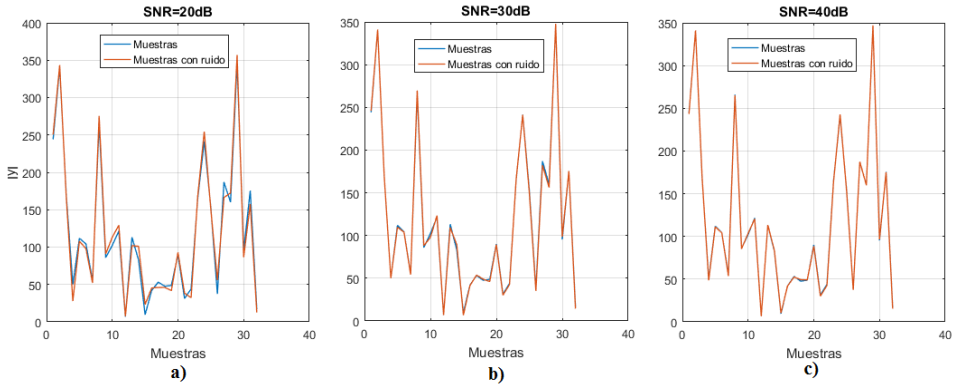

Figura 4.18: Arreglo de muestras(y) para la señal dispersa con $k = 5$ contaminado con ruido: a)SNR=20dB, b) SNR=30dB y c) SNR=40dB.

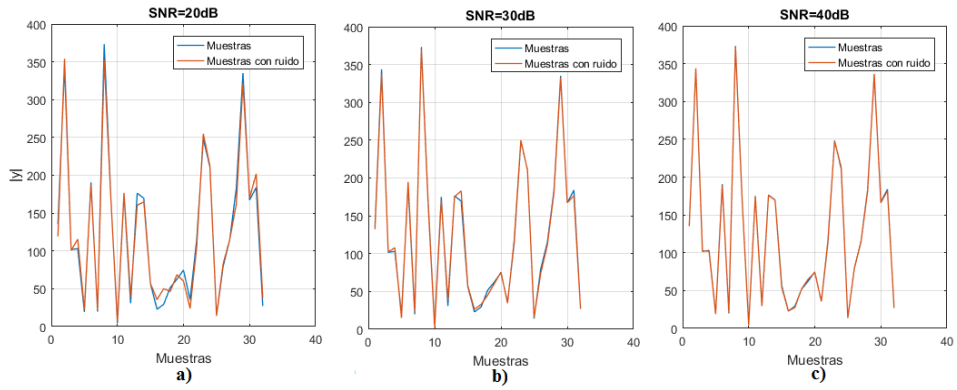

Figura 4.19: Arreglo de muestras(y) para la señal dispersa con $k = 6$ contaminado con ruido: a)SNR=20dB, b) SNR=30dB y c) SNR=40dB.

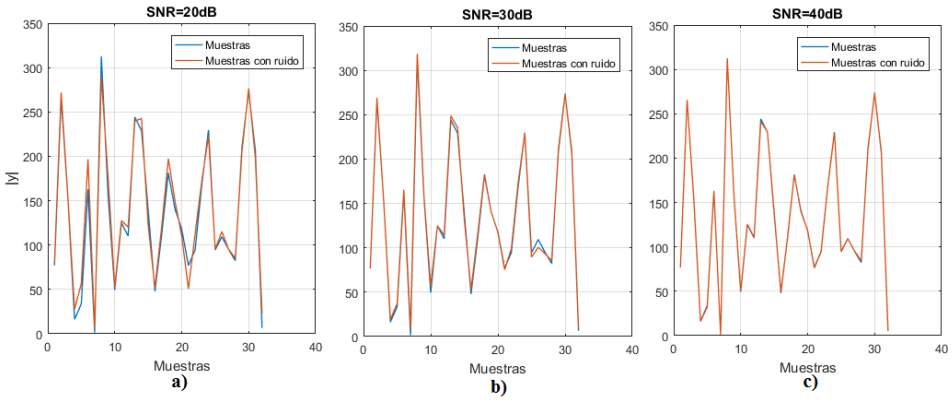

Figura 4.20: Arreglo de muestras(y) para la señal dispersa con $k = 7$ contaminado con ruido: a)SNR=20dB, b) SNR=30dB y c) SNR=40dB.

4.6. Transformada inversa de Fourier discreta rápida

Si además se requiere recuperar la señal en el dominio original o no disperso, se presentan resultados de latencia y de utilización de recursos utilizando el algoritmo de la IFFT(Inverse Fast Fourier Transform) y producto punto. En la figura 4.21 se muestra la latencia total necesaria para obtener la transformada inversa de Fourier de vectores de diferente longitud.

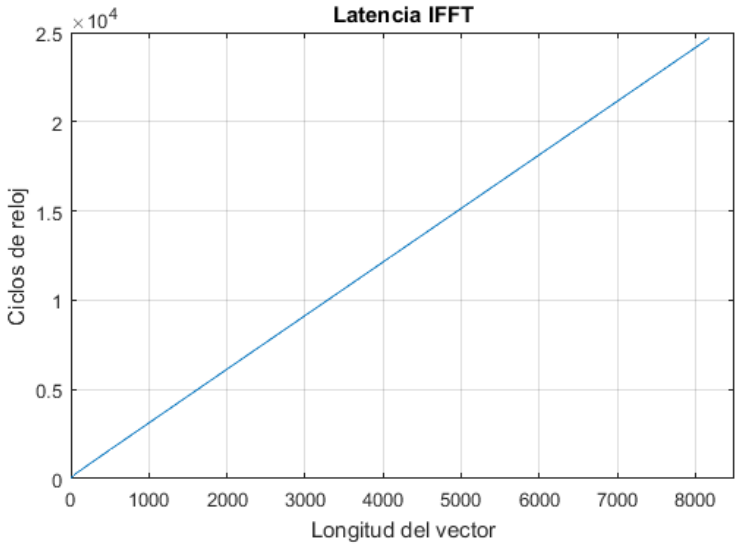

Figura 4.21: Latencia Inverse Fast Fourier Transform.

En la tesis, se utilizaron vectores de tamaño 128. Una vista mas selectiva de la misma gráfica se visualiza en la figura 4.22, la latencia exacta para un vector de longitud 128 fue de 484, que para un reloj de 165MHz, corresponde a 2.93μS. El uso de recursos de la FFT se muestra en la tabla 4.18. Para hacer la arquitectura de la IFFT se utilizo streaming de datos con pipeline utilizando LUTS, teniendo también la opción de utilizar DSPs y una arquitectura tipo radix-2 o radix-4.

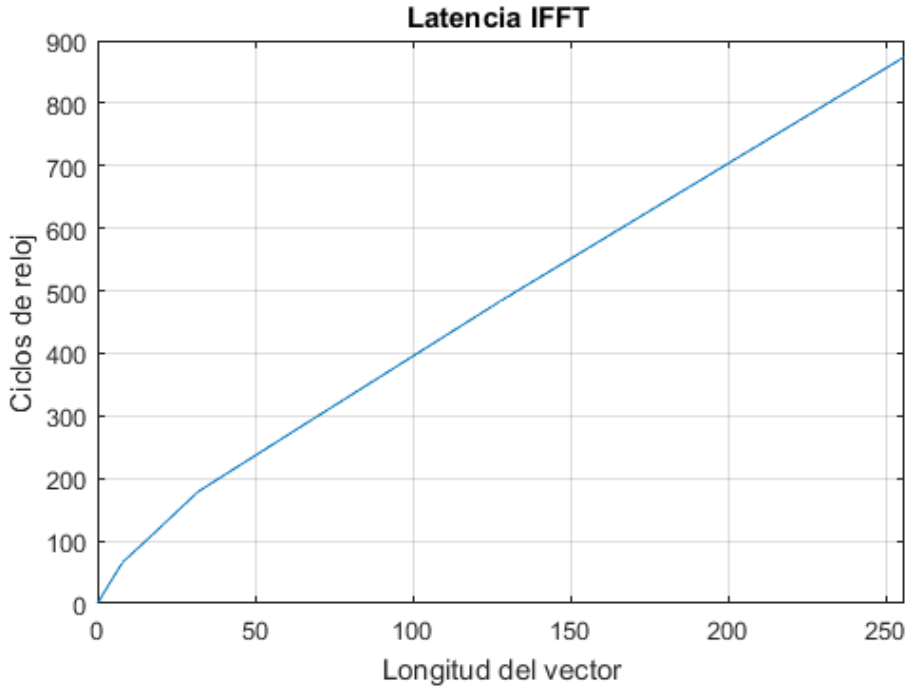

Figura 4.22: Vista de latencia Inverse Fast Fourier Transform.

Tabla 4.18: Recursos utilizados IFFT

Tamaño del vector	BRAM_18K	DSP48E	FF	LUT
8	4	12	5291	4037
16	12	12	5441	3997
32	12	24	7252	5527
64	14	24	9065	7083
128	**14**	**36**	**10878**	**8571**
256	14	36	12697	10043
512	15	48	14512	11593
1024	19	48	16336	13194
2048	37	60	18247	15081
4096	72	60	20178	17156

Se hace la comparación en rapidez y en recursos utilizados teniendo en cuenta un diccionario con la transformada inversa de Fourier. En la figura 4.23 se muestra la latencia obtenida.

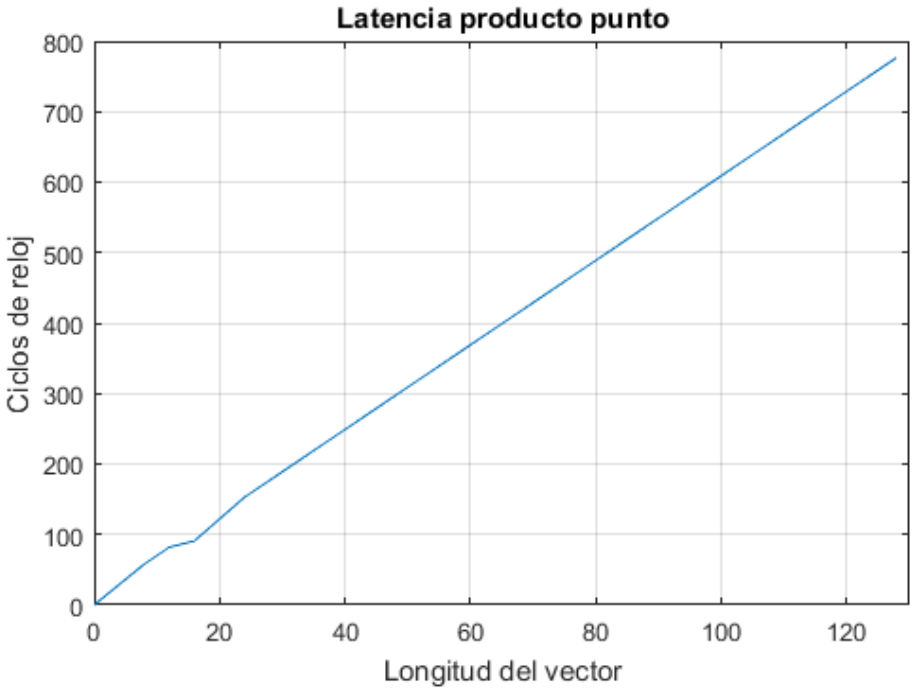

Figura 4.23: Latencia utilizando producto punto.

Sin embargo calcular la transformada inversa de Fourier con este método no es recomendable ya que si bien uno alcanza una velocidad comparable a la IFFT con longitud pequeña de vectores, la utilización de recursos es muy alta. En la tabla 4.19 se muestra la cantidad de recursos necesarios para calcular una transformada inversa de Fourier de longitud 128 a una velocidad comparable a la de la IFFT. La latencia de la transformada fue de 777 ciclos de reloj, es decir, 4,70μS utilizando un reloj de 165MHz.

Tabla 4.19: Utilización recursos de una transformada inversa de Fourier discreta(longitud 128), utilizando producto punto.

Tamaño del vector	BRAM_18K	DSP48E	FF	LUT
128	2054	2560	219955	181869
Utilización(%)	**69**	**71**	**25**	**41**

4.6.1. IFFT para recuperación de la señal continua

A continuación se presentan las transformadas de Fourier inversas de las señales que fueron reconstruidas en la sección 4.5. En la figura 4.24 se muestran las señales obtenidas después de aplicar el algoritmo de la IFFT a las reconstrucciones sin muestras ruidosas.

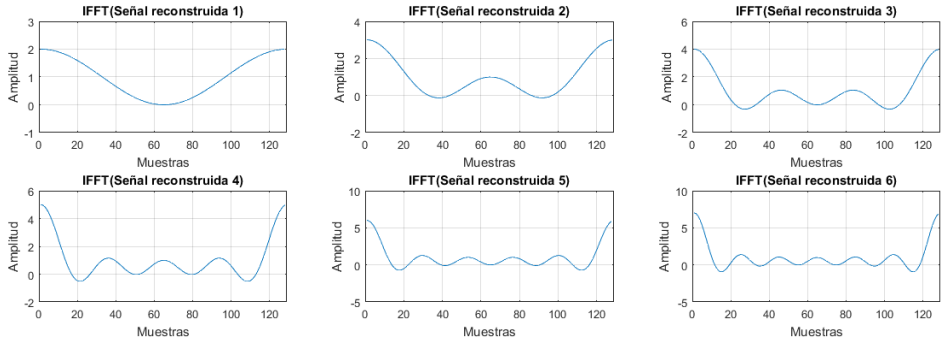

Figura 4.24: IFFT de las señales reconstruidas.

En las figuras 4.25, 4.26 y 4.27 se muestran las señales reconstruidas con diferente nivel de dispersidad k y nivel de ruido.

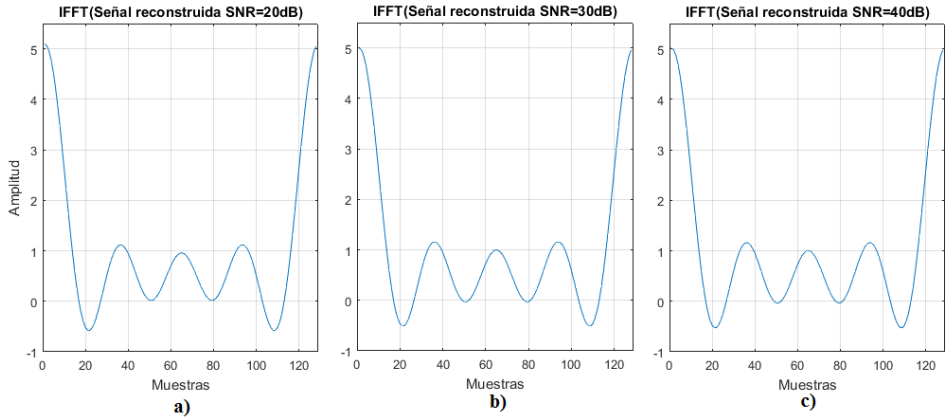

Figura 4.25: IFFT de las señales reconstruidas con dispersidad $k = 5$ con ruido: a)SNR=20dB, b)SNR=30dB y c)SNR=40dB.

Se puede observar claramente que hay cierta robustez al ruido en las gráficas. En donde las muestras tenían un $SNR = 20dB$ se logra observar que la forma de la señal cambió un poco.

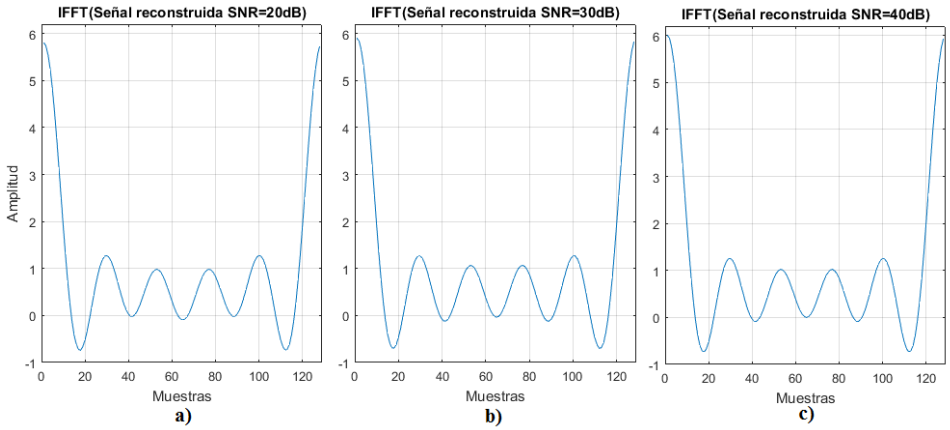

Figura 4.26: IFFT de las señales reconstruidas con dispersidad $k = 6$ con ruido: a)SNR=20dB, b)SNR=30dB y c)SNR=40dB.

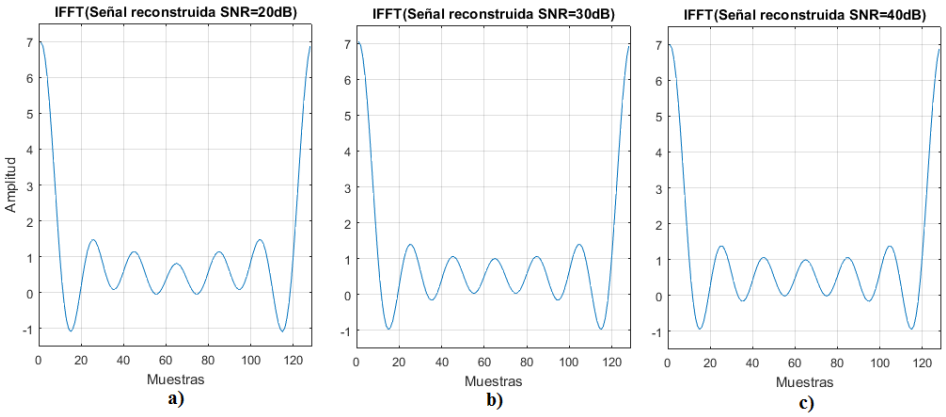

Figura 4.27: IFFT de las señales reconstruidas con dispersidad $k = 7$ con ruido: a)SNR=20dB, b)SNR=30dB y c)SNR=40dB.

4.7. Aplicación del algoritmo CoSaMP para reconstrucción de señales EEG

Se presenta a continuación la reconstrucción de la representación dispersa de señales cerebrales obtenidas mediante EEGs; además, se aplica la transformada de Fourier inversa a estas para poder observar el error de las reconstrucciones realizadas.

Los EEG fueron extraídos de la base de datos de *BCI Competition*. Se extrajeron las señales de la competición *IV* y se utilizó el data set 2a. Del electrodo *fz* se obtuvieron las señales correspondientes al movimiento de la lengua. Se hi-

zo también un *downsample* de las mismas señales para obtener una frecuencia de muestreo de 128Hz. También, las señales fueron separadas en bandas mediante descomposición wavelet para un mejor estudio. Para las pruebas mostradas a continuación, se tomó solo 1 segundo de la señal obtenida de las diferentes bandas.

Se reconstruyeron 3 señales, una en la banda Alfa, otra en la banda Beta y la ultima en la banda Theta. Para reconstruirlas se utilizó un diccionario de tamaño 64x128 ya que se utilizan valores elevados de nivel de dispersidad k. Cada reconstrucción fue realizada a partir de tres iteraciones del algoritmo.

La primer banda que se utilizo fue la Alfa, son señales que están en el rango de frecuencias de 8Hz a 13.99Hz. En la figura 4.28 se observa la señal que se utilizó para realizar sensado comprimido.

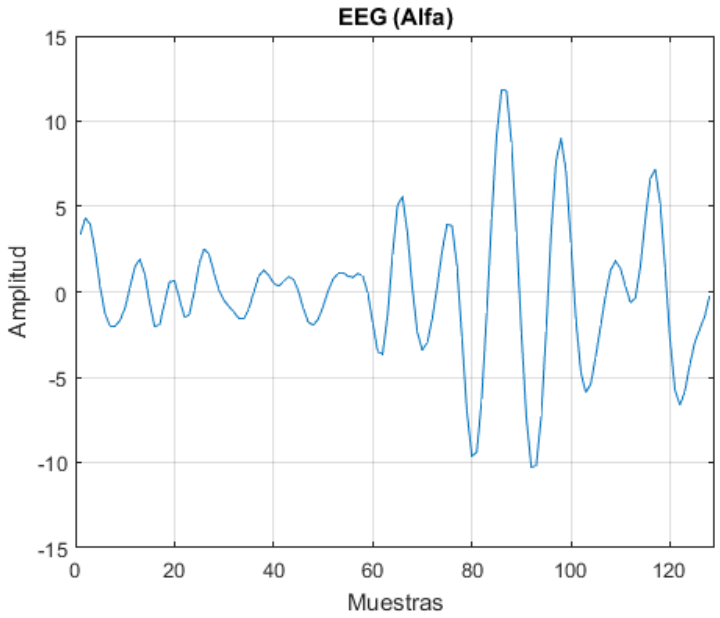

Figura 4.28: Banda Alfa de EGG.

En la figura 4.29 se observa la mitad del espectro de Fourier de la banda Alfa.

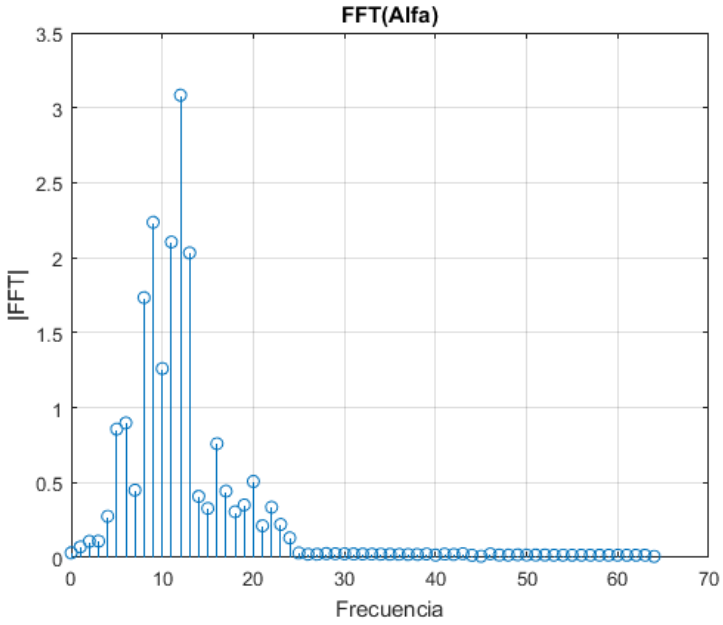

Figura 4.29: Mitad del espectro de Fourier de la banda Alfa.

Para reconstruir la señal se tomaron tres valores diferentes de nivel de dispersidad de la señal($k = 20$, $k = 15$ y $k = 10$). En la figura 4.30 se muestra la mitad del espectro de Fourier reconstruido.

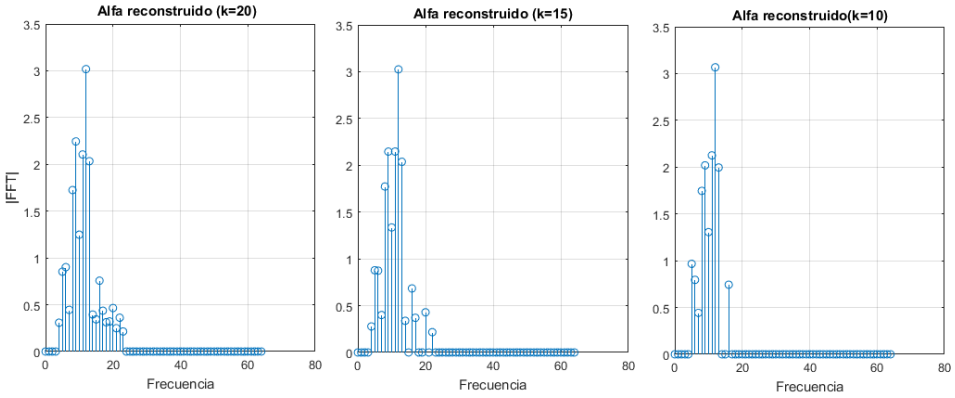

Figura 4.30: Mitad del espectro reconstruido de Fourier de la banda Alfa para diferentes valores de nivel de dispersidad k.

Al aplicar la transformada inversa de Fourier a las señales reconstruidas anteriormente, se obtienen las señales presentadas en la figura 4.31.

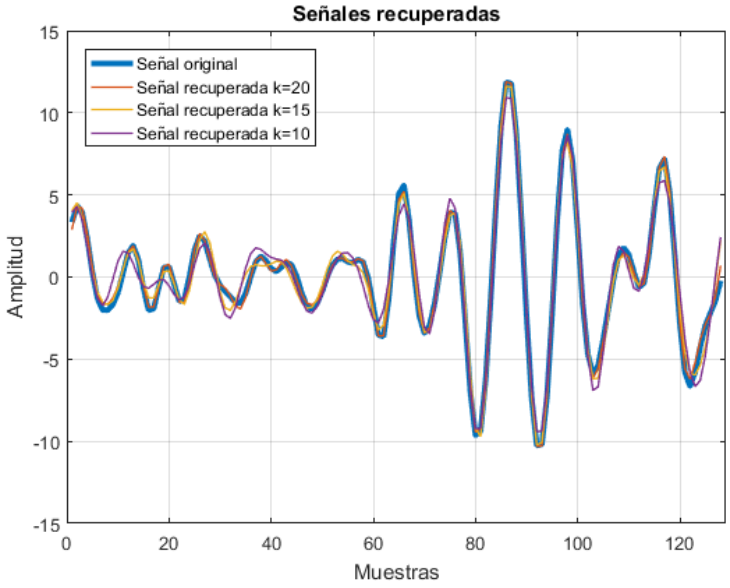

Figura 4.31: Comparación de las señales recuperadas con la señal original tomando diferentes valores de k.

La segunda señal cerebral que se reconstruyó fue una en la banda Beta. En la banda Beta están las señales que tienen un rango de frecuencias de 14Hz a 30Hz. En la figura 4.32 se muestra la señal a recuperar.

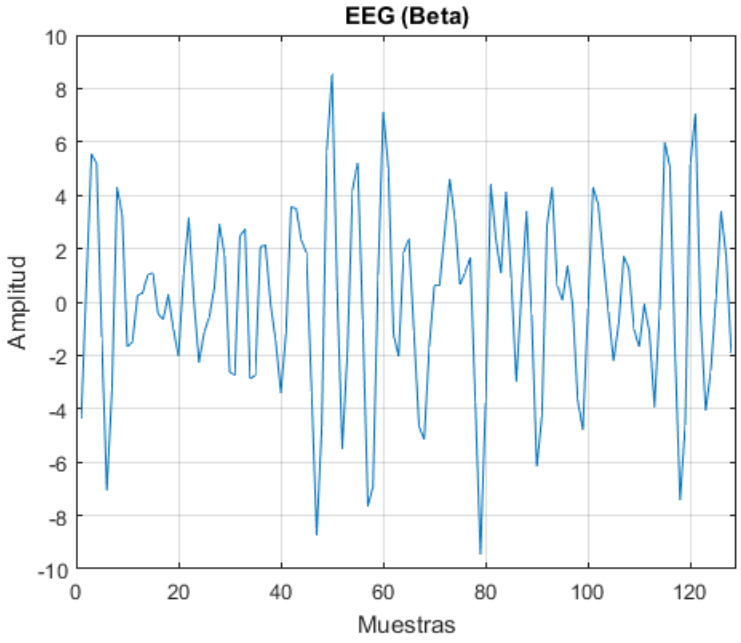

Figura 4.32: Banda Beta de EGG.

El espectro de Fourier para la señal en la banda Beta se muestra en la figura 4.33.

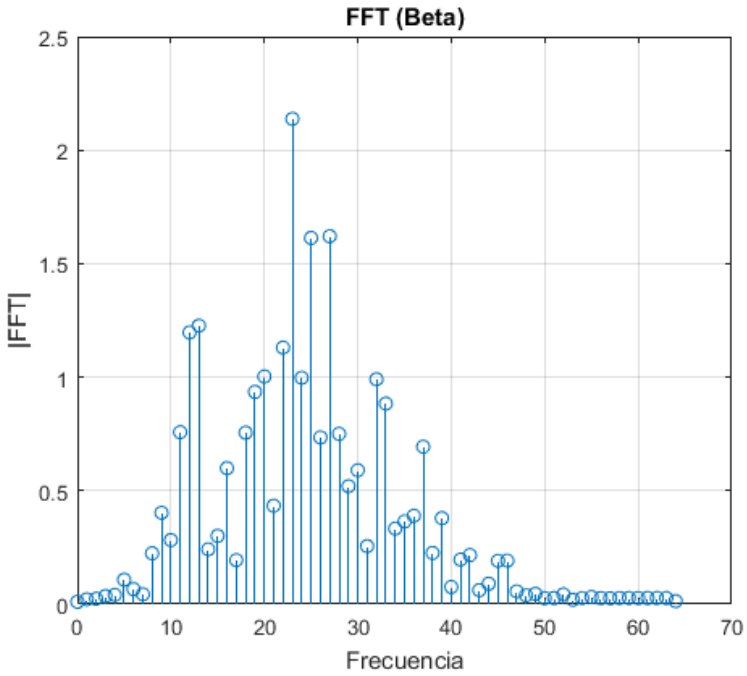

Figura 4.33: Mitad del espectro de Fourier de la banda Beta.

Las señales reconstruidas a partir de las muestras tomadas, se muestran en la figura 4.34.

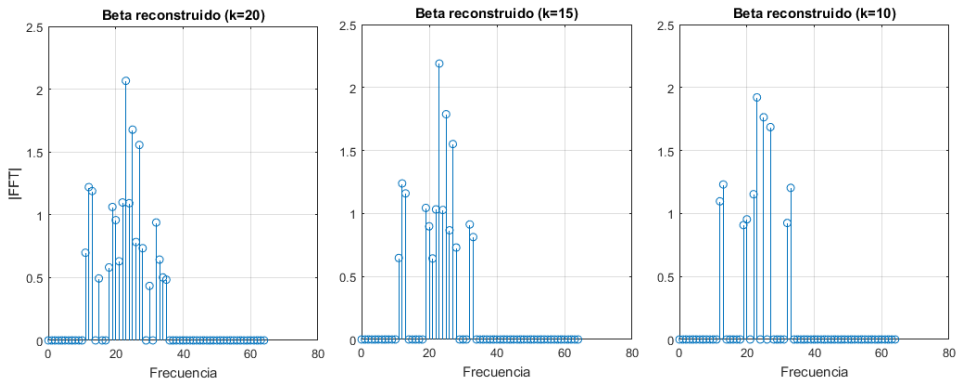

Figura 4.34: Mitad del espectro reconstruido de Fourier de la banda Beta para diferentes valores de nivel de dispersidad k.

Mediante la transformada inversa de Fourier, se pudieron obtener las siguientes señales cerebrales utilizando diferentes valores de k.

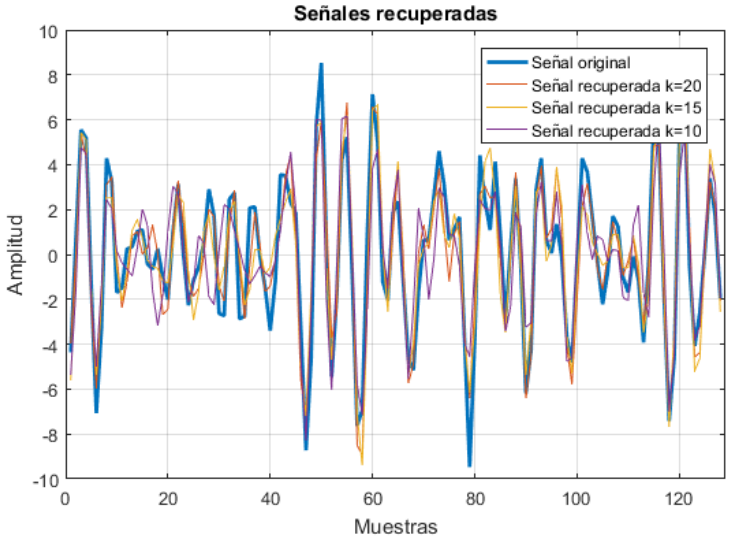

Figura 4.35: Comparación de las señales recuperadas con la señal original tomando diferentes valores de k.

Se le hace un zoom a la figura 4.35, para poder observar mejor el error de la reconstrucción.

Figura 4.36: Zoom de la figura anterior.

Finalmente se reconstruye una señal cerebral en la banda Theta. Son señales que tienen un rango de frecuencias que van de 4Hz a 7.99Hz y se alcanzan bajo un estado de calma profunda. En la figura 4.37 se muestra la señal utilizada.

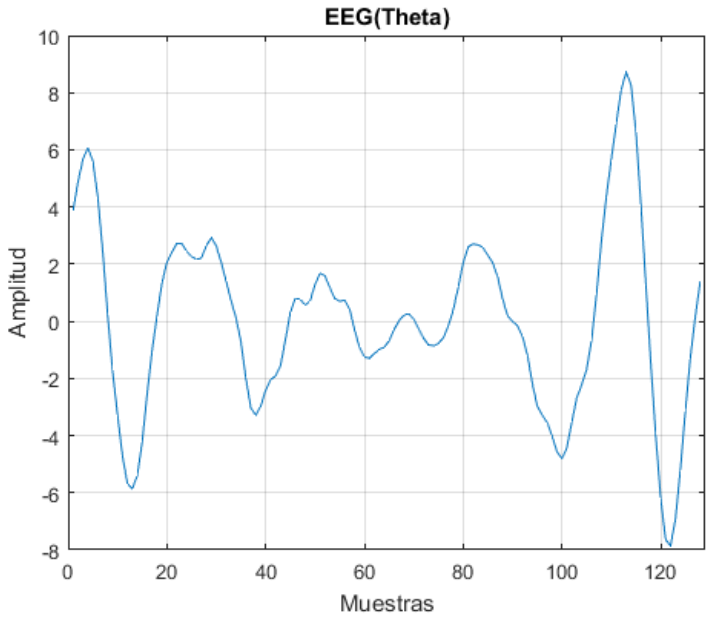

Figura 4.37: Banda Theta de EGG.

La mitad del espectro de Fourier de la señal en la banda Theta a utilizar se muestra en la figura 4.38.

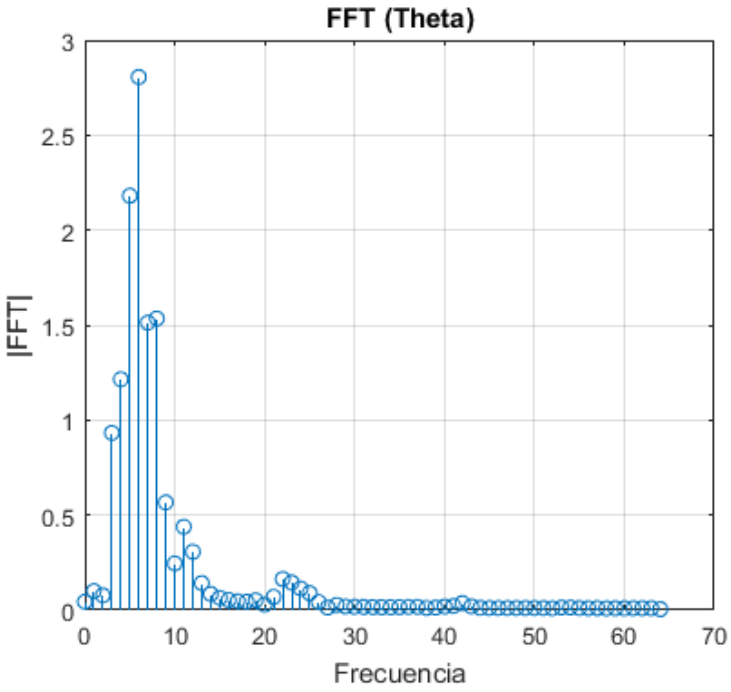

Figura 4.38: Mitad del espectro de Fourier de la banda Theta.

Las señales reconstruidas se muestran en la figura 4.39, se utilizaron tres valores de k.

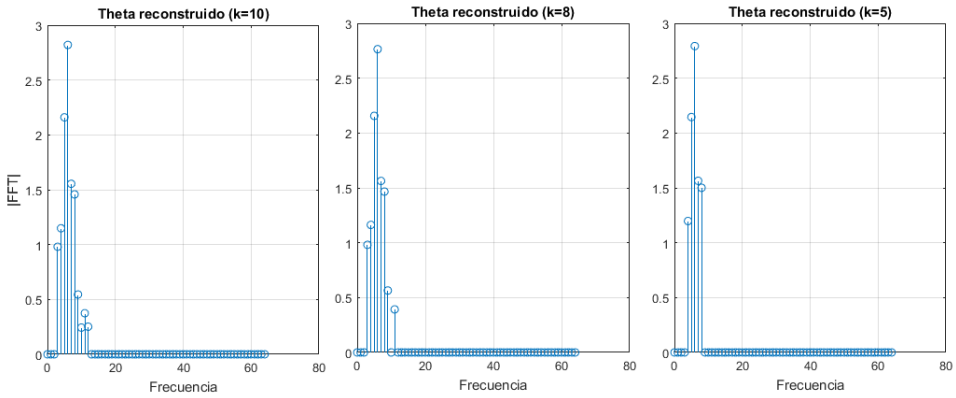

Figura 4.39: Mitad del espectro reconstruido de Fourier de la banda Theta para diferentes valores de nivel de dispersidad k.

Al aplicar la transformada inversa de Fourier a las señales reconstruidas anteriormente, recuperamos la señal cerebral en la banda Theta.

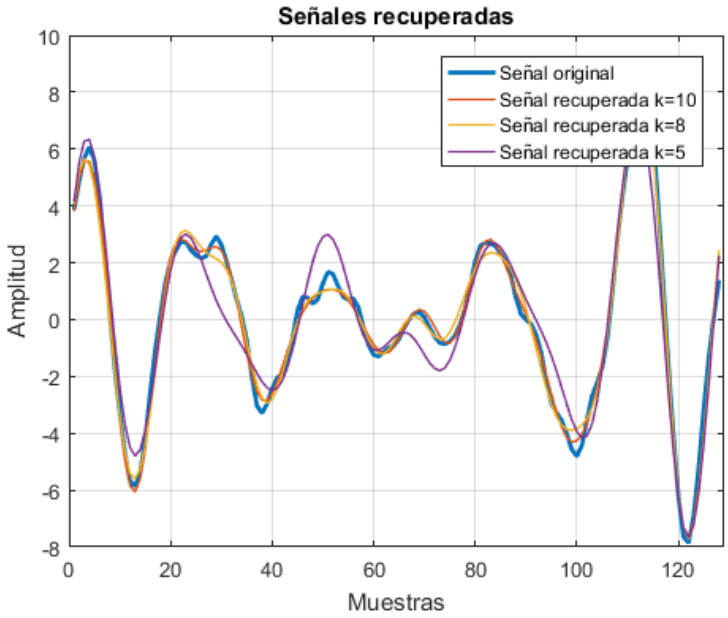

Figura 4.40: Comparación de las señales recuperadas con la señal original tomando diferentes valores de k.

Capítulo 5

Conclusiones

5.1. Conclusiones

Las arquitecturas que fueron presentadas son para señales a reconstruir con nivel de dispersidad k no tan grande, ya que lo que se buscó fue reconstruir señales a alta velocidad. Sin embargo, dichas arquitecturas pueden reconstruir señales con valores de dispersidad mas elevado si se utilizan varios conjuntos de muestras. Al trabajar con matrices pequeñas le es mas difícil al sintetizador, y por lo tanto se requiere una mayor cantidad de recursos para lograr que funcione el algoritmo a la misma frecuencia de operación que las de mayor tamaño(esfuerzo del sintetizador = default).

5.2. Modificaciones al algoritmo

Al reducir el tamaño de Ω no se observó mayor problema en cuanto al numero de iteraciones necesarias para reconstruir una señal.
Se probó la primer sugerencia que se hace en el artículo [52]. Si bien uno logra un algoritmo mas eficiente en el uso de recursos, la compresión de la señal tuvo que ser menor y el numero de iteraciones debió aumentar con las pruebas que se hicieron.

5.3. Trabajo futuro

- Hacer un análisis de latencia para señales con nivel de dispersidad mayor a las presentadas y ver si es mejor mantener el nivel de dispersidad bajo y

hacer varias reconstrucciones o una sola reconstrucción con el valor del nivel de dispersidad alto.

- Configurar una memoria RAM externa para poder utilizar señales mas grandes. Al aumentar el tamaño de la señal también aumenta el tamaño del diccionario.
- Utilizar un método iterativo como gradiente descendiente o gradiente conjugado para resolver el problema de mínimos cuadrados; esta demostrado que es una forma mas eficiente en cuanto a utilización de recursos y latencia.
- Utilizar una configuración de punto fijo y una mayor precision en la palabra.
- Para el método de Chebyshev probar iniciando la matriz con una matriz diagonal en la cual los datos son el recíproco de la diagonal de la matriz a invertir.

Apéndices

Apéndice

Detalles de los experimentos

Para verificar resultados de la simulación, se utilizó System Generator en Simulink. En la figura A.1se puede observar el diseño que se hizo.

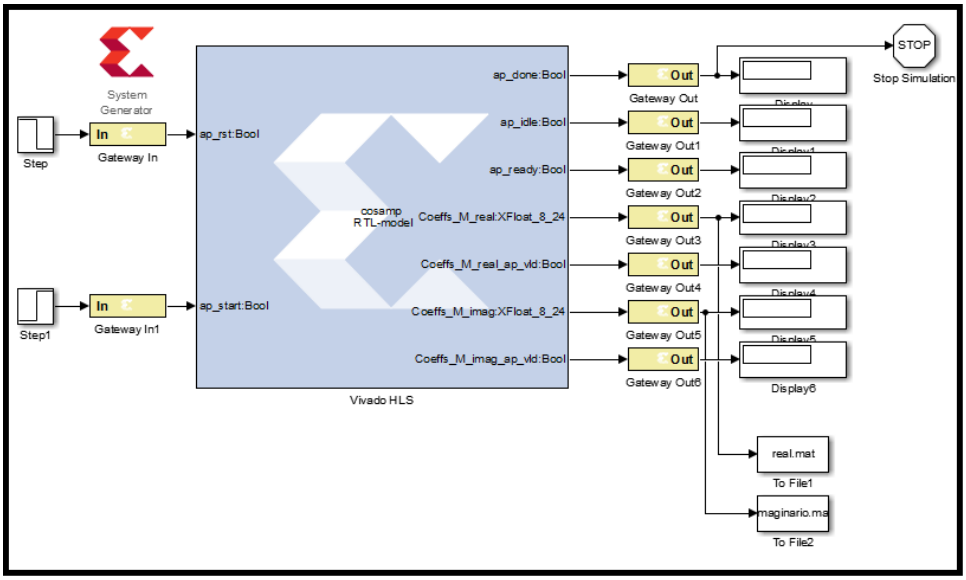

Figura A.1: Co-simulación de hardware con System Generator.

Como en Simulink, las entradas de los bloques se encuentran a la izquierda y las salidas a la derecha. Hay dos entradas, el *reset* y *start*. Para iniciar la simulación se les pone un escalón unitario para excitarlas. Cuando el reset esta en un valor lógico de 1, el start esta en 0 y viceversa. En las salidas se tienen diferentes banderas las cuales dicen el estado en que se encuentra la simulación. La primera *ap_done* da información de cuando acaba o si aún esta corriendo la simulación. Si es 1, quiere decir que la simulación ya acabo, si es 0 que la simulación aun esta corriendo. La salida *ap_idle* da información sobre si el bloque se esta ejecutando o no. Si tiene un

valor de 1, quiere decir que no esta haciendo nada, si tiene 0 que el bloque se esta ejecutando. La salida *ap_ready* nos dice que el bloque esta listo para ejecutarse. Por la salida *Coeffs_M_real* se obtiene la parte real de los coeficientes calculados de la señal dispersa. Adicionalmente la salida *Coeffs_M_real_ap_vld* se pone en alto cuando hay salidas de datos en *Coeffs_M_real*. En *Coeffs_M_imag* se obtiene la parte imaginaria de los coeficientes calculados de la señal dispersa. Y también *Coeffs_M_imag_ap_vld* se pondrá en alto cuando haya una salida en *Coeffs_M_imag*. La configuración del FPGA y del reloj utilizado se muestra en las figuras A.2 y A.3.

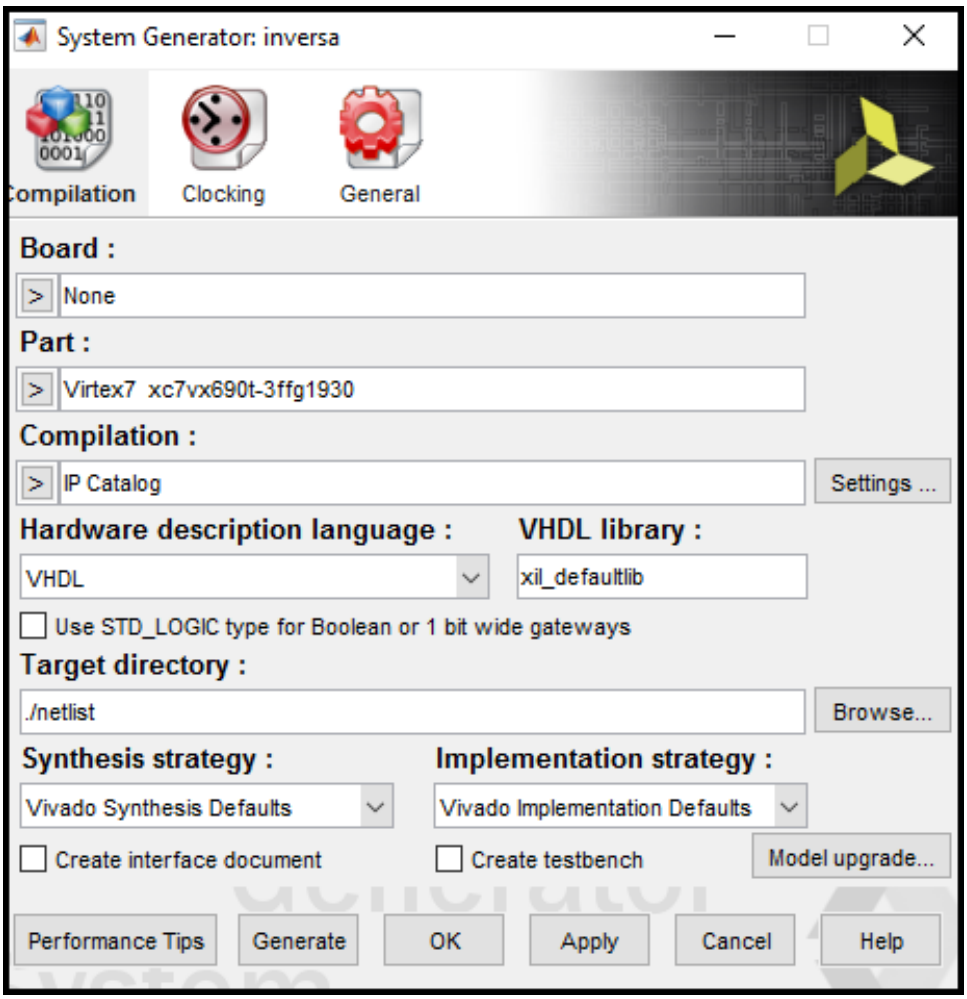

Figura A.2: FPGA a utilizar.

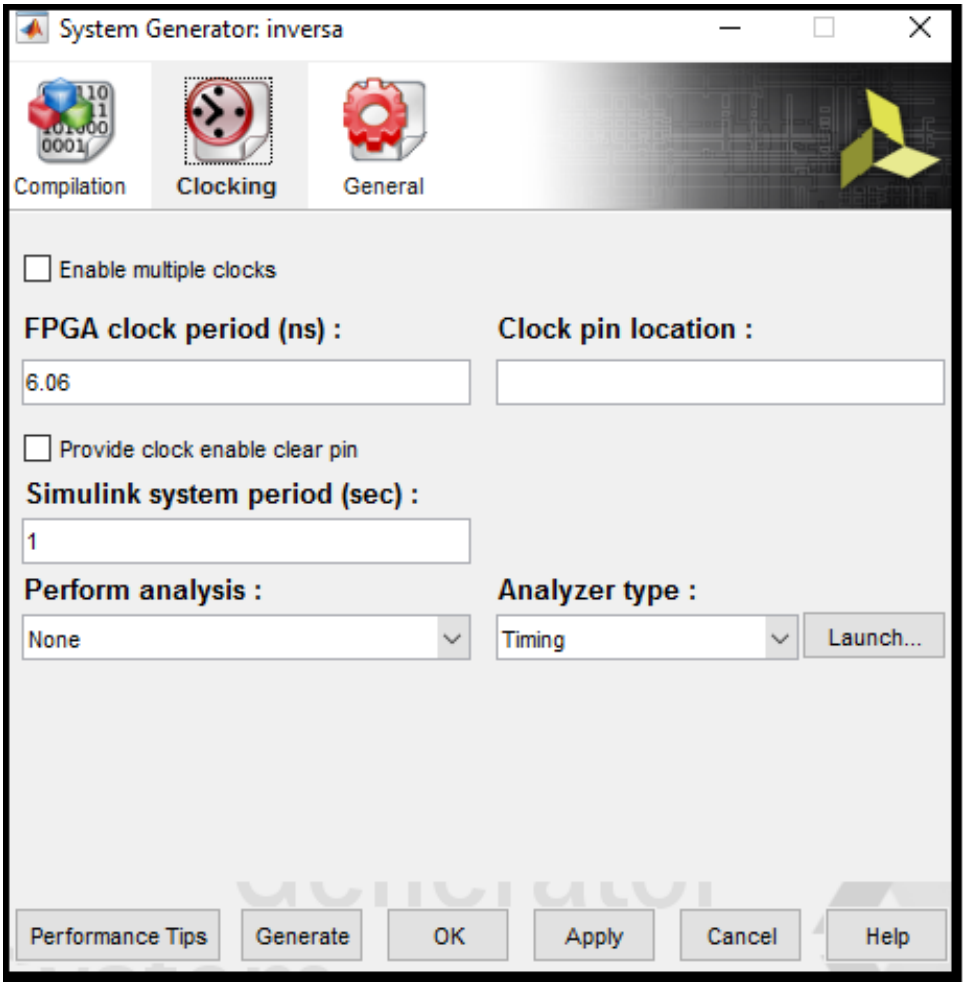

Figura A.3: Configuración del reloj.

Bibliografía

[1] H. Nyquist, "Certain topics in telegraph transmission theory," *Transactions of the American Institute of Electrical Engineers*, vol. 47, pp. 617–644, April 1928.

[2] E. J. Candès, J. K. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on Pure and Applied Mathematics*, vol. 59, no. 8, pp. 1207–1223, 2006.

[3] D. L. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, pp. 1289–1306, April 2006.

[4] M. Lustig, D. L. Donoho, J. M. Santos, and J. M. Pauly, "Compressed sensing mri," *IEEE Signal Processing Magazine*, vol. 25, pp. 72–82, March 2008.

[5] M. A. Hadi, S. Alshebeili, K. Jamil, and F. E. A. El-Samie, "Compressive sensing applied to radar systems: an overview," *Signal, Image and Video Processing*, vol. 9, pp. 25–39, Dec 2015.

[6] A. Y. Yang, S. S. Sastry, A. Ganesh, and Y. Ma, "Fast l1-minimization algorithms and an application in robust face recognition: A review," in *2010 IEEE International Conference on Image Processing*, pp. 1849–1852, Sept 2010.

[7] H. Jiang, W. Deng, and Z. Shen, "Surveillance video processing using compressive sensing," *CoRR*, vol. abs/1302.1942, 2013.

[8] X. Fei, Z. Wei, and L. Xiao, "Iterative directional total variation refinement for compressive sensing image reconstruction," *IEEE Signal Processing Letters*, vol. 20, pp. 1070–1073, Nov 2013.

[9] H. Cheng, *Sparse Representation, Modeling and Learning in Visual Recognition: Theory, Algorithms and Applications*. Advances in Computer Vision and Pattern Recognition, Springer London, 2015.

[10] Y. Eldar and G. Kutyniok, *Compressed Sensing: Theory and Applications*. Cambridge University Press, 2012.

[11] A. M. Bruckstein, D. L. Donoho, and M. Elad, "From sparse solutions of systems of equations to sparse modeling of signals and images," *SIAM Review*, vol. 51, no. 1, pp. 34–81, 2009.

[12] M. Elad, *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*. Springer New York, 2010.

[13] R. A. DeVore, "Nonlinear approximation," *Acta Numerica*, vol. 7, p. 51–150, 1998.

[14] T. G. Stockham, Jr., "High-speed convolution and correlation," in *Proceedings of the April 26-28, 1966, Spring Joint Computer Conference*, AFIPS '66 (Spring), (New York, NY, USA), pp. 229–233, ACM, 1966.

[15] K. Rao and P. Yip, *The Transform and Data Compression Handbook*. Electrical Engineering & Applied Signal Processing Series, CRC Press, 2000.

[16] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, vol. C-23, pp. 90–93, Jan 1974.

[17] I. W. Selesnick, R. G. Baraniuk, and N. C. Kingsbury, "The dual-tree complex wavelet transform," *IEEE Signal Processing Magazine*, vol. 22, pp. 123–151, Nov 2005.

[18] S. G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, pp. 674–693, July 1989.

[19] D. Donoho and M. Elad, "Optimally sparse representation in general (nonorthogonal) dictionaries via l1 minimization," vol. 100, pp. 2197–202, 04 2003.

[20] A. Cohen, W. Dahmen, and R. DeVore, "Compressed sensing and best k-term approximation," *Journal of the American mathematical society*, vol. 22, no. 1, pp. 211–231, 2009.

[21] E. J. Candes and T. Tao, "Decoding by linear programming," *IEEE Transactions on Information Theory*, vol. 51, pp. 4203–4215, Dec 2005.

[22] J. Bourgain, S. Dilworth, K. Ford, S. Konyagin, and D. Kutzarova, "Explicit constructions of rip matrices and related problems," *Duke Math. J.*, vol. 159, pp. 145–185, 07 2011.

[23] R. A. DeVore, "Deterministic constructions of compressed sensing matrices," *Journal of Complexity*, vol. 23, no. 4, pp. 918 – 925, 2007. Festschrift for the 60th Birthday of Henryk Woźniakowski.

[24] J. Haupt, L. Applebaum, and R. Nowak, *On the restricted isometry of deterministically subsampled Fourier matrices.* 2010.

[25] P. Indyk, "Explicit constructions for compressed sensing of sparse signals," in *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, (Philadelphia, PA, USA), pp. 30–33, Society for Industrial and Applied Mathematics, 2008.

[26] M. A. Davenport, J. N. Laska, P. T. Boufounos, and R. G. Baraniuk, "A simple proof that random matrices are democratic," *Computing Research Repository*, vol. abs/0911.0, 2009.

[27] J. N. Laska, P. T. Boufounos, M. A. Davenport, and R. G. Baraniuk, "Democracy in action: Quantization, saturation, and compressive sensing," *Applied and Computational Harmonic Analysis*, vol. 31, no. 3, pp. 429 – 443, 2011.

[28] R. Baraniuk, M. Davenport, R. DeVore, and M. Wakin, "A simple proof of the restricted isometry property for random matrices," *Constructive Approximation*, vol. 28, pp. 253–263, Dec 2008.

[29] A. Ben-Israel and T. Greville, *Generalized Inverses: Theory and Applications.* CMS Books in Mathematics, Springer, 2003.

[30] J. Stoer, R. Bartels, W. Gautschi, R. Bulirsch, and C. Witzgall, *Introduction to Numerical Analysis*. Texts in Applied Mathematics, Springer New York, 2002.

[31] G. Stewart, *Matrix Algorithms: Volume 1: Basic Decompositions*. Society for Industrial and Applied Mathematics, 1998.

[32] R. Parker, *Geophysical Inverse Theory*. Princeton series in geophysics, Princeton University Press, 1994.

[33] E. Cheney and D. Kincaid, *Linear Algebra: Theory and Applications*. Jones and Bartlett Publishers, 2009.

[34] G. H. Golub and C. F. Van Loan, *Matrix Computations (3rd Ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[35] O. Alter and G. H. Golub, "Integrative analysis of genome-scale data by using pseudoinverse projection predicts novel correlation between dna replication and rna transcription," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 47, pp. 16577–16582, 2004.

[36] S. Amat, S. Busquier, and J. Gutiérrez, "Geometric constructions of iterative functions to solve nonlinear equations," *Journal of Computational and Applied Mathematics*, vol. 157, no. 1, pp. 197 – 205, 2003.

[37] H.-B. Li, T.-Z. Huang, Y. Zhang, X.-P. Liu, and T.-X. Gu, "Chebyshev-type methods and preconditioning techniques," *Applied Mathematics and Computation*, vol. 218, no. 2, pp. 260 – 270, 2011.

[38] L. H. Chang and J. Y. Wu, "Compressive-domain interference cancellation via orthogonal projection: How small the restricted isometry constant of the effective sensing matrix can be?," in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 256–261, April 2012.

[39] M. A. Davenport, P. T. Boufounos, M. B. Wakin, and R. G. Baraniuk, "Signal processing with compressive measurements," *IEEE Journal of Selected Topics in Signal Processing*, vol. 4, pp. 445–460, April 2010.

[40] M. A. Davenport, M. Duarte, M. B. Wakin, J. N. Laska, D. Takhar, K. Kelly, and R. G. Baraniuk, "The smashed filter for compressive classification and target recognition - art. no. 64980h," vol. 6498, 02 2007.

[41] M. F. Duarte, M. A. Davenport, M. B. Wakin, and R. G. Baraniuk, "Sparse signal detection from incoherent projections," in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, vol. 3, pp. III–III, May 2006.

[42] M. F. Duarte, M. A. Davenport, M. B. Wakin, J. N. Laska, D. Takhar, K. F. Kelly, and R. G. Baraniuk, "Multiscale random projections for compressive classification," in *2007 IEEE International Conference on Image Processing*, vol. 6, pp. VI – 161–VI – 164, Sept 2007.

[43] J. Haupt, R. Castro, R. Nowak, G. Fudge, and A. Yeh, "Compressive sampling for signal classification," in *2006 Fortieth Asilomar Conference on Signals, Systems and Computers*, pp. 1430–1434, Oct 2006.

[44] D. Needell, "Compressed sensing with coherent and redundant dictionaries," 01 2011.

[45] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM JOURNAL ON SCIENTIFIC COMPUTING*, vol. 20, pp. 33–61, 1998.

[46] T. Blumensath and M. E. Davies, "Gradient pursuits," *IEEE Transactions on Signal Processing*, vol. 56, pp. 2370–2382, June 2008.

[47] T. Blumensath and M. E. Davies, "Iterative hard thresholding for compressed sensing," *Applied and Computational Harmonic Analysis*, vol. 27, no. 3, pp. 265 – 274, 2009.

[48] A. Cohen, W. Dahmen, and R. Devore, "Instance optimal decoding by thresholding in compressed sensing," vol. 505, 12 2008.

[49] W. Dai and O. Milenkovic, "Subspace pursuit for compressive sensing signal reconstruction," *IEEE Transactions on Information Theory*, vol. 55, pp. 2230–2249, May 2009.

[50] M. A. Davenport and M. B. Wakin, "Analysis of orthogonal matching pursuit using the restricted isometry property," *IEEE Transactions on Information Theory*, vol. 56, pp. 4395–4401, Sept 2010.

[51] D. L. Donoho and J. Tanner, "Precise undersampling theorems," *Proceedings of the IEEE*, vol. 98, pp. 913–924, June 2010.

[52] D. Needell and J. Tropp, "Cosamp: Iterative signal recovery from incomplete and inaccurate samples," *Applied and Computational Harmonic Analysis*, vol. 26, no. 3, pp. 301 – 321, 2009.

[53] D. Du and F. Hwang, *Combinatorial Group Testing and Its Applications*. Series on Applied Mathematics, 1999.

[54] Y. Erlich, N. Shental, A. Amir, and O. Zuk, "Compressed sensing approach for high throughput carrier screen," in *2009 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 539–544, Sept 2009.

[55] S. Sarvotham, D. Baron, and R. G. Baraniuk, "Sudocodes 2013; fast measurement and reconstruction of sparse signals," in *2006 IEEE International Symposium on Information Theory*, pp. 2804–2808, July 2006.

[56] P. Blache, H. Rabah, and A. Amira, "High level prototyping and fpga implementation of the orthogonal matching pursuit algorithm," in *2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*, pp. 1336–1340, July 2012.

[57] A. Borghi, J. Darbon, S. Peyronnet, T. F. Chan, and S. Osher, "A simple compressive sensing algorithm for parallel many-core architectures," *Journal of Signal Processing Systems*, vol. 71, pp. 1–20, Apr 2013.

[58] Y. Fang, L. Chen, J. Wu, and B. Huang, "Gpu implementation of orthogonal matching pursuit for compressive sensing," in *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pp. 1044–1047, Dec 2011.

[59] F. Ren, R. Dorrace, W. Xu, and D. Marković, "A single-precision compressive sensing signal reconstruction engine on fpgas," in *2013 23rd International Conference on Field programmable Logic and Applications*, pp. 1–4, Sept 2013.

[60] J. Lu, H. Zhang, and H. Meng, "Novel hardware architecture of sparse recovery based on fpgas," in *2010 2nd International Conference on Signal Processing Systems*, vol. 1, pp. V1–302–V1–306, July 2010.

[61] A. Septimus and R. Steinberg, "Compressive sampling hardware reconstruction," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 3316–3319, May 2010.

[62] L. Bai, P. Maechler, M. Muehlberghuber, and H. Kaeslin, "High-speed compressed sensing reconstruction on fpga using omp and amp," in *2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*, pp. 53–56, Dec 2012.

[63] H. Rabah, A. Amira, B. K. Mohanty, S. Almaadeed, and P. K. Meher, "Fpga implementation of orthogonal matching pursuit for compressive sensing reconstruction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, pp. 2209–2220, Oct 2015.

[64] Y. Quan, Y. Li, X. Gao, and M. Xing, "Fpga implementation of real-time compressive sensing with partial fourier dictionary," vol. 2016, pp. 1–12, 01 2016.

[65] J. N. F. M. V. Daniele Bagni, A. Di Fresco, "A zynq accelerator for floating point matrix multiplication designed with vivado hls," January, 2016. Application note.