



Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

Arquitecturas Digitales de Procesamiento de Señales para Sistemas de Comunicación con Entrenamiento Implícito

Tesis que presenta:
Eduardo Romero Aguirre

para obtener el grado de:
Doctor en Ciencias

en la especialidad de:
Ingeniería Eléctrica

Director de Tesis
Dr. Ramón Parra Michel

Arquitecturas Digitales de Procesamiento de Señales para Sistemas de Comunicación con Entrenamiento Implícito

**Tesis de Doctorado en Ciencias
Ingeniería Eléctrica**

Por:

Eduardo Romero Aguirre

Maestro en Ciencias en Ingeniería Electrónica
Centro Nacional de Investigación y Desarrollo Tecnológico
(CENIDET) 1996-1999

Becario de PROMEP, expediente no. ITSON-092

**Director de Tesis
Dr. Ramón Parra Michel**

CINVESTAV del IPN Unidad Guadalajara, Diciembre de 2012.

Las huellas del señor

*¡Señor, tú me dijistes
que una vez que yo hubiera decidido
seguirte, siempre estarías
a mi lado en el camino.*

*Pero he notado que cuando
yo más sufria, sólo había un par de
huellas y no entiendo porque me
abandonastes cuando más te necesitaba!*

*El señor respondió:
“hijo amado, yo nunca te abandonaría
en tus tiempos de prueba y sufrimiento;
cuando tú has visto sólo un par
de huellas, era que yo te
cargaba en mis brazos.”*

Dedico esta tesis

A mis amados hijos: Andrea y Gibrán, mis razones de ser y mis fuentes de fortaleza al momento de enfrentar cualquier adversidad.

A mi esposa, compañera de vida, amiga y confidente, te amo Norma.

A mis padres: Julia y Javier, por darme el don de la vida y el orgullo de ser su hijo.

A mis queridos hermanos: Rosa, Gaby, May y Javier, por todo el cariño que me han proporcionado y que nos ha permitido ser tan unidos. Además porque gran parte de lo que soy, se lo debo a ellos y a sus sacrificios.

A todos los miembros de las familias Flores C. y Romero A. por cuidar a mi esposa e hijos en mi ausencia.

A Dios, por estar siempre a mi lado y nunca abandonarme en estos tiempos de prueba.

Agradezco

A mi asesor el Dr. Ramón Parra Michel y al Dr. Roberto Carrasco Alvarez, quienes con su invaluable ayuda y asistencia hicieron posible llevar a buen término este proyecto de investigación doctoral.

Al Dr. Mariano Aguirre Hernández por guiarme en los inicios de mi doctorado y al Dr. Antonio Mondragón T. por su valioso apoyo durante mi estancia en el Rochester Institute of Technology.

A mis sinodales, Dr. Deni Librado Torres Román, Dr. Mario Angel Siller González Pico, Dra. Susana Ortega Cisneros y Dr. Omar Humberto Longoria Gándara, quienes con sus atinados comentarios y sugerencias permitieron mejorar el presente trabajo.

A todos mis profesores por haber compartido conmigo su experiencia y conocimientos.

A todas las personas con las que tuve la fortuna de convivir durante mi estancia en CINVESTAV-GDL, en particular a: los infiltrados (Francisco Aguilera, David Arditti); mis compañeros de la maestría (generación tele 07); los doctorantes de la vieja guardia (Omar Longoria., Alberto Alcocer, Alberto Sánchez, Roberto Carrasco, Alejandro Castillo, Homero Toral, Leopoldo Estrada, Iván Loyola); los doctorantes de la nueva guardia (Javier Vázquez, Jesús Argaez., David Castro, José Luis Vázquez, Israel Yañez) y las generaciones de tele 08, 09, 10, 11.

Al CINVESTAV-GDL, por darme la oportunidad de estudiar en su programa de doctorado en ciencias.

Al Instituto Tecnológico de Sonora, por depositar su confianza en mí y proporcionarme todas las facilidades necesarias para llevar a cabo estos estudios de doctorado.

Al pueblo de México que a través de la SEP y PROMEP me brindó el apoyo económico para la realización de este posgrado.

Resumen

El uso de técnicas de entrenamiento implícito (IT) en sistemas de comunicación inalámbrica ha emergido en años recientes como una alternativa viable para sustituir a las técnicas de entrenamiento convencionales. La mayoría de las investigaciones publicadas en el estado de arte coinciden en que los sistemas de comunicaciones con IT tienen un rendimiento comparable a aquellos basados en entrenamiento explícito (ET), pero con la ventaja de un mejor aprovechamiento del ancho de banda disponible; además de permitir enviar datos y entrenamiento en todo momento durante la transmisión. Tales características hacen factible su uso en sistemas tales como WiFi (*Wireless Fidelity*) o en difusión de video digital (DVB), por citar algunos. Pero, a pesar de esta motivación y aunque la investigación asociada a IT es lo suficientemente madura desde el punto de vista de: modelado matemático, aplicaciones y análisis de desempeño; son muy pocos los trabajos en la literatura que presentan algún tipo implementación física incluso a nivel experimental. Hasta el momento, los prototipos reportados están basados mayormente en software y emplean punto flotante para sus cálculos. Ninguna de estas propuestas ha sido capaz de desarrollar velocidades de procesamiento requeridas por los estándares de comunicación actuales. Por tal motivo, en este trabajo doctoral se abordan los pormenores relativos al diseño e implementación de arquitecturas de tipo *full-hardware* en punto fijo, para las etapas que conforman un sistema de comunicación con IT en sus variantes de: entrenamiento superimpuesto (ST) y entrenamiento superimpuesto dependiente de los datos (DDST). En particular, se desarrollan arquitecturas digitales para: un transmisor ST/DDST, estimadores de canal para un receptor DDST, procesadores FFT y un correlacionador. Los resultados de la evaluación de todos ellos bajo métricas que cuantifican tanto la eficiencia de su implementación como su desempeño funcional, permiten establecer que el concepto de ST/DDST puede ser utilizado en sistemas de comunicaciones prácticos.

Abstract

The use of implicit training (IT) techniques in wireless communication systems has emerged in recent years as a viable alternative to replace the conventional training techniques. Most of the published state-of-the-art research agrees that communications systems with IT included have the same performance as those with explicit training (ET), but with the advantage of a better use of available bandwidth. Additionally, they permit to send data and training pilots continuously during the transmission. These features makes it feasible to apply the IT technique in systems such as WiFi (Wireless Fidelity) or digital video broadcasting (DVB), to name a few. Nevertheless, despite this motivation and the fact that IT-related research is mature from the point of view of mathematical modeling, applications and performance analysis, there are few articles in the literature that report any physical implementation even to the experimental level. So far, the prototypes are based mostly on software and they using floating-point arithmetic. None of them have been able to achieve the required processing speeds for the current communication standards. For that reason, this doctoral research presents the details concerning the design and implementation of the fixed-point full-hardware architectures for the stages of communication system based on IT variants known as superimposed training (ST) and data-dependent ST (DDST). In particular, digital architectures for a ST/DDST transmitter, channel estimators for DDST receiver, FFT processors and a correlator are developed. The evaluation results of all them under metrics for quantifying both implementation efficiency and functional performance, lead us to the conclusion that the DDST concept can be utilized in practical communication systems.

Contenido

Contenido	1
1 Introducción	5
1.1. Sistemas de comunicación y el concepto de entrenamiento implícito	5
1.2. Estado del arte	6
1.3. Planteamiento del problema	8
1.4. Objetivos de la tesis	9
1.4.1. Objetivo general	9
1.4.2. Objetivos particulares	9
1.5. Organización de la tesis	10
1.6. Contribuciones de la tesis	11
2 Sistema de comunicación con entrenamiento superimpuesto	13
2.1. Introducción	13
2.2. Clasificación de las técnicas de estimación de canal	13
2.2.1. Estimación ciega	14
2.2.2. Estimación basada en entrenamiento	14
2.2.3. Entrenamiento explícito (ET)	15
2.2.4. Entrenamiento implícito (IT)	15
2.3. Sistema de comunicación con entrenamiento implícito	17
2.3.1. Transmisor con entrenamiento implícito	17
2.3.2. Receptor con entrenamiento implícito	18
2.4. Conclusiones del capítulo	19
3 Algoritmos de procesamiento digital de señales para un sistema de comunicación con entrenamiento implícito	21
3.1. Introducción	21
3.2. Algoritmos para estimación de canales invariantes en el tiempo empleando ST/DDST	21
3.2.1. Modelo del sistema y consideraciones generales	21

3.2.2. Estimación de canal considerando sincronía perfecta y ausencia desplazamiento de CD	23
3.2.3. Estimación de canal bajo sincronía perfecta y con desplazamiento de CD	24
3.2.4. Análisis de desempeño	25
3.2.5. Estimación de CFO	26
3.2.6. Algoritmo para sincronización de secuencia de entrenamiento y sincronización de bloque	27
3.2.7. Estimación del desplazamiento de CD y estimación de canal	29
3.2.8. Ecualización	29
3.2.9. Estimación iterativa (detección) de los datos	30
3.2.10. Diseño de la secuencia de entrenamiento	30
3.3. Algoritmo de estimación de canales invariantes usando proyección en bases y ST	31
3.3.1. Proyección en bases	31
3.3.2. Base universal	32
3.3.3. Modelo matemático	33
3.3.4. Análisis de desempeño	35
3.4. Algoritmos para la transformada discreta de Fourier	36
3.4.1. Clasificación	37
3.4.2. Algoritmo FFT radix-2 decimado en tiempo (DIT-2)	37
3.4.3. Algoritmo FFT radix-2 decimado en frecuencia (DIF-2)	38
3.4.4. Algoritmo FFT radix-4 decimado en tiempo (DIT-4)	39
3.5. Algoritmos para la convolución y correlación	41
3.5.1. Convolución discreta	41
3.5.2. Correlación discreta	41
3.6. Conclusiones del capítulo	42
4 Desarrollo de un transmisor ST/DDST configurable	43
4.1. Introducción	43
4.2. Procedimiento de mapeo de algoritmo a arquitectura	43
4.2.1. Generación del modelo de oro	44
4.2.2. Adecuación del algoritmo	44
4.2.3. Análisis de punto fijo	44
4.2.4. Diseño de la arquitectura de <i>hardware</i>	46
4.2.5. Implementación y verificación de la arquitectura	47
4.3. Arquitectura digital de un transmisor ST/DDST configurable	48
4.3.1. El adecuador de símbolos	49
4.3.2. El mapeador	52
4.3.3. Módulo transformador de secuencia	54
4.3.4. Generador de direcciones (AGU_Tx) y unidad de control	58
4.4. Resultados de implementación y simulación	59

5 Desarrollo de arquitecturas de PDS para un receptor DDST	67
5.1. Introducción	67
5.2. Arquitecturas de <i>hardware</i> de estimadores de canal para DDST bajo sincronía perfecta y sin desplazamiento de CD	68
5.2.1. Arquitectura digital del estimador CEDAM	69
5.2.2. Arquitectura digital del estimador CEOF	72
5.2.3. Diseño de un estimador de canal sistólico para DDST	74
5.2.4. Resultados de implementación en FPGA	78
5.2.5. Resultados de implementación en ASIC	81
5.2.6. Resultados de simulación	83
5.3. Arquitectura digital de un estimador de canal para DDST con corrección de sincronía y estimación del desplazamiento de CD.	86
5.3.1. Adecuaciones del algoritmo	87
5.3.2. Descripción de la arquitectura del estimador de canal ECEOFR	89
5.3.3. Reutilización del módulo MB y el estimador CEOF	91
5.3.4. Generador de direcciones de lectura para el MB (AGU_MB)	91
5.3.5. Memoria de almacenamiento temporal (TB)	92
5.3.6. Generador de direcciones de escritura para el TB (AGU_HEST)	92
5.3.7. Generador de direcciones de lectura para el TB (AGU_SYNC)	92
5.3.8. Arquitectura para el cálculo de la norma Euclidiana al cuadrado	93
5.3.9. El módulo FMIN	93
5.3.10. Resultados de implementación en FPGA	93
5.3.11. Resultados de simulación	99
5.4. Arquitecturas de <i>hardware</i> para la FFT	108
5.4.1. Procesador FFT/IFFT radix–2 decimado en tiempo	109
5.4.2. Procesador FFT/IFFT radix–2 con decimado en frecuencia	114
5.4.3. Resultados de implementación y simulación	117
5.5. Correlacionador en el dominio de la frecuencia (FDCP)	122
5.5.1. Correlación sin permutaciones usando la FFT/IFFT radix–2	123
5.5.2. Arquitectura de <i>hardware</i> y funcionamiento	125
5.5.3. Control general del correlacionador (CGCU)	126
5.5.4. Procesador FFT/IFFT reconfigurable con decimado mixto (RIFP)	127
5.5.5. Resultados de implementación y simulación	129
6 Conclusiones	135
6.1. Conclusiones	135
6.2. Líneas de investigación futuras	138
Bibliografía	141
A Artículos publicados en congresos internacionales	147

B Artículos publicados en revistas indexadas	167
Nomenclatura matemática	201
Lista de símbolos y abreviaciones	203
Lista de figuras	205
Lista de tablas	210

Capítulo 1

Introducción

1.1. Sistemas de comunicación y el concepto de entrenamiento implícito

Se puede definir como sistema de comunicación al conjunto de procesos necesarios para transmitir y recibir información. Aunque existen diferentes tipos de estos sistemas —empleados en diversos campos de aplicación— de manera general todos ellos consisten de tres bloques fundamentales: el transmisor, el canal de comunicación y el receptor. Sin embargo, para propósitos de estudio estos bloques son a su vez divididos y tratados como un grupo de módulos o etapas independientes que realizan una función específica, tal como lo sugiere la figura 1.1. La complejidad de cada bloque depende del método de transmisión y de las características del canal.

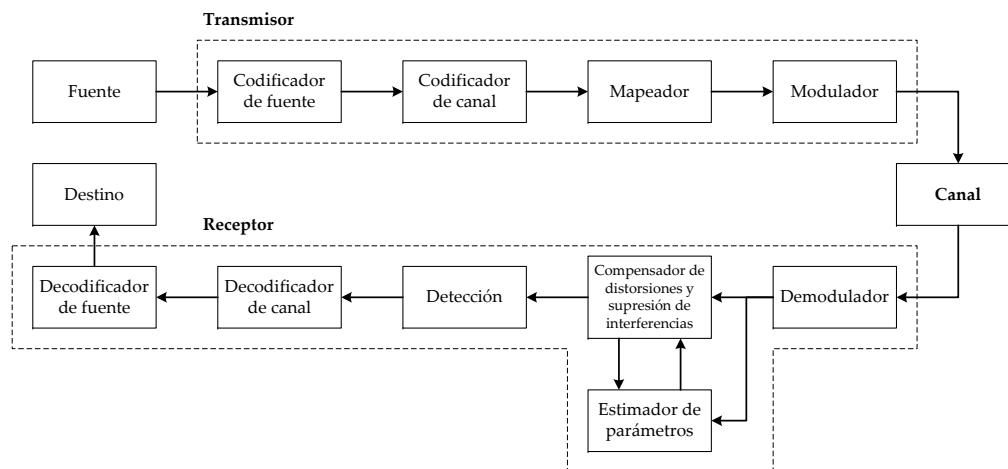


Figura 1.1: Diagrama a bloques simplificado de un sistema de comunicación.

Por otro lado, en la actualidad existe la necesidad continua de diseñar sistemas de comunicación capaces de transmitir/recibir diversos tipos de información (voz, datos, video, etc.) a velocidades cada vez mayores. Esta problemática se acentúa si el entorno donde opera el sistema es el canal inalámbrico debido a su naturaleza compleja y a su susceptibilidad a los diversos tipos de interferencia. Como no es posible evitar la influencia del canal sobre la señal que se envía a través de él, la alternativa es caracterizar sus parámetros con suficiente precisión para que estos efectos puedan ser revertidos en el receptor. Esto se conoce como estimación (de parámetros) del canal y el método que se utilice para tal fin, es crucial en el desempeño del sistema.

Asimismo, para facilitar el proceso de estimación del canal se emplean símbolos de entrenamiento llamados pilotos, que son conocidos por el receptor e insertados en cada bloque de transmisión de datos. En los métodos de entrenamiento explícito (ET), como los usados en GSM, WiFi, Bluetooth, etc., los pilotos son multiplexados con cierta regularidad junto con los datos, con lo que parte del ancho de banda de transmisión debe ser sacrificado. Las técnicas de entrenamiento implícito (IT) permite superar este inconveniente, ya que la secuencia de entrenamiento se embebe o combina en los datos que se transmiten sin consumir ancho de banda adicional [1], además de tener un rendimiento similar a los métodos de entrenamiento tradicional. Esto ha motivado que, parte de la investigación actual esté encaminada a la búsqueda de técnicas de IT novedosas o al refinamiento de las ya existentes, que permitan diseñar mejores algoritmos e incluirlos dentro de los sistemas de comunicación inalámbrica y así incrementar su desempeño.

1.2. Estado del arte

El primer trabajo en el que se consideró el uso de IT en un sistema de comunicaciones fue presentado por Farhang en [2]. Aquí se propone una técnica para estimar la respuesta al impulso de canales invariantes empleando un caso especial de IT llamado entrenamiento superpuesto (ST). Si bien el método básico es aclarado, nunca se expone un análisis de desempeño. Además, aunque se reconoce que para el caso de una aplicación práctica se necesitaría tener sincronía de secuencia de entrenamiento (TSS), la solución propuesta para lograrla es incorrecta. En esa misma línea, otras contribuciones aparecen en [3], donde se plantea la posibilidad de llevar a cabo la ecualización del canal directamente usando ST. Nuevamente no se expone ningún análisis de desempeño y el problema de sincronización tampoco es considerado. En [4] se presenta el primer análisis de desempeño para ST pero solo para el caso especial de una secuencia de entrenamiento compuesta de un tren de funciones delta equi-espaciadas. Otra vez la sincronización es dejada de lado. Cabe mencionar que en [3] y [4] se pasó por alto la contribución hecha en [2]. Varias aportaciones nuevas del tema se presentan en [5], por principios de cuenta el problema de sincronización es por primera vez abordado, analizado y resuelto empleando estadísticas de cuarto orden. El modelo empleado es más realista, ya que toma en cuenta el desplazamiento de CD que comúnmente aparece en el

receptor. Dicho problema es resuelto por medio de raíces polinomiales. Además se analiza el desempeño para cualquier secuencia periódica de entrenamiento y se identifica una familia de secuencias que por sus propiedades son adecuadas para la estimación de canal en ST. Posteriormente en [6] se presenta una alternativa basada en el procesamiento en el dominio de la frecuencia para estimar tanto el canal como el desplazamiento de CD. A pesar de que el método presentado posee un desempeño superior en la estimación del desplazamiento de CD y una complejidad computacional menor que el expuesto en [5], no toma en cuenta el problema de la sincronía de secuencia de entrenamiento.

Otras vertientes interesantes siguen a partir de [7]. En este trabajo, se introduce una nueva forma de ST, llamada **entrenamiento superimpuesto dependiente de los datos** (DDST) que tiene la propiedad de eliminar la interferencia que producen los datos al estimar el canal. Una nueva problemática adicional denominada sincronía de bloque (BS) debe ser considerada al usar DDST. En [8, 9, 10, 11] los problemas de desplazamiento de CD así como los problemas de sincronía para ST y DDST son analizados y resueltos desde un nuevo paradigma, dando lugar a métodos de estimación que tienen baja complejidad computacional y gran desempeño. Otra contribución, encaminada a eliminar la interferencia causada por los datos puede consultarse en [12], donde un precodificador lineal es aplicado a los datos del transmisor y la secuencia de entrenamiento es sumada ortogonalmente a la secuencia resultante. La estimación iterativa del canal y detección de los datos bajo el esquema de ST se consideran en [13]. Detalles concernientes a la asignación óptima de la potencia para los datos y entrenamiento se presentan en [14]. El uso de ST para acceso múltiple por división de código (CDMA) se considera en [15]. En [16], son aclaradas las implicaciones en la razón potencia pico a promedio (PAPR) de la señal OFDM al usar ST. Otras aplicaciones de ST en sistemas OFDM se estudian en [17, 18, 19, 20]. El uso de ST en conjunto con codificación espacio-temporal se aborda en [21]. Los detalles relativos al empleo de ST para sistemas con múltiples antenas transmisoras y múltiples antenas receptoras (sistemas MIMO) son expuestos en [22] y [23]. Una investigación concerniente a la mejora en la estimación de canal usando DDST y expansión en bases para un canal limitado en banda se aborda en [24]. El trabajo en [25] explora la posibilidad de usar ST para lograr la separación de paquete en el contexto de una red *ad-hoc*. Para solucionar el problema de desplazamiento de frecuencia (CFO), en [26] se propone la inclusión de una nueva secuencia de entrenamiento en DDST para facilitar su estimación. En [27], la técnica presentada para estimar el desplazamiento de frecuencia sólo hace uso de ST/DDST y el algoritmo se centra en restituir la periodicidad de la señal recibida. En [28, 29, 30, 31, 32], se investiga la estimación de canales doblemente selectivos. Las contribuciones en [33, 34] resuelven la problemática de la estimación de canales variantes en el tiempo en sistemas SISO empleando bases bidimensionales y ST. Una novedosa variante de ST, denominada *Iterative Mean Removal Superimposed Training* (MRST) [35, 36] ha demostrado su eficiencia en la estimación de canales en sistemas MIMO. De igual manera, en [37, 38] exploran el uso de ST en tales sistema. De igual forma en [39], se proponer un estimador de canales, linearmen-

te variables en el tiempo, para sistemas OFDM/MIMO basado en ST. El problema del incremento del PAPR en amplificadores de potencia prácticos cuando se usa DDST se aborda a detalle en [40].

Como lo sugieren los párrafos anteriores, a pesar de que la idea de incluir la técnica de ST en los sistemas de comunicaciones es nueva, su evolución ha sido creciente convirtiéndola hoy en día en un tópico de investigación activa alrededor del mundo.

1.3. Planteamiento del problema

Como puede desprenderse del estado del arte, la mayoría de las trabajos coinciden en que los sistemas de comunicaciones con ST/DDST tienen un rendimiento comparable a aquellos con entrenamiento explícito pero con la ventaja de un mejor uso del ancho de banda, lo cual hace factible su empleo en GSM y en difusión de video digital (DVB) [41]. A pesar de esta motivación y aunque la investigación asociada a ST/DDST es lo suficientemente madura desde el punto de vista de modelado matemático, aplicaciones y análisis de desempeño; son muy pocos los trabajos en la literatura que presentan algún tipo implementación física incluso a nivel experimental. Esto puede ser atribuido a la intensa carga computacional y la complejidad que implican sus algoritmos aunado también a su relativa novedad. Hasta el momento, los prototipos reportados están basados mayormente en software y emplean punto flotante para sus cálculos. Así, en [42] se describe un sistema de comunicación con DDST implementado bajo la filosofía *full-software*, programado en un procesador digital de señales (DSP). Por otro lado, en [43] la solución presentada es un sistema híbrido de *software–hardware* (co-diseño), centrado en un procesador, el cual ejecuta un programa que realiza la mayoría de las tareas y cuatro aceleradores en *hardware* para operaciones matemáticas complejas (co-procesadores), todos embebidos en un FPGA. Cabe resaltar, que ninguna de estas propuestas ha sido capaz de desarrollar velocidades de procesamiento requeridas por los estándares de comunicación actuales. Tampoco en ellas se exponen resultados concernientes a métricas que permitan evaluar su desempeño en términos de: (en los casos que aplique) funcionalidad, eficiencia en el área consumida y rendimiento en los cálculos de punto fijo. Por tal motivo, en este trabajo doctoral se pretende desarrollar arquitecturas de *hardware* en punto fijo para las etapas que conforman un sistema de comunicación con ST/DDST. Todas ellas concebidas bajo el enfoque *full-hardware*, lo que les permitirá ascender de nivel dentro de la jerarquía mostrada en la figura 1.3 y superar el problema de velocidad del que adolecen los prototipos antes mencionados. Por otro lado, se pondrá énfasis en el sistema transmisor y al bloque que corresponde al estimador de canal para ST/DDST. Cabe señalar que este último es pieza fundamental de cualquier receptor y su diseño es crucial ya que allanará el camino para una implementación práctica del concepto ST/DDST en los estándares de comunicaciones actuales.

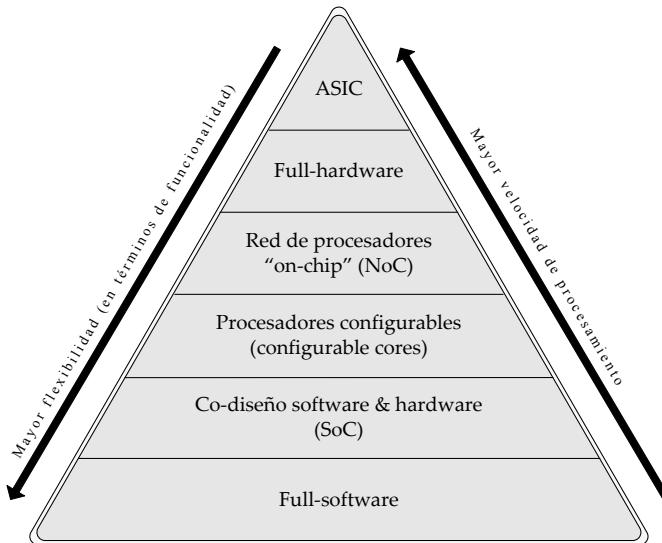


Figura 1.2: Niveles de jerarquía de los diversos enfoques existentes para implementar un sistema de procesamiento digital.

1.4. Objetivos de la tesis

1.4.1. Objetivo general

Establecer la pertinencia del uso de los algoritmos de entrenamiento implícito en los sistemas de comunicaciones de alto desempeño.

1.4.2. Objetivos particulares

- Analizar a profundidad los algoritmos usados en cada uno de los bloques que conforman un sistema de comunicación con ST/DDST, con el fin de modificar su ejecución mediante: técnicas de sistolización, transformaciones, factorizaciones o aproximaciones matemáticas, a partir de sus ecuaciones originales; que permitan hacer más eficiente, en términos de velocidad y consumo de área, su mapeo a una arquitectura de *hardware*.
- Diseñar e implementar en aritmética de punto fijo y bajo el enfoque *full-hardware* las arquitecturas de procesamiento digital de señales (PDS) de los algoritmos previamente analizados, considerando los siguientes preceptos:
 - Paralelismo.
 - Reconfigurabilidad
 - Reusabilidad
 - Portabilidad

- Verificar exhaustivamente las arquitecturas a través de simulaciones Monte Carlo que permitan evaluar su rendimiento bajo métricas de: consumo de área, funcionales y de desempeño; y comparando sus resultados con respecto a los obtenidos con los modelos de simulación en punto flotante.
- Proponer una arquitectura integral *full-hardware custom* para un sistema de comunicación ST/DDST operando bajo condiciones de falta sincronía y en presencia de desplazamiento de CD.

1.5. Organización de la tesis

El documento esta organizado en 6 capítulos y su contenido es el siguiente:

En el capítulo 1 se realiza una breve descripción de lo que es un sistema de comunicación y del concepto de entrenamiento implícito. Los beneficios de incluir la técnica IT en los sistema de comunicación, su evolución, el estado del arte que guarda y los problemáticas que no han sido superados por las implementaciones físicas existentes y que dan origen a los objetivos de esta tesis.

En el capítulo 2 se exponen las diversas técnicas de estimación del canal en los sistemas inalámbricos, así como su principio básico de operación. Posteriormente, se ofrece una descripción de las etapas que conforman un sistema de comunicaciones con ST, así como los procesos que influyen durante el flujo de datos entre el transmisor y el receptor.

En el capítulo 3 se proporciona los fundamentos matemáticos de los algoritmos de estimación para canales invariantes en el tiempo basados en ST/DDST, incluyendo aquel que usa conjuntamente proyección en bases. De igual forma, por la relevancia y complejidad que implican; los algoritmos para el cálculo eficiente de la DFT y correlación son también descritos.

En el capítulo 4 se describe a detalle el proceso por el cual los algoritmos y las ecuaciones, las cuales rigen el funcionamiento de un transmisor configurable ST/DDST, son trasladados a su respectivos módulos de hardware. Al final del capítulo se presentan los resultados obtenidos al evaluar la arquitectura de *hardware* del transmisor bajo métricas que permitan analizar su desempeño funcional así como también la eficiencia de su implementación.

En el capítulo 5 se detalla —de forma similar al anterior— el procedimiento mediante el cual los algoritmos de estimación de canal para un receptor ST/DDST son mapeados a *hardware*, así como también los correspondientes a la DFT y a la correlación. Posteriormente, se presentan los resultados relativos a su desempeño funcional y la eficiencia

de su implementación, para cada uno de ellos.

Finalmente, las diversas conclusiones extraídas de la tesis y las líneas de investigación abiertas para trabajos futuros relacionados con el tema se presentan en el capítulo 6.

1.6. Contribuciones de la tesis

- Se desarrollaron seis arquitecturas de procesadores para la transformada rápida de Fourier (FFT).
- Se propone la adaptación del algoritmo para la operación de correlación en el dominio de la frecuencia, que permite suprimir las operaciones de reordenamiento de las secuencias a procesar. Los pormenores del diseño e implementación en FPGA de dicho correlacionador han dado lugar a la publicación del artículo:

Romero-Aguirre, E.; Parra-Michel, R.; Longoria-Gandara, O.; Aguirre-Hernández, M., A Hardware-efficient frequency domain correlator architecture for acquisition stage in GPS, *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, vol. 2010, pp. 412-417, Dec. 2010.

- Se propone una arquitectura reconfigurable para un transmisor ST/DDST en banda base para modulación de amplitud en cuadratura (QAM) con constelaciones de tamaño 4, 16 y 64. Los detalles de su diseño e implementación se aceptaron para su publicación en el artículo de revista:

Romero-Aguirre, E.; Parra-Michel, R.; Carrasco-Alvarez, R.; Orozco-Lugo, A.G., Configurable transmitter and systolic channel estimator architectures for data-dependent superimposed training communications systems, *International Journal of Reconfigurable Computing (IJRC)*, 2012, Hindawi.

- Se propusieron dos arquitecturas *full-hardware* para estimadores de canal invariantes en el tiempo para un receptor ST/DDST con sincronía perfecta. En el apéndice A se muestra el artículo que se generó con esta contribución, el cual está referenciado como:

Romero-Aguirre, E.; Parra, R.; Orozco, A.G.; Carrasco-Alvarez, R., Full-hardware architectures for data-dependent superimposed training channel estimation, *Signal Processing Systems (SiPS), 2011 IEEE Workshop on*, vol. 2011, pp. 49-54, Oct. 2011.

- Se propone una adaptación del algoritmo de la media cíclica para su sistolización. A partir de ella, se desarrolló una arquitectura *full-hardware* basada en un arreglo sistólico de procesadores para estimar; ya sea la media cíclica o la respuesta al impulso de un canal (CIR) invariante en el tiempo, en un receptor ST/DDST con sincronía perfecta. Los resultados obtenidos de este trabajo se publicaron en:

Romero-Aguirre, E.; Parra-Michel, R.; Carrasco-Alvarez, R.; Orozco-Lugo, A.G., Architecture based on array processors for Data-dependent superimposed training channel estimation, *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, pp. 303-308, Nov.-Dec. 2011.

- Se desarrolló una arquitectura *full-hardware* de un estimador de canal para un receptor ST/DDST, en la cual se considera un escenario de operación más apegado a la realidad. En donde por principios de cuenta se estima/corrigió el retardo de sincronía entre el transmisor y el receptor, luego se estima el desplazamiento de CD para finalmente estimar la respuesta al impulso del canal. Los resultados obtenidos en este estimador fueron aceptados para su publicación en el siguiente artículo de revista indexada:

Romero-Aguirre, E.; Carrasco-Alvarez, R.; Parra-Michel, R.; Orozco-Lugo, A.; Mondragón-Torres, A., “Full-hardware architectures for data-dependent superimposed training channel estimation (extended version),” *Journal of Signal Processing Systems*, pp. 1-19, 10.1007/s11265-012-0706-2, Springer.

Vale la pena señalar que todas las arquitecturas digitales para ST/DDST que se desarrollaron en este trabajo de tesis, son las primeras que se reportan en la literatura desarrolladas bajo el enfoque *full-hardware*.

Capítulo 2

Sistema de comunicación con entrenamiento superimpuesto

2.1. Introducción

El estudio de un determinado sistema de comunicación es una tarea en extremo ardua que a menudo se facilita cuando el sistema es disociado en pequeñas tareas. El asunto estriba en que cada una estas tareas por lo regular resulta lo suficientemente compleja para requerir atención dedicada por parte de personas expertas en dicha área, pero a su vez tengan el conocimiento de todo su entorno de operación. Esto explica el hecho de que la mayoría de los estudios de investigación se centran en problemáticas muy puntuales de todo un sistema. Esto no excluye al sistema de comunicación con IT, el cual será objeto de estudio en este trabajo, donde se impone el reto de dominar tanto la operatividad de todo el sistema, como la interacción entre sus partes.

2.2. Clasificación de las técnicas de estimación de canal

Detección y estimación de señales son dos de los procesos fundamentales que debe realizar un sistema de comunicación. La teoría de estimación aborda el problema básico de inferir y extraer información (p. ej., amplitud, frecuencia, fase, tiempo de retardo o estadísticas de una señal) a partir de observaciones de señales que han sido corrompidas por el ruido y perturbaciones. Por otro lado, la teoría de detección, permite determinar la presencia o ausencia de señales específicas. Consecuentemente, la detección y la estimación forman una relación simbiótica, cada una requiere de la otra para producir algoritmos de procesamiento de alto desempeño. Lo anterior es evidente cuando estas dos tareas se aprovechan de manera conjunta e iterativa, con el consecuente resultado de un mejor desempeño del sistema de comunicación.

De igual forma, se ha señalado la relevancia de la estimación del canal en sistemas inalámbricos, ya que permite conocer las características del medio y así poder compensar las distorsiones que éste genera. Los métodos de estimación de canal en sistemas inalámbricos se pueden clasificar dependiendo en que tanto conoce el receptor acerca de los parámetros del transmisor y del canal de comunicación. En la figura 2.1 puede visualizarse una de las posibles formas de clasificación.

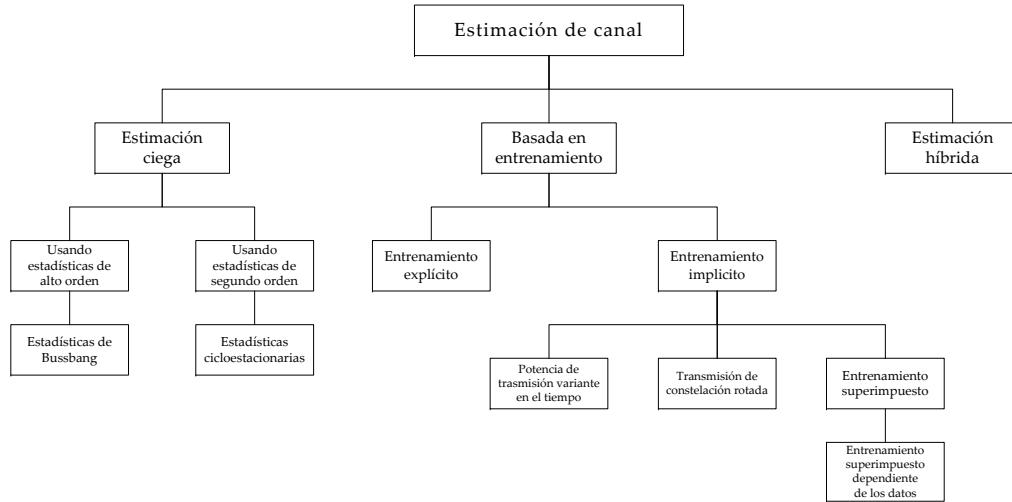


Figura 2.1: Taxonomía de las técnicas de estimación de canal.

2.2.1. Estimación ciega

Su principio de operación descansa en la estimación de los parámetros estadísticos que se conocen del canal a partir de la señal transmitida. Es recomendable cuando los recursos (potencia y ancho de banda) del sistema son escasos, cuando el tráfico de información es continuo o para una adquisición inicial de estado del canal. El problema radica en que dicha técnica tienen una complejidad computacional elevada —aún cuando se usen estadísticas de segundo orden— lo que la hace poco atractiva para ser incorporada al receptor.

2.2.2. Estimación basada en entrenamiento

En esta técnica se emplea una señal de referencia (entrenamiento o piloto) que es conocida por el receptor y a partir de las variaciones que sufre esta señal, el receptor sea capaz de conocer las distorsiones producidas por el canal. Este método es el que tiene mayor auge en los sistemas actuales y puede ser catalogado en dos grandes ramas: entrenamiento explícito y entrenamiento implícito.

2.2.3. Entrenamiento explícito (ET)

Inicialmente conocida como entrenamiento basado en símbolos piloto (PSAM) y más recientemente como transmisión asistida por pilotos (PAT), esta técnica consiste en alternar una señal piloto junto a los datos transmitidos [44]. Es el tipo de entrenamiento instaurado en los estándares GSM, TDMA, CDMA, HyperLAN2, entre otros. En la figura 2.2 se muestran varios ejemplos de transmisiones asistidas por pilotos, ahí se puede observar que los pilotos pueden ser colocados alternadamente en el dominio del tiempo y la frecuencia.

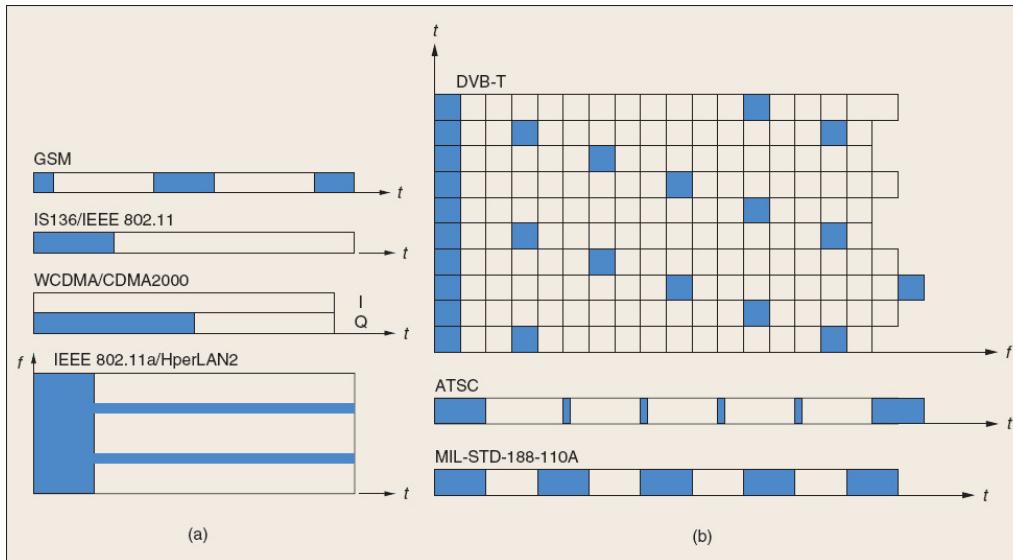


Figura 2.2: Posición de los pilotos de entrenamiento en sistemas inalámbricos. (a) transmisiones por paquetes y (b) transmisiones continuas. Las áreas sombreadas indican los pilotos (de [44] ©IEEE, 2004).

En [44, 45, 46, 47], se mencionan técnicas para la estimación de canales utilizando pilotos, en [47] se hace mención de la potencia y posición óptimos que deben tener los pilotos respecto a los datos transmitidos.

2.2.4. Entrenamiento implícito (IT)

En el entrenamiento implícito se transmite la señal piloto de manera oculta en los datos que se envían. Como el receptor conoce esta secuencia y la manera en que fue combinada, entonces puede extraer parámetros desconocidos del sistema. Además no compromete el uso de ancho de banda extra para transmitir la información de entrenamiento, tal y como ocurre en PAT. No obstante, IT conlleva un pequeño gasto de potencia la cual será asignada al entrenamiento. Existen varios caminos potenciales para ocultar un patrón de entrenamiento en un flujo de datos para entrenar el receptor de manera implícita, a continuación se describirán aquellos que son objeto de estudio de esta tesis

2.2.4.1. Entrenamiento superimpuesto (ST)

La aproximación más sencilla de IT se conoce entrenamiento superimpuesto donde la secuencia de entrenamiento es aritméticamente sumada a la secuencia de datos [2]. La figura 2.3 muestra el concepto descrito, donde cada dato contiene una pequeña porción de la secuencia de entrenamiento. En consecuencia, se cuenta con la ventaja de estar enviando continuamente información de entrenamiento al receptor sin consumir recursos adicionales de tiempo o frecuencia, pero con la penalidad de que parte de la potencia de los datos debe ser destinada al entrenamiento. Por otro lado, existe una perdida de ortogonalidad entre los datos y el entrenamiento, lo que ocasiona que los datos sean visualizados como ruido por el estimador y que el entrenamiento actúe como ruido en el proceso de detección de los datos.



Figura 2.3: Secuencia de datos con entrenamiento superimpuesto.

2.2.4.2. Entrenamiento superimpuesto dependiente de los datos (DDST)

Otra forma de IT es el entrenamiento superimpuesto dependiente de los datos, propuesto en [48]. Esta técnica, que es una variante de ST, tiene la particularidad remover la interferencia introducida por los datos sobre los símbolos de entrenamiento. Esto se logra con la ayuda de una tercera secuencia que se genera a partir de los datos que se transmiten y que es desconocida para el receptor. Tal como lo sugiere la figura 2.4, esta nueva secuencia también es superpuesta a la secuencia de datos y a la de entrenamiento. La secuencia dependiente de los datos (SDD), cancela las componentes de frecuencia en las cuales la secuencia de entrenamiento tiene energía. Por lo que, el rendimiento en la estimación del canal es notablemente mejorado. Por otro lado, DDST requiere de la inserción de un prefijo cíclico (CP) para operar adecuadamente, pero puede funcionar sin él [49, 50]. Asimismo, DDST resulta muy conveniente para operar en constelaciones de orden bajo, como lo es 4-QAM. Sin embargo, cuando se usa en constelaciones de alto orden, p. ej. 64-QAM, se tiene el problema en la detección de los datos (*data identifiability*), descrito a detalle en [51] y resuelto recientemente en [52].

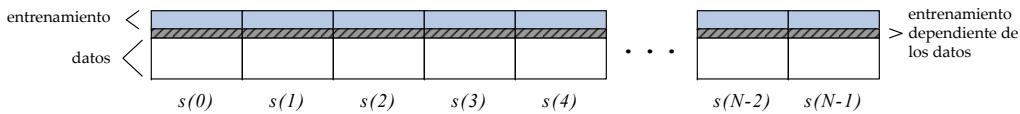


Figura 2.4: Secuencia de datos con entrenamiento superimpuesto dependiente de los datos.

2.3. Sistema de comunicación con entrenamiento implícito

En esta sección, se describirán las diferentes etapas que conforman un transmisor y un receptor con IT, así como las interacciones que existen entre ellas. También se considera que el sistema es de una entrada y una salida (SISO), y que la modulación digital que se usa es de tipo QAM. Cabe señalar, que a pesar de que las configuraciones propuestas para el transmisor y el receptor con IT no son las únicas posibles, éstas en particular son las que serán abordadas en esta tesis y por lo tanto serán usadas como referencia en lo subsecuente.

2.3.1. Transmisor con entrenamiento implícito

La figura 2.5 muestra el diagrama a bloques de un transmisor digital con IT. De manera general, su funcionamiento es el siguiente: un flujo de bits alimenta la entrada del conformador de símbolos, el cual va tomando bloques de $\log_2(M_{QAM})$ bits (donde M_{QAM} indica el orden de la modulación QAM), los agrupa y los envía en paralelo a su salida. Después, el mapeador (*mapper*) lee en su entrada el contenido de cada grupo de bits, calcula a qué punto de la constelación corresponde y a partir de dicho punto genera un símbolo complejo b . Debido a que el procesamiento en los sistemas con IT es por bloques de datos, la etapa de transformación de secuencia debe leer un bloque de N símbolos (secuencia de datos $b(k)$, donde $k = 0, \dots, N - 1$) y embeber en él la secuencia de entrenamiento, dando como resultado la secuencia compleja $s(k)$, que representa la señal en banda base que se va a transmitir. Una representación matricial general que define la transformación que sufre la señal para las posibles técnicas con IT está dada por

$$\mathbf{s} = \mathbf{Ab} + \mathbf{c}, \quad (2.1)$$

donde \mathbf{A} es una matriz de precodificación, \mathbf{b} es el vector de datos y \mathbf{c} es la secuencia de entrenamiento.

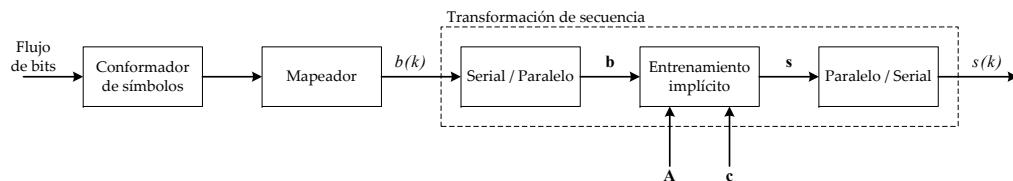


Figura 2.5: Diagrama a bloques de un transmisor digital con IT.

Es posible establecer el tipo de entrenamiento en el transmisor modificando la matriz de precodificación acorde a;

$$\text{Tipo de entrenamiento} = \begin{cases} \mathbf{A} = \mathbf{I}_N \text{ y } \mathbf{c} \neq \mathbf{0} & \text{para ST} \\ \mathbf{A} = \mathbf{I}_N - \mathbf{G} \text{ y } \mathbf{c} \neq \mathbf{0} & \text{para DDST} \end{cases} \quad (2.2)$$

con

$$\mathbf{G} = \frac{1}{N_P} (\mathbf{1}_{N_P \times 1} \otimes \mathbf{K}), \quad (2.3a)$$

y

$$\mathbf{K} = \mathbf{1}_{1 \times N_P} \otimes \mathbf{I}_P, \quad (2.3b)$$

donde \mathbf{I}_N es la matriz identidad de tamaño N , P es el periodo de la secuencia de entrenamiento, $N_P = N/P$, $\mathbf{1}_{N_P \times 1}$ y $\mathbf{1}_{1 \times N_P}$ son vector fila y vector columna cuyos elementos son todos unos, \mathbf{I}_P es la matriz identidad de tamaño P , \mathbf{G} and \mathbf{K} son matrices de tamaño $N \times N$ y $P \times N$ respectivamente y \otimes denota el producto Kronecker.

2.3.2. Receptor con entrenamiento implícito

El objetivo de cualquier receptor inalámbrico es el de ejecutar los algoritmos necesarios para contrarrestar las distorsiones introducidas por el canal. Algunas de tales distorsiones, representadas en el modelo de canal de la figura 2.6, son:

Desplazamiento de frecuencia (CFO)

Este fenómeno, representado en el modelo por el término $e^{j\omega t}$, se presenta debido a las variaciones de frecuencia ($\Delta\omega$) ocasionadas por la falta de osciladores perfectos y a cambios en la distancia entre el transmisor y receptor. Estas variaciones desplazan la señal original por $\cos \Delta\omega t$.

Desplazamiento de CD (DC-offset)

Este problema tiene su origen en las no-idealidades dentro de los circuitos electrónicos del sistema. Esto puede ser mitigado hasta cierto punto por medio de un buen diseño electrónico, pero no puede ser eliminado en su totalidad. El desplazamiento de CD, denotado en la figura 2.6 con la variable m , puede degradar significativamente el rendimiento del detector de datos [53].

Sincronización entre transmisor y receptor

Un problema de sincronía surge cuando entre el receptor y el transmisor, existe un desfase de muestras. Tal desfase, identificado con la variable τ , debe ser corregido porque en la realidad es imposible que el receptor pueda conocer el valor de τ antes de que inicie el procesamiento de la información que recibe. Este fenómeno puede visualizarse como la conjunción de dos problemas diferentes: sincronización de secuencia de entrenamiento (TSS) y sincronización de bloque o trama (BS). Si no existe TSS, se produce una ambigüedad al identificar el canal. Por otro lado, una falta de BS se traduce en un detrimiento en el rendimiento de la estimación del canal debido a que cada SDD se calcula para cada bloque y por ende es diferente para cada uno de ellos.

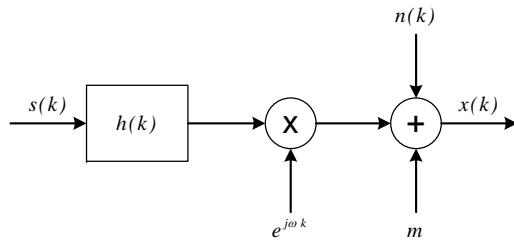


Figura 2.6: Modelo de canal inalámbrico y perturbaciones más comunes.

Otras perturbaciones

Además de los anteriores, existen otros fenómenos como la selectividad en frecuencia y el desplazamiento de fase, que están considerados dentro de la respuesta al impulso de canal $h(k)$ y el ruido aditivo Gaussiano que viene representado con el término $n(k)$.

Para afrontar las perturbaciones mencionadas, se ha adoptado la configuración mostrada en la figura 2.7 para el receptor con IT. Este es una versión modificada de la que se presenta en [1] y es capaz de operar tanto con ST como DDST. De inicio es posible notar que su complejidad es mucho mayor que la de su contraparte transmisora. Además, cada una de las tareas involucradas en su funcionamiento deben realizarse secuencialmente y en un orden específico. En principio se estima y compensa el CFO, luego en forma conjunta se lleva a cabo la estimación/compensación del desplazamiento de CD y corrección de sincronía. Una vez que estas tres tareas han sido completadas, el receptor está en condiciones de ejecutar la estimación de canal y la ecualización. Por último, se procede a la detección de los datos.

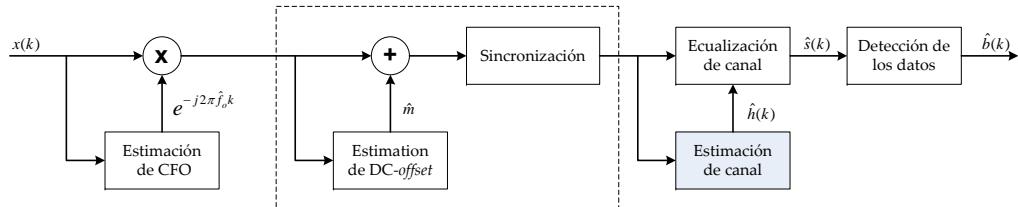


Figura 2.7: Diagrama a bloques de un receptor con IT.

2.4. Conclusiones del capítulo

De lo presentado en este capítulo se puede resaltar lo siguiente:

- La estimación de canal y la identificación de las perturbaciones introducidas en la señal, son parte fundamental en el desarrollo de todo receptor inalámbrico.

- Las técnicas de IT tales como ST y DDST, ofrecen una alternativa simple y eficiente en ancho de banda, para transmitir todo el tiempo datos y entrenamiento.
- La disposición de cada una de las etapas que conforman un transmisor y un receptor con IT, son parte crucial para su diseño.
- Las técnicas de ST y DDST son una alternativa viable a los métodos de ET usados actualmente, debido a que permiten incrementar las tasas de transferencias de datos.

Capítulo 3

Algoritmos de procesamiento digital de señales para un sistema de comunicación con entrenamiento implícito

3.1. Introducción

En el capítulo anterior se introdujo en forma descriptiva el panorama general de un sistema de comunicación con IT, haciendo énfasis en aquellos con ST/DDST. Se plantearon las configuraciones para las partes transmisoras y receptoras, y de forma descriptiva se definieron las tareas que deben ejecutar cada una de las etapas para su funcionamiento. Bajo ese mismo tenor, el presente capítulo se avocará a exponer desde un enfoque matemático, los algoritmos involucrados en cada de las etapas de un sistema de comunicación con ST/DDST, operando sobre un canal selectivo en frecuencia y asumiendo que es invariante sobre un bloque de datos, pero variante a través de varios de ellos.

3.2. Algoritmos para estimación de canales invariantes en el tiempo empleando ST/DDST

3.2.1. Modelo del sistema y consideraciones generales

Considere el modelo discreto y pasa–bajas del sistema de comunicación ST/DDST presentado en la figura 3.1, en el cual todas las señales están muestreadas al periodo de símbolo T , el índice k se usará para enumerar las muestras de estas señales. Por el momento se considera que se transmiten bloques de N datos, antecedidos por un prefijo

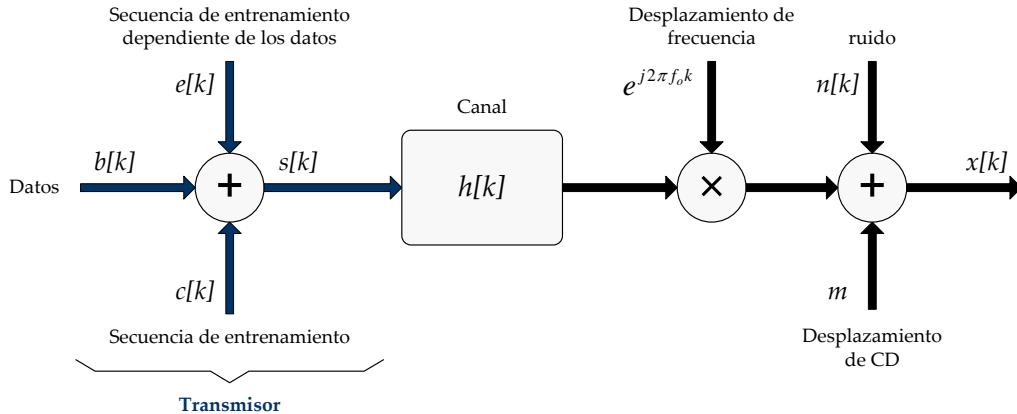


Figura 3.1: Modelo discreto de sistema de comunicación pasa-baja basado en DDST.

cíclico de tamaño P el cual es la repetición de los últimos P datos transmitidos del bloque de tamaño N . La señal $s[k]$ está conformada por la suma de la señal de datos $b[k]$, la secuencia de entrenamiento $c[k]$ y la secuencia de entrenamiento dependiente de los datos $e[k]$. $s[k]$ es transmitida por el canal de comunicación $h[k]$, el cual por el momento se considerara invariante en el tiempo y que puede ser visto como un filtro de respuesta finita al impulso (FIR) de M elementos diferentes de cero. Después de que la señal ha pasado por el canal, ésta es afectada por el CFO, el cual se modela como una modulación en amplitud de la señal recibida con una exponencial compleja de frecuencia f_o . La señal recibida es contaminada con ruido aditivo Gaussiano $n[k]$ y además se le añade el desplazamiento de CD m . Matemáticamente la señal recibida después de eliminar el prefijo cíclico es expresada como:

$$x[k] = e^{j2\pi f_o k} \left(\sum_{l=0}^{M-1} h[l] s[< k - \tau_s - l >_N] \right) + n[k] + m, \quad k = 0, \dots, N-1, \quad (3.1)$$

donde $\langle \cdot \rangle_N$ denota el operador módulo N , τ_s es el desfase en muestras entre el transmisor y el receptor; y $s[k]$ está dada por:

$$s[k] = b[k] + c[k] + e[k], \quad k = 0, \dots, N-1. \quad (3.2)$$

El objetivo es estimar $h[k]$, m , τ_s y f_o en función de la señal recibida $x[k]$ y el conocimiento previo de $c[k]$; y una vez con estos valores hacer la estimación de $b[k]$. Para poder realizar estas tareas, se harán las siguientes suposiciones en el modelo:

- (a) $b[k]$ es un proceso aleatorio, con media $E(b[k]) = 0$, varianza $E(|b[k]|^2) = \sigma_b^2$ y cuyos elementos son eventos independientes con la misma probabilidad de ocurrencia.
- (b) $c[k]$ es una secuencia que se obtiene al replicar N/P veces la secuencia de entrenamiento $c(n)$, la cual se obtiene como se indica en el apartado 3.2.10. Esta secuencia es periódica con periodo igual a P muestras, su valor promedio está dado por: $\bar{c} = \frac{1}{P} \sum_{n=0}^{P-1} c[n]$ y su potencia se define como $\sigma_c^2 = \frac{1}{P} \sum_{n=0}^{P-1} |c[n]|^2$.

- (c) $e[k]$ es una secuencia que se obtiene de replicar N_P veces la secuencia de entrenamiento dependiente de los datos con periodo P , la cual está dada por:

$$e[j] = -\frac{1}{N_P} \sum_{i=0}^{N_P-1} b(iP + j), \quad j = 0, 1, \dots, P-1. \quad (3.3)$$

- (d) $n[k]$ es un proceso aleatorio de media cero y varianza σ_n^2 .
- (e) $b[k]$ y $n[k]$ son estadísticamente independientes.
- (f) $\mathbf{C} = circ(\mathbf{c})$ es una matriz circulante* de tamaño $P \times P$ formada por el vector $\mathbf{c} = [c(0), c(1), \dots, c(P-1)]^T$ [54]. Además \mathbf{C} es de rango completo [5].
- (g) Se considera que N es un múltiplo entero de P .
- (h) Se establece que $P \geq 2M + 1$. Dado que no es posible determinar con precisión la duración (en muestras) del canal, se puede establecer el parámetro M como una cota superior dicho parámetro.
- (i) El desplazamiento de tiempo está contenido en el intervalo $0 \leq \tau_s \leq N-1$, donde $\tau_s := \tau_P P + \tau$ con $0 \leq \tau_P \leq N_P - 1$, $0 \leq \tau \leq P-1$ y $N_P = \frac{N}{P}$.

3.2.2. Estimación de canal considerando sincronía perfecta y ausencia desplazamiento de CD

Existen en el estado de arte varias propuestas para realizar la estimación de canal usando IT, no obstante el método que se detalla a continuación, esta fundamentado en las bases teóricas presentadas en [5]. Dicho método se apoya en el conocimiento de las estadísticas de la señal recibida (particularmente la media cíclica) para poder estimar la respuesta al impulso del canal de comunicación. A pesar de cuando, se usan estadísticas de primer orden no es posible despreciar la componente de CD provocada por no-idealidades en la electrónica de los receptores [5, 13], para propósitos de este análisis se supondrá que $m = 0$ y τ_s es conocido. La media cíclica de $x(k)$ está denotada por

$$y[j] := E[x[iP + j]] = \sum_{l=0}^{M-1} h[l]c[j-l]. \quad (3.4)$$

donde $j = 0, \dots, P-1$. Esta última ecuación representa un sistema de ecuaciones que contiene la información del canal. Es posible recuperar $h(k)$ sin ambigüedad si \mathbf{C} cumple con el inciso (f) de 3.2.1 y $P = M$. Aunque esta última condición pudiera no

*Una matriz circulante se define como: $circ([c[1], c[2], \dots, c[n]]^T) = \begin{pmatrix} c[1] & c[n] & \dots & c[2] \\ c[2] & c[1] & \dots & c[3] \\ \vdots & \vdots & \ddots & \vdots \\ c[n] & c[n-1] & \dots & c[1] \end{pmatrix}$

cumplirse en la práctica, es posible sustituir M por un límite superior. Con esta nueva consideración el sistema aún tiene solución única, donde los coeficientes del canal $h(M), h(M+1), \dots, h(P-1)$ serán cero. Despejando $h[l]$ de (3.4) y reescribiendo en forma matricial, entonces

$$\mathbf{h} = \mathbf{C}^{-1} \mathbf{y}, \quad (3.5)$$

donde $\mathbf{y} = [y[0], y[1], \dots, y[P-1]]^T$ y $\mathbf{h} = [h[0], h[1], \dots, h[P-1]]^T$. El algoritmo estima los valores de \mathbf{y} a partir de un bloque de N muestras recibidas [5] y si se cumple 3.2.1(g). El estimador de \mathbf{y} es

$$\hat{\mathbf{y}}[j] = \frac{1}{N_P} \sum_{i=0}^{N_P-1} x[iP + j], \quad j = 0, \dots, P-1 \quad (3.6)$$

donde $N_P = \frac{N}{P}$. En consecuencia, 3.5 llega a ser

$$\hat{\mathbf{h}} = \mathbf{C}^{-1} \hat{\mathbf{y}}, \quad (3.7)$$

donde ahora $\hat{\mathbf{y}} = [\hat{y}[0], \hat{y}[1], \dots, \hat{y}[P-1]]^T$ y $\hat{\mathbf{h}} = [\hat{h}[0], \hat{h}[1], \dots, \hat{h}[P-1]]^T$.

3.2.3. Estimación de canal bajo sincronía perfecta y con desplazamiento de CD

A diferencia del caso anterior, ahora si se considera el desplazamiento de CD que se introduce en el receptor y se asume que por el momento que se tiene TSS y BS ($\tau_s = 0$), así la media cíclica de la señal recibida queda expresada como:

$$y[j] := E[x[iP + j]] = \sum_{l=0}^{M-1} h[l]c[j-l] + m, \quad (3.8)$$

donde $j = 0, \dots, P-1$, representando en forma matricial (3.8) se tiene:

$$\mathbf{y} = \mathbf{Ch} + \mathbf{m}, \quad (3.9)$$

donde $\dot{\mathbf{h}} = [\mathbf{h}^T \mathbf{0}_{1 \times (P-M)}]^T$ es un vector de dimensión P cuyos primeros M elementos son las muestras del canal $[\mathbf{h}]_k = h[k]$ y los elementos restantes son ceros, \mathbf{C} es una matriz circulante similar a la que se define en el inciso (f) del apartado 3.2.1, $\mathbf{y} = [y[0], y[1], \dots, y[P-1]]$, $\mathbf{m} = m\mathbf{1}_{P \times 1}$ y $\mathbf{1}_{P \times 1}$ es un vector columna de tamaño P cuyos elementos son iguales a 1; debido a las propiedades de la matriz \mathbf{C} es fácil comprobar que $\mathbf{C}\mathbf{1}_{P \times 1} = P\bar{c}\mathbf{1}_{P \times 1}$, de esta manera después de algunas manipulaciones matemáticas la ecuación (3.9) puede ser reescrita como:

$$\mathbf{y} = \mathbf{C}(\dot{\mathbf{h}} + \tilde{\mathbf{m}}), \quad (3.10)$$

donde $\tilde{\mathbf{m}} = \tilde{m}\mathbf{1}_{P \times 1}$ y $\tilde{m} = \frac{m}{P\bar{c}}$. Dado que \mathbf{C} es de rango completo, el valor de $\dot{\mathbf{h}}$ está dado por:

$$\dot{\mathbf{h}} = \mathbf{C}^{-1} \mathbf{y} - \tilde{\mathbf{m}}. \quad (3.11)$$

Para llevar a cabo la implementación, es necesario tener un estimado de la media cíclica \mathbf{y} , en [5] proponen el uso del siguiente estimador:

$$\hat{\mathbf{y}}[j] = \frac{1}{N_P} \sum_{i=0}^{N_P-1} x[iP + j], \quad j = 0, \dots, P-1 \quad (3.12)$$

donde $N_P = \frac{N}{P}$, lo cual en el dominio de la frecuencia es equivalente a estimar los P componentes de frecuencia diferentes de cero de la señal periódica $x[k]$. Así, usando (3.11) y (3.12) se tiene que el estimado del canal está dado por:

$$\hat{\mathbf{h}} = \mathbf{C}^{-1}\hat{\mathbf{y}} - \tilde{\mathbf{m}}, \quad (3.13)$$

donde $\hat{\mathbf{y}} = [\hat{\mathbf{y}}[0], \hat{\mathbf{y}}[1], \dots, \hat{\mathbf{y}}[P-1]]^T$ y $\tilde{\mathbf{m}}$ es un estimado de \mathbf{m} .

3.2.4. Análisis de desempeño

Para analizar el desempeño de un estimador es necesario conocer si su valor esperado difiere con el valor de la variable estimada y cual es la variación de dicho estimador. Partiendo de las ecuaciones (3.8) y (3.12) es posible observar que:

$$E(\hat{\mathbf{y}}[j]) = \frac{1}{N_P} \sum_{i=0}^{N_P-1} E(x[iP + j]) = \frac{1}{N_P} \sum_{i=0}^{N_P-1} y[j] = y[j], \quad (3.14)$$

lo cual indica que $E(\hat{\mathbf{y}}) = \mathbf{y}$, al calcular el valor esperado de (3.13) y empleando (3.11) se tiene que:

$$E(\hat{\mathbf{h}}) = \mathbf{E}(\mathbf{C}^{-1}\hat{\mathbf{y}} - \tilde{\mathbf{m}}) = \mathbf{C}^{-1}E(\hat{\mathbf{y}}) - \tilde{\mathbf{m}} = \mathbf{C}^{-1}\mathbf{y} - \tilde{\mathbf{m}} = \dot{\mathbf{h}}, \quad (3.15)$$

lo anterior nos indica que el valor esperado en la estimación de canal propuesta es igual al valor del canal original.

El siguiente paso es analizar la variación del error entre el canal estimado y el canal original, para tal fin se propone la siguiente métrica de error:

$$\epsilon = \|\hat{\mathbf{h}} - \dot{\mathbf{h}}\|, \quad (3.16)$$

quedando definida la varianza de este error como:

$$\begin{aligned} var(\epsilon) &= tr(E((\hat{\mathbf{h}} - \dot{\mathbf{h}})(\hat{\mathbf{h}} - \dot{\mathbf{h}})^T)) = \\ &= tr(E((\mathbf{C}^{-1}(\hat{\mathbf{y}} - \mathbf{y}))(\mathbf{C}^{-1}(\hat{\mathbf{y}} - \mathbf{y}))^H)) = \\ &= tr(\mathbf{C}^{-1}E((\hat{\mathbf{y}} - \mathbf{y})(\hat{\mathbf{y}} - \mathbf{y})^H)\mathbf{C}^{-H}) = tr(\mathbf{C}^{-H}\mathbf{C}^{-1}\mathbf{R}_{\hat{\mathbf{y}}}), \end{aligned} \quad (3.17)$$

donde $\mathbf{R}_{\hat{\mathbf{y}}} = \frac{1}{N_P}(\mu\sigma_b^2\sigma_h^2 + \sigma_n^2)\mathbf{I}_P$ es la matriz de covarianza de $\hat{\mathbf{y}}$, σ_h^2 es la ganancia del canal y μ es 1 para ST y 0 para DDST. Debido a las propiedades de la matriz \mathbf{C} se cumple que $\mathbf{C}\mathbf{C}^H = P\sigma_c^2\mathbf{I}$, de esta manera (3.17) queda reducido a:

$$var(\epsilon) = \frac{P}{N} \frac{\mu\sigma_b^2\sigma_h^2 + \sigma_n^2}{\sigma_c^2}. \quad (3.18)$$

3.2.5. Estimación de CFO

Para contrarrestar el efecto del CFO en la señal recibida, se ha tomado como referencia el trabajo propuesto en [27, 55], ya que su implementación es más sencilla comparado con trabajos publicados anteriormente. El autor en [27], propone tres métodos heurísticos para estimar el CFO de manera burda, de los cuales se ha optado por el método de maximización de componentes de frecuencia (MFB) por su baja complejidad computacional. Para explicar este algoritmo, se considera que por el momento que la señal recibida consiste solamente de la secuencia de entrenamiento $c[k]$, esto es: $n[k] = 0$, $b[k] = 0$, $e[k] = 0$ y $m = 0$; así la señal recibida queda expresada como:

$$x[k] = e^{j2\pi f_o k} r[k], \quad (3.19)$$

donde $r[k] = \sum_{l=0}^{M-1} h[l]c[l - k]$, $f_o = \Delta f T$, T es el periodo de símbolo y Δf es el desplazamiento de frecuencia. Si se conociera f_o , sería posible restaurar $r[k]$ de la siguiente manera:

$$z[k] = e^{-j2\pi f_o k} x[k]. \quad (3.20)$$

Dado que $r[k]$ es una señal periódica de periodo P , su espectro de frecuencia tendrá solamente P elementos diferentes de cero, con lo que el espectro de la señal $z(k)$ presentará los P elementos de $r(k)$ pero desplazados cíclicamente. Basado en lo anterior, el objetivo del algoritmo MFB es el de encontrar un estimado \hat{f}_o tal que restaure a su posición original las P componentes de frecuencia. Lo anterior se consigue al maximizar la siguiente función de costo:

$$J_{MFB} = \sum_{k \in S} |Z[k]|^2, \quad (3.21)$$

donde S representa un conjunto cuyos elementos son determinados por $i\frac{N}{P}$ con $i = 0, \dots, P - 1$ y $Z[k]$ es la DFT de $z[k]$. El algoritmo MFB consiste de dos etapas, la primera es una búsqueda burda del CFO y la segunda etapa consiste en una búsqueda fina la cual concluye cuando el error entre \hat{f}_o y f_o no sobrepase un umbral preestablecido. Para realizar la búsqueda burda primero hay que calcular:

$$W[k] = |X[k]|^2, \quad (3.22)$$

donde $X[k]$ es la DFT de $x[k]$. Una vez obtenidos los valores de (3.22), se construye una matriz de dimensión $N_{PD} \times P$ donde $N_{PD} = \frac{N_D}{P}$ y N_D es la longitud de $X[k]$:

$$[\mathbf{A}]_{i,j} = W[iP + j], \quad (3.23)$$

donde $i = 0, \dots, N_{PD}$ y $j = 0, \dots, P - 1$. Luego, se suman por columnas los elementos de la matriz \mathbf{A} y se selecciona el índice i de la columna cuya suma sea mayor de todas para estimar el CFO bajo el siguiente criterio:

$$\hat{f}_b^0 = \begin{cases} \frac{i}{N_D}, & i \leq \frac{N_{PD}}{2} \\ \frac{i}{N_D} - \frac{1}{P}, & \text{cualquier otro} \end{cases} \quad (3.24)$$

Para realizar la búsqueda fina iterativa, se emplea el valor de \hat{f}_b^0 para evaluar:

$$\begin{aligned}\hat{f}_{oc} &= \hat{f}_b^{iter-1} \\ \hat{f}_{oi} &= \hat{f}_{oc} - 2^{\frac{-iter}{ND}} \\ \hat{f}_{od} &= \hat{f}_{oc} + 2^{\frac{-iter}{ND}},\end{aligned}\tag{3.25}$$

donde $iter$ señala el índice de iteración, el cual se inicializa con uno. Utilizando los valores de (3.25) se procede al cálculo de:

$$\begin{aligned}z_{oc}[k] &= e^{-j2\pi\hat{f}_{oc}k}x[k] \\ z_{oi}[k] &= e^{-j2\pi\hat{f}_{oi}k}x[k] \\ z_{od}[k] &= e^{-j2\pi\hat{f}_{od}k}x[k],\end{aligned}\tag{3.26}$$

los valores conseguidos con (3.26) sirven para evaluar la función de costo 3.21, así:

$$\begin{aligned}J_c &= \sum_{i=0}^{P-1} |Z_{oc}[iN_{PD}]|^2 \\ J_i &= \sum_{i=0}^{P-1} |Z_{oi}[iN_{PD}]|^2 \\ J_d &= \sum_{i=0}^{P-1} |Z_{od}[iN_{PD}]|^2,\end{aligned}\tag{3.27}$$

de los resultados obtenidos en (3.27), se elige aquel que presente el mayor valor y su valor de frecuencia estimado $\hat{f}_{o(.)}$ se asigna al valor de \hat{f}_b^{iter} en la siguiente iteración.

3.2.6. Algoritmo para sincronización de secuencia de entrenamiento y sincronización de bloque

Una falta de sincronía con la secuencia de entrenamiento ($\tau \neq 0$), produce que el estimado de la media cíclica (3.12) presente una permutación cíclica, i.e. $\mathbf{P}_\tau \hat{\mathbf{y}}$ donde \mathbf{P} es una matriz de permutación cíclica [56] que depende del desfasamiento τ que existe entre el transmisor y el receptor. Así, el objetivo del algoritmo de sincronía es encontrar el valor estimado de τ tal que $\hat{\mathbf{y}}$ no presente ninguna permutación. Para lograr el objetivo antes mencionado, se utiliza el algoritmo presentado en [8] el cual se fundamenta en proyecciones en diferentes subespacios vectoriales. Tomando en consideración la ecuación (3.13) con $m \neq 0$ y $\tau \neq 0$ se obtiene:

$$\mathbf{C}^{-1}\mathbf{y} = \mathbf{P}_\tau(\dot{\mathbf{h}} + \tilde{\mathbf{m}}),\tag{3.28}$$

donde $\tilde{\mathbf{m}}$ es un vector con todos sus elementos iguales a $\tilde{m} = \frac{m}{Pc}$. Nótese que en el caso de que $\tau = 0$ entonces $\mathbf{P}_0 = \mathbf{I}$, donde \mathbf{I} es la matriz identidad y los $P - M$ últimos elementos del vector $\dot{\mathbf{h}} + \tilde{\mathbf{m}}$ serían iguales a \tilde{m} . Observando lo anterior, el algoritmo trata de encontrar la permutación donde se cumple la condición antes mencionada, lo cual asegura la existencia de sincronía con la secuencia de entrenamiento. Matemáticamente se debe encontrar el mínimo de la siguiente función de costo:

$$\hat{\tau} = \arg \min_{\tau} \left\| \mathbf{V} \left(circshift_{\tau}(\mathbf{C}^{-1}) \hat{\mathbf{y}} \right) \right\|_{[P-M]}, \quad 0 \leq \tau \leq P - 1,\tag{3.29}$$

con

$$\mathbf{V} = \mathbf{I}_{(P-M)} - \frac{1}{P-M} \mathbf{1}_{(P-M) \times (P-M)},$$

donde $\mathbf{I}_{(P-M)}$ es la matriz identidad de tamaño $P - M$, $\mathbf{1}_{(P-M) \times (P-M)}$ es una matriz de tamaño $(P - M) \times (P - M)$ cuyos elementos son todos unos. También, se ha adoptado la nomenclatura de que para cualquier matriz \mathbf{A} , entonces $\mathbf{A}^{[n]}$ y $\mathbf{A}_{[n]}$ corresponden a los primeras y últimas n filas de la matriz \mathbf{A} , mientras que el operador $\text{circshift}_n(\mathbf{A})$ desplaza circularmente n posiciones hacia abajo los renglones de la matriz \mathbf{A} .

Como se estableció con antelación, la sincronía de bloque es requerida cuando se utiliza el algoritmo de DDST para estimar el canal de comunicación. Para explicar el porqué la secuencia de entrenamiento dependiente de los datos elimina la interferencia que producen los datos, considere por el momento que no se está utilizando tal señal dependiente de los datos, con lo que el espectro de frecuencia de la señal recibida será:

$$X[k] = H[k](C[k] + B[k]) + N[k], \quad (3.30)$$

donde $H[k]$, $C[k]$, $B[k]$ y $N[k]$ son los espectros de frecuencia del canal de comunicación, la señal de entrenamiento, los datos y el ruido respectivamente. Dado que $c[k]$ es periódica, su espectro va a presentar solamente P componentes equiespaciadas diferentes de cero, mientras que $b[k]$ y $n[k]$ —por las suposiciones establecidas en el planteamiento matemático— contendrán componentes en todo su espectro de frecuencia. Como se vió en la sección 3.2.3, para llevar a cabo la estimación del canal, es necesario estimar los P componentes de la señal periódica recibida, pero al tener presencia de los datos y del ruido estas interferirán en la estimación. El efecto entonces de la señal DDST es hacer cero aquellas componentes de frecuencia de $B[k]$, las cuales están ubicadas en las mismas posiciones de las componentes de la señal periódica, esto se consigue añadiendo a la señal transmitida la siguiente señal de interferencia controlada:

$$e[k] = -\frac{1}{N_P} \sum_{i=0}^{N_P-1} b[iP + k], \quad (3.31)$$

donde $k = 0, \dots, P - 1$, con lo anterior se asegura que la interferencia introducida por los datos durante la estimación del canal sea completamente eliminada, pero implica estar sincronizado con el bloque con el cual se calcula la señal $e[k]$. Para lograr dicha sincronía de bloque, se ha optado usar el algoritmo presentado en [8] en el cual minimiza la siguiente función de error, una vez que se ha encontrado la sincronía de secuencia de entrenamiento:

$$\hat{\tau}_P = \arg \min_{\tau_P} \left\| \mathbf{V} \left(\mathbf{C}^{-1} \hat{\mathbf{y}}(\tau_p, \tau) \right)_{[P-M]} \right\|, \quad 0 \leq \tau_P \leq N_P - 1, \quad (3.32)$$

donde el vector $\hat{\mathbf{y}}(\tau_p, \tau)$ es definido como:

$$[\hat{\mathbf{y}}(\tau_p, \tau)]_j = \frac{1}{N_P} \sum_{i=0}^{N_P-1} x[iP + j + \tau_P P + \tau], \quad 0 \leq j \leq P - 1. \quad (3.33)$$

3.2.7. Estimación del desplazamiento de CD y estimación de canal

Una vez estimado el retardo $\tau_s = \tau_P P + \tau$, los parámetros del canal y el desplazamiento de CD pueden ser calculados aplicando la formula (3.13), i.e.

$$\hat{\mathbf{h}} = \mathbf{C}^{-1} \hat{\mathbf{y}}(\hat{\tau}_P, \hat{\tau}), \quad (3.34)$$

de aquí los $P - M$ últimos elementos de $\hat{\mathbf{h}}$ contienen la información de m , el cual puede ser obtenido como:

$$\hat{m} = \frac{P\bar{c}}{P - M} \sum_{i=M}^{P-1} [\hat{\mathbf{h}}]_i, \quad (3.35)$$

y los parámetros del canal pueden ser calculados como:

$$[\hat{\mathbf{h}}]_i = [\mathbf{C}^{-1} (\hat{\mathbf{y}}(\hat{\tau}_P, \hat{\tau}) - \hat{\mathbf{m}})]_i, \quad i = 0, 1, \dots, M - 1 \quad (3.36)$$

donde $\hat{\mathbf{m}}$ es un vector de tamaño P donde todos sus elementos son iguales a \hat{m} . Cabe señalar que \hat{m} se encuentra normalizada por un factor dado por el cociente de la media \bar{c} y la potencia σ_c^2 de la secuencia de entrenamiento

3.2.8. Ecualización

Toda vez corregido el CFO, el desplazamiento de CD y habiendo estimado los parámetros del canal, es necesario compensar las distorsiones que produce este en los datos transmitidos, a este proceso se llama ecualización. La ecualización puede ser realizada tanto en el dominio del tiempo, como en el dominio de la frecuencia. En este caso se emplea la ecualización en el dominio de la frecuencia [57]. Para llevarla a cabo, primero es necesario eliminar la secuencia de entrenamiento $c[k]$ de la señal recibida, esto se puede hacer de manera sencilla, al igualar a cero los P componentes de frecuencia correspondientes a la secuencia de entrenamiento.

En forma matricial, la ecualización puede ser expresada como:

$$\mathbf{u} = \mathbf{F}_N \mathbf{G} \tilde{\mathbf{z}}, \quad (3.37)$$

donde \mathbf{u} es un vector de longitud N que contiene la señal ecualizada, \mathbf{F}_N es una matriz de dimensión $N \times N$ que realiza la transformada de Fourier y cuyos elementos están dados por $[\mathbf{F}_N]_{m,n} = \frac{1}{\sqrt{N}} e^{-j2\pi mn/N}$, $[\tilde{\mathbf{z}}]_k = s[k] - c[k]$ y \mathbf{G} es una matriz diagonal de dimensión $N \times N$ donde sus elementos están definidos de la siguiente manera:

$$[\mathbf{G}]_{k,k} = \begin{cases} \frac{1}{\hat{H}[k]}, & \text{ecualización forzada a cero} \\ \frac{\hat{H}[k]}{|\hat{H}[k]|^2 + \check{\sigma}_n^2} - \frac{1}{P}, & \text{ecualización MMSE} \end{cases} \quad (3.38)$$

donde $\check{\sigma}_n^2 = \sigma_n^2(1 - \frac{1}{N_P})$; y $\hat{H}[k]$ es la DFT de tamaño N del canal estimado.

3.2.9. Estimación iterativa (detección) de los datos

La señal ecualizada \mathbf{u} obtenida de la sección anterior, es utilizada para estimar los datos transmitidos. Para llevar a cabo esta tarea se utiliza el algoritmo propuesto en [48] el cual de manera iterativa estima los posibles datos transmitidos a pesar de la distorsión producida por la secuencia de entrenamiento dependiente de los datos. El algoritmo es inicializado de esta forma:

$$\bar{\mathbf{b}}^{(0)} = \lfloor \mathbf{u} \rfloor, \quad (3.39)$$

donde $\lfloor \mathbf{u} \rfloor$ asigna a cada elemento del vector \mathbf{u} el punto mas cercano de la constelación con la cual se transmitió. El algoritmo iterativo es planteado de la siguiente manera:

$$\bar{\mathbf{b}}^{(i)} = \lfloor \mathbf{u} + \mathbf{J}\bar{\mathbf{b}}^{(i-1)} \rfloor, \quad (3.40)$$

donde $\mathbf{J} = \frac{1}{N_p} \mathbf{1}_{N_p} \otimes \mathbf{I}_P$, \otimes denota el producto Kronecker de dos matrices e i indica el índice de iteración. Este proceso es realizado cuantas veces desee el diseñador, sin embargo en [48], se demuestra por medio de simulaciones que con una sola iteración es suficiente para obtener una mejoría significante.

Otra opción es emplear el algoritmo iterativo propuesto en [51], el cual realiza una mejor estimación de la distorsión introducida por la secuencia de entrenamiento dependiente de los datos, lo que se traduce en una mejor detección de los datos transmitidos y por lo tanto en un mejor desempeño que el mostrado en [48].

3.2.10. Diseño de la secuencia de entrenamiento

La secuencia de entrenamiento $c(n)$ es pieza clave en la tareas de estimación. Sus propiedades determinan el desempeño que un estimador pueda tener. A continuación de exponen las propiedades deseables en $c(n)$ para un rendimiento eficiente de los estimadores ST/DDST.

En [5] se deduce que el estimador de canal es independiente de las características del canal si los coeficientes de la DFT de $c(n)$ son constantes. Por otro lado, la secuencia de entrenamiento impacta también en el algoritmo para encontrar TSS, ya que como se mencionó en la sección 3.2.6 éste se basa en las permutaciones cíclicas de $\mathbf{C}^{-1}\mathbf{y}$. La $c(n)$ se diseña de manera que la potencia del error de estimación $\mathbf{C}^{-1}\mathbf{y}$ sea mínima. Esto se logra si los coeficientes de la DFT de $c(n)$ son iguales. Debido a que el algoritmo para BS está en función de TSS, el objetivo es mejorar la estimación de TSS para que en consecuencia el desempeño de BS se incremente. Otra característica deseable en $c(n)$ es que su relación pico a potencia promedio sea lo más cercana o igual a uno, esto para evitar las no linealidades en el transmisor. En [5] se presenta un procedimiento

—mismo que se usa en este trabajo— para encontrar $c(n)$ ^{**} con su DFT constante y una relación pico a potencia a promedio óptima (lo más cercana a uno). Estas secuencias son conocidas como secuencias OCI (*optimum channel independent*) y se obtienen de la siguiente forma:

$$c(n) = \sigma_c e^{j\frac{\pi}{P}(n(n+v))} \quad \text{con} \quad \begin{cases} v = 1, & \text{si } P \text{ es impar} \\ v = 2, & \text{si } P \text{ es par} \end{cases} \quad (3.41)$$

donde $n = 0, 1, \dots, P - 1$.

3.3. Algoritmo de estimación de canales invariantes usando proyección en bases y ST

Como se puede inferir, la tarea de estimar el canal se complica conforme más dispersivo y/o variante se vuelva éste. Para abordar esto, varios autores han considerado la técnica de proyección en bases para modelar y estimar el canal de comunicación. Por tal motivo, en esta sección se procederá a exponer la manera de estimar un canal de comunicación invariante empleando proyección en bases en conjunto con la técnica de entrenamiento superpuesto. Esta idea fue desarrollada por primera vez en [24] y esta sección es un compendio de lo que en él se presenta.

3.3.1. Proyección en bases

Dado una señal $p \in P$ cualquiera, existen diferentes maneras de representarla. Cuando dicha señal se representa como la suma ponderada de las funciones pertenecientes al conjunto $\Theta \subset P$, las cuales se suponen que son linealmente independientes entre si y la combinación lineal de estas puede generar cualquier señal perteneciente a P , entonces se dice que la señal está siendo expandida en la base θ . La utilidad de representar las señales de esta manera, radica de la facilidad que puede proveer para visualizar ciertas características de interés o para facilitar y mejorar el proceso de estimación de las mismas. Algunos ejemplos de expansión en bases son la representación de una señal definida en el dominio del tiempo mediante su transformada de Fourier o la expansión dicha señal mediante polinomios. Durante este documento supondremos que los elementos de la base θ son ortonormales entre si, es decir, una vez definido un producto interno, el producto entre elementos diferentes de la base será igual a cero mientras que el producto de un elemento de la base consigo mismo da como resultado uno.

Puesto que en este trabajo se utilizan procesos estocásticos, sería deseable poder representar estos procesos mediante expansión en bases. Para lograr este objetivo es posible usar la expansión de Karhunen-Loëve, la cual provee una caracterización del proceso estocástico en función de la suma ponderada por variables aleatorias decorrelacionadas

^{**}Cabe señalar que esta secuencia es diferente a la secuencia $c(k)$, ya que esta última se construye replicando N_P veces la secuencia OCI $c(n)$ para alcanzar la misma longitud N que la de datos

de los elementos de una base [58]. Para esto, es necesario conocer la función (matriz) de covarianza, la cual permite calcular los elementos de la base a usar. Esto a menudo es complicado debido que las características del proceso estocástico a analizar cambian con respecto al tiempo, aunado a lo complejidad computacional del método. Por tal motivo, es necesario otro tipo de bases tales que puedan construirse a partir de valores que puedan ser conocidos *a priori* acerca del proceso y facilite el cálculo de las mismas. Una opción para anterior son las bases universales.

3.3.2. Base universal

La base universal (UB), consiste en un conjunto de secuencias limitadas en tiempo y en frecuencia, cuyo espectro de frecuencia depende de la respuesta al impulso de un filtro dado. Estas bases fueron propuestas por primera vez en [59] y en [60] para modelar canales invariantes en el tiempo en función del filtro formador del transmisor y el filtro acoplado del receptor teniendo como objetivo reducir el numero de parámetros a estimar. La manera de obtener estas bases es la siguiente: Sea $h(\tau)$ el equivalente pasa bajas de la respuesta al impulso del canal de comunicación en un instante de tiempo t dado, el cual es un proceso estocástico no estacionario en la variable τ y es definido por:

$$h(\tau) = \int_{-\infty}^{\infty} c(\tau - v)g(v)dv, \quad (3.42)$$

donde $c(\tau)$ representa la respuesta al impulso del medio de propagación y $g(\tau)$ es la respuesta al impulso combinada del filtro formador del transmisor y el filtro acoplado del receptor. Dado que $h(\tau)$ no es estacionario, su función de autocorrelación dependerá de dos variables y es definida como:

$$R(\tau, \epsilon) = E(h(\tau)h^*(\epsilon)), \quad (3.43)$$

si se considera que el medio de propagación consiste de un perfil de potencia de retardo (PDP) discreto, es decir, que las múltiples trayectorias pueden modelarse como la suma ponderada de pulsos de Dirac:

$$c(\tau) = \sum_{i=1}^M \alpha_i \delta(\tau - \gamma_i), \quad (3.44)$$

donde α_i es el peso de la i -ésima trayectoria el cual es una variable aleatoria compleja con distribución Gaussiana, M es la cantidad de trayectorias del medio de transmisión y γ_i es el tiempo de arribo de la i -ésima trayectoria. Sustituyendo (3.44) en (3.42) y calculando (3.43) se tiene:

$$R(\tau, \epsilon) = \sum_{i=1}^M \sigma_i^2 g(\tau - \gamma_i)g(\epsilon - \gamma_i), \quad (3.45)$$

donde $\sigma_i^2 = E(\alpha_i \alpha_i^*)$. Basados en el teorema de Karhunen-Loëve, el proceso estocástico $h(\tau)$, puede ser representado mediante la suma ponderada de funciones ortogonales, las

cuales son las funciones propias de su función de autocorrelación $R(\tau, \epsilon)$. Dado que es de interés la implementación discreta de estos algoritmos, se puede muestrear la función de autocorrelación en los instantes de tiempo:

$$\tau_k = k \frac{T}{L} + \tau_0, \quad k = 0, 1, \dots, LN - 1, \quad (3.46)$$

$$\epsilon_k = k \frac{T}{L} + \epsilon_0, \quad k = 0, 1, \dots, LN - 1, \quad (3.47)$$

donde $\tau_0 = \epsilon_0$ es algún instante de tiempo donde se empezó el muestreo, T es el periodo de símbolo, L es el radio de sobre muestreo y N es el numero de periodos de símbolo donde el canal tiene suficiente energía para su consideración. Tomando lo anterior en consideración, la función de autocorrelación se convierte en la matriz de autocorrelación:

$$[\mathbf{R}]_{i,j} = R(\tau_i, \epsilon_j) = \sum_{k=1}^M \sigma_k^2 g(\tau_i - \gamma_k)g(\epsilon_j - \gamma_k), \quad (3.48)$$

definiendo el vector $[\mathbf{g}_k]_i = g(\tau_i - \gamma_k)$ la ecuación 3.48 puede ser expresada como:

$$\mathbf{R} = \sum_{k=1}^M \sigma_k^2 \mathbf{g}_k \mathbf{g}_k^T, \quad (3.49)$$

cuyos vectores propios son una base para el proceso estocástico discreto $h[k]$. Se puede apreciar que estas bases son dependientes de la función de autocorrelación del canal de comunicación, lo cual en la práctica es difícil de estimar para cada realización; así el objetivo de UB, es encontrar una base “universal” la cual sea capaz de expandir cualquier realización de canal con la única restricción de que la duración de la respuesta al impulso del canal este acotada a una duración máxima τ_{max} . En [60], se propone como ejemplo para generar esta base universal un PDP consistente en un escalón unitario de duración τ_{max} , que ha sido sobre-muestreado a un radio Q , así la función de autocorrelación queda dada por:

$$\mathbf{R}_u = \sum_{i=1}^M \mathbf{g}_k \mathbf{g}_k^T, \quad (3.50)$$

donde se ha fijado $\gamma_i = (i - 1)\tau_{max}/(Q - 1)$ para generar el vector \mathbf{g}_k . De esta manera la UB quedará definida por los vectores propios de la matriz (3.50).

3.3.3. Modelo matemático

Se toma como modelo matemático el expuesto en la sección 3.2.1, con la diferencia de que el receptor es capaz de realizar un sobre-muestreo, lo anterior se ejemplifica en la figura 3.2. Donde, $s[k']$ es la superposición de la secuencia de datos, la secuencia de entrenamiento y la secuencia de entrenamiento dependiente de los datos, denotadas por $b[k']$, $c[k']$ y $e[k']$ respectivamente; el índice k' se usa para enumerar las muestras de estas señales las cuales han sido tomadas a una tasa $1/T$. Estas señales presentan las

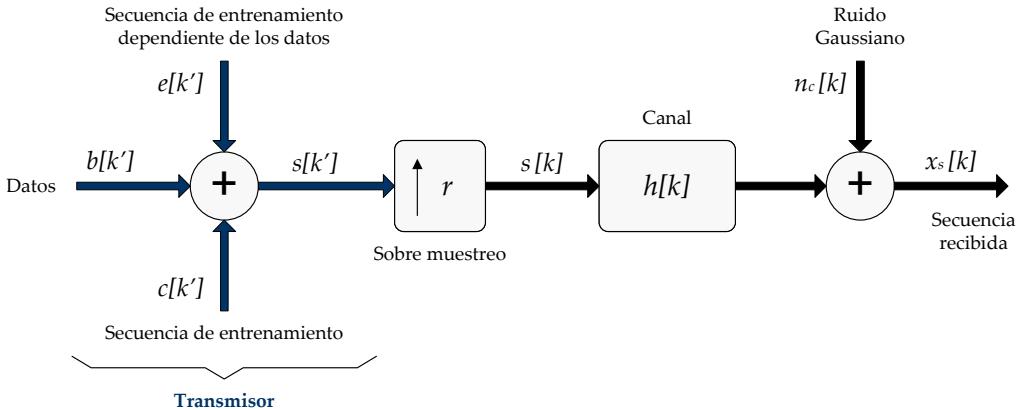


Figura 3.2: Sistema de comunicación invariante, discreto y pasa-baja basado en ST/DDST.

propiedades mencionadas en la sección 3.2.1. Ya que el receptor es capaz de sobremuestrear la señal recibida, tal como puede observarse en la figura 3.2, en el sistema existe un bloque para sobremuestrear la señal recibida a una tasa r/T , donde r es el factor de sobremuestreo. La señal $h[k]$ representa la CIR discreta del canal de comunicación invariante en el tiempo, k es el índice que enumera las muestras tomadas a una tasa r/T , y $s[k]$ es la señal que se propaga a través del canal de comunicación. En el lado del receptor la señal se contamina ahora con ruido Gaussiano coloreado $n_c[k]$ con media cero (debido al filtro acoplado) y función de autocorrelación: $R_{n_c} = E(n_c[k_1]n_c^*[k_2])$; produciendo la señal $x_s[k]$. Se supone una transmisión en bloques de N -símbolos junto con un prefijo cíclico de tamaño mayor o igual a M , donde M es la duración del canal en períodos de símbolo. El periodo de la secuencia de entrenamiento $c[k']$ es P . Dado que no se toma en cuenta en este modelo el desplazamiento de CD y se asume TSS, se puede disminuir el tamaño del prefijo a $P = M$. La señal recibida $x_s[k]$ después de remover el prefijo cíclico queda expresada por:

$$x_s[k] = \sum_{l=0}^{Mr-1} h[k-l]s[l] + n_c[k] = \sum_{l=0}^{Mr-1} s[k-l]h[l] + n_c[k], \quad (3.51)$$

donde $Mr = M \times r$ es la duración del canal a la tasa de muestreo r/T . Un estimador de la media cíclica de la señal recibida necesario para el algoritmo de DDST está dada por:

$$y[j] = \frac{1}{N_P} \sum_{i=0}^{N_P-1} x_s[iP_r+j] = \frac{1}{N_P} \sum_{l=0}^{Mr-1} h[l] \sum_{i=0}^{N_r-1} s[iP_r+j-l] + \frac{1}{N_P} \sum_{i=0}^{N_r-1} n_c[iP_r+j], \quad (3.52)$$

donde $j = 0, 1, \dots, P_r - 1$. Debido a la manera como se definió la secuencia $e[k]$, y a que $c[k] = c[iP_r + k]$ para $i, k \in \mathbb{Z}$, la ecuación (3.52) se transforma en:

$$y[j] = \sum_{l=0}^{Mr-1} h[l] \left(\frac{1}{N_P} \sum_{i=0}^{N_P-1} c[j-l] \right) = \sum_{l=0}^{Mr-1} h[l]c[j-l] + \frac{1}{N_P} \sum_{i=0}^{N_r-1} n_c[iP_r+j], \quad (3.53)$$

definiendo los vectores $[\mathbf{y}]_j = y[j]$, $[\mathbf{h}]_l = h[l]$, $[\mathbf{n}_c]_k = n_c[k]$ y la matriz $\mathbf{C} = circ(c[k])$ la ecuación (3.53) queda representada de forma matricial como:

$$\mathbf{y} = \mathbf{Ch} + \frac{1}{N_p} \mathbf{Jn}_c, \quad (3.54)$$

donde, $\mathbf{J} = \mathbf{1}_{1 \times N_p} \otimes \mathbf{I}_{P_r}$. Un estimado del canal de comunicación está dado por:

$$\hat{\mathbf{h}} = \mathbf{C}^{-1}\mathbf{y} = \mathbf{h} + \frac{1}{N_p} \mathbf{C}^{-1} \mathbf{Jn}_c. \quad (3.55)$$

Si se restringe el canal estimado a un subespacio de las posibles realizaciones del canal al proyectarlo con una base dada, esta nueva estimación del canal estará definida por:

$$\hat{\mathbf{h}}_b = \Delta \Delta^H \hat{\mathbf{h}}, \quad (3.56)$$

donde las columnas de Δ contienen los elementos ortonormales de una base que expande el subespacio supuesto. De la ecuación 3.56 es posible notar que $\Delta^H \hat{\mathbf{h}}$ constituye la proyección del canal con cada uno de los elementos de la base ortonormal, lo cual da como resultado los pesos que ponderan cada uno de los elementos de dicha base.

3.3.4. Análisis de desempeño

Sin perdida de generalidad se supone para este análisis que $\sum_{l=0}^{L_r-1} |h[l]|^2 = 1$, de esta manera haciendo uso de (3.55) y (3.56) el valor medio de $\hat{\mathbf{h}}_b$ está dado por:

$$E(\hat{\mathbf{h}}_b) = E(\Delta \Delta^H \hat{\mathbf{h}}) = \Delta \Delta^H E(\hat{\mathbf{h}}) = \Delta \Delta^H (E(\mathbf{h}) + \frac{1}{N_p} E(\mathbf{C}^{-1} \mathbf{Jn}_c)) = \Delta \Delta^H \mathbf{h}, \quad (3.57)$$

de esta manera se observa que el estimador no tiene sesgo si la proyección no introduce error (de modelado) alguno. Ahora definiendo el error del canal como $\epsilon = \|\hat{\mathbf{h}}_b - \mathbf{h}\|^2$, entonces la varianza de ϵ puede ser interpretado como la potencia del error de estimación, el cual estaría dada por:

$$var(\epsilon) = E(tr((\hat{\mathbf{h}}_b - \mathbf{h})(\hat{\mathbf{h}}_b - \mathbf{h})^H)). \quad (3.58)$$

Sustituyendo $\hat{\mathbf{h}}_b$ de (3.56) y haciendo la consideración: $\mathbf{h} \approx \Delta \Delta^H \mathbf{h}$ la ecuación anterior se transforma en:

$$var(\epsilon) = E[tr((\Delta \Delta^H \hat{\mathbf{h}} - \Delta \Delta^H \mathbf{h})(\Delta \Delta^H \hat{\mathbf{h}} - \Delta \Delta^H \mathbf{h})^H)], \quad (3.59)$$

después de algunas manipulaciones matemáticas, (3.59) puede ser expresada como:

$$var(\epsilon) = E[tr(\Delta \Delta^H (\hat{\mathbf{h}} - \mathbf{h})(\hat{\mathbf{h}} - \mathbf{h})^H \Delta \Delta^H)]. \quad (3.60)$$

Usando (3.55) se tiene que: $\hat{\mathbf{h}} - \mathbf{h} = \frac{1}{N_p} \dot{\mathbf{C}}^{-1} \mathbf{Jn}_c$; aplicando esto en (3.60) se obtiene:

$$var(\epsilon) = E[\frac{1}{N_p^2} tr(\Delta \Delta^H \dot{\mathbf{C}}^{-1} \mathbf{Jn}_c \mathbf{n}_c^H \mathbf{J}^H \dot{\mathbf{C}}^{-H} \Delta \Delta^H)], \quad (3.61)$$

la cual al utilizar las propiedades de la traza $tr(\mathbf{ABC}) = tr(\mathbf{CAB}) = tr(\mathbf{BCA})$ es equivalente a:

$$var(\epsilon) = E\left[\frac{1}{N_P^2} tr(\Delta \Delta^H \Delta \Delta^H \dot{\mathbf{C}}^{-1} \mathbf{J} \mathbf{n}_c \mathbf{n}_c^H \mathbf{J}^H \dot{\mathbf{C}}^{-H})\right]. \quad (3.62)$$

Debido a que Δ es una matriz ortonormal $\Delta^H \Delta = \mathbf{I}_P$, donde P es el numero de elementos utilizados en la base, la ecuación (3.62) puede ser simplificada a:

$$var(\epsilon) = E\left[\frac{1}{N_P^2} tr(\Delta \Delta^H \dot{\mathbf{C}}^{-1} \mathbf{J} \mathbf{n}_c \mathbf{n}_c^H \mathbf{J}^H \dot{\mathbf{C}}^{-H})\right]. \quad (3.63)$$

Aplicando el operador de esperanza en (3.63), se tiene que la varianza del error es

$$var(\epsilon) = \frac{1}{N_P^2} tr(\Delta \Delta^H \dot{\mathbf{C}}^{-1} \mathbf{J} \mathbf{R}_{n_c} \mathbf{J}^H \dot{\mathbf{C}}^{-H}). \quad (3.64)$$

Se observa en (3.64) que la varianza del error depende de la matriz de autocorrelación del ruido, la cual a su vez depende del filtro acoplado en el receptor y la base que se esté utilizando para proyectar el canal. En el caso particular de que el factor de sobre-muestreo sea $r = 1$, se tiene que \mathbf{R}_{n_c} se vuelve una matriz identidad escalada por la potencia del ruido, así (3.64) se reduce a:

$$var(\epsilon) = \frac{\sigma_n^2}{N_P^2} tr(\Delta \Delta^H \dot{\mathbf{C}}^{-1} \mathbf{J} \mathbf{I}_N \mathbf{J}^H \dot{\mathbf{C}}^{-H}), \quad (3.65)$$

dado que $\mathbf{J} \mathbf{J}^H = N_P \mathbf{I}_{Pr}$, la ultima ecuación se reescribe como:

$$var(\epsilon) = \frac{\sigma_n^2}{N_P} tr(\Delta \Delta^H \dot{\mathbf{C}}^{-1} \mathbf{I}_P \dot{\mathbf{C}}^{-H}). \quad (3.66)$$

Ahora si se considera que \mathbf{c} es una secuencia OCI como en [5], se puede hacer que $\mathbf{C}^{-1} \mathbf{C}^{-H} = \mathbf{C}^H \mathbf{C} = \mathbf{I}_{Pr}$ y (3.66) se convierta en:

$$var(\epsilon) = \frac{\sigma_n^2}{N \sigma_c^2} tr(\Delta \Delta^H) = \frac{\rho \sigma_n^2}{N \sigma_c^2}. \quad (3.67)$$

3.4. Algoritmos para la transformada discreta de Fourier

La transformada discreta de Fourier (DFT) es una de las operaciones más empleadas en el procesamiento digital de señales, ya que facilita la transformación eficiente entre los dominios temporal y frecuencial de una señal discreta.

La DFT es una secuencia discreta que se obtiene mediante el muestreo de un periodo de la transformada de Fourier; tal muestreo es realizado en N puntos sobre un periodo de $0 \leq \omega \leq 2\pi$. En particular, la DFT proporciona la descomposición de una señal dentro de N componentes periódicas.

La DFT de N -puntos de la señal discreta en el dominio del tiempo $x(n)$ se define como:

$$X[k] = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad k = 0, \dots, N-1, \quad (3.68)$$

$$W_N^k = e^{-j\frac{2\pi k}{N}} = \cos\left(\frac{2\pi k}{N}\right) - j \sin\left(\frac{2\pi k}{N}\right), \quad (3.69)$$

donde W_N denota los factores rotatorios, n y k representan los índices en el dominio del tiempo y en el de la frecuencia normalizada, respectivamente. Similarmente, la transformada discreta inversa de Fourier (IDFT) es dada por:

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} X(k) W_N^{-nk}. \quad (3.70)$$

No obstante, la evaluación directa de (3.68) requiere de N multiplicaciones complejas y $N - 1$ sumas complejas para cada valor de la DFT. En consecuencia, para el cálculo de una DFT completa serán necesarias un total de N^2 multiplicaciones complejas y $N(N - 1)$ sumas complejas, esto implica un tiempo de procesamiento muy grande incluso para valores pequeños de N . Por tal motivo, resulta preferible usar la FFT, el cual es un algoritmo para calcular de forma eficiente la DFT a través de la reducción de su complejidad computacional. Para esto, los algoritmos para la FFT se apoyan en dos principios; el primero consiste en descomponer una DFT de longitud N en DFT sucesivas de subsecuencias más cortas y fáciles de calcular. El segundo consiste en explotar las propiedades de simetría ($W_N^{k+N/2} = -W_N^k$) y periodicidad ($W_N^{k+N} = W_N^k$) de los factores rotatorios. También, dado que la DFT y la IDFT básicamente involucran los mismos tipos de cálculos, los algoritmos para la FFT aplican de igual forma para el cálculo de la IDFT, lo que se conoce como transformada inversa rápida de Fourier (IFFT) [61].

3.4.1. Clasificación

Los algoritmos de la FFT pueden a grandes rasgos clasificarse de dos formas: por el tipo de *radix* (base) o por el tipo de decimado. En lo que respecta al tipo de *radix*, éste a su vez puede subdividirse en: fijo, dividido o mixto. Desde la perspectiva del tipo de decimado existen dos variantes: decimado en tiempo (DIT) y decimado en frecuencia (DIF). Ambos se basan en la descomposición recursiva de la transformada original y no existe diferencia en la complejidad computacional entre uno y el otro [62].

3.4.2. Algoritmo FFT radix-2 decimado en tiempo (DIT-2)

Este algoritmo para la FFT divide una secuencia de longitud $N = 2^v$, donde $v \in \mathbb{Z}$, en dos secuencias de $\frac{N}{2}$ datos $f_1(n)$ y $f_2(n)$, que contienen las muestras con índice par e impar de $x(n)$, respectivamente [63]. Entonces, la DFT de N -puntos puede calcularse como,

$$X(k) = F_1(k) + W_N^k F_2(k), \quad (3.71)$$

$$X(k + \frac{N}{2}) = F_1(k) - W_N^k F_2(k), \quad (3.72)$$

donde $k = 0, 1, \dots, \frac{N}{2} - 1$, $F_1(k)$ y $F_2(k)$ son las respectivas DFTs de $\frac{N}{2}$ -puntos de $f_1(n)$ y $f_2(n)$.

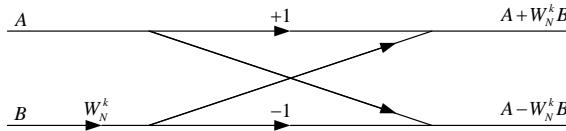


Figura 3.3: Estructura del *butterfly* DIT-2.

Las subsecuencias $F_1(k)$ y $F_2(k)$ son resueltas recursivamente aplicando (3.71) y (3.72). De esta forma, es posible identificar la operación elemental conocida como *butterfly* DIT-2 y su representación es similar a la que se muestra en la figura 3.3.

En general, cada *butterfly* conlleva una multiplicación y dos sumas complejas. Por lo que, para $N = 2^v$ existen $\frac{N}{2}$ *butterflies* a lo largo de $\log_2(N)$ etapas que dura el algoritmo, por lo que el total de multiplicaciones complejas es $\frac{N}{2} \log_2(N)$ y el de sumas complejas $N \log_2(N)$.

Para la ejecución correcta de la FFT DIT-2, el orden en la secuencia de entrada debe ser modificado para que las muestras tengan un orden binario invertido (*bit-reversal permutation*) antes que los cálculos tengan lugar. De esta forma se asegura que los datos de salida sean entregados en el orden correcto por el algoritmo. Esto se puede visualizar claramente en el flujo de datos de ejemplo que se muestra en la figura 3.4.

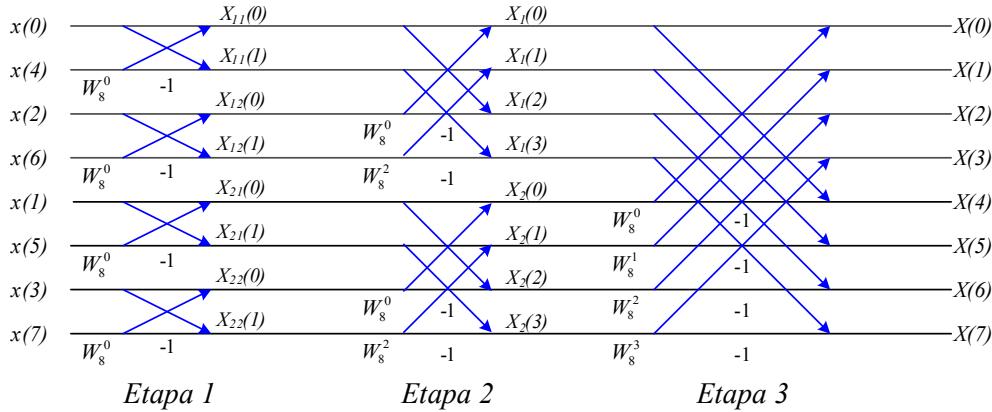


Figura 3.4: Flujo de datos de una FFT decimada en tiempo de 8-puntos.

3.4.3. Algoritmo FFT radix-2 decimado en frecuencia (DIF-2)

En este algoritmo —presentado por primera vez en [64]— la idea principal radica en dividir la secuencia de salida en muestras pares $X(2k)$ e impares $X(2k + 1)$, i.e.

$$X(2k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + x(n + \frac{N}{2}) \right] W_{N/2}^{kn} \quad y \quad (3.73)$$

$$X(2k+1) = \sum_{n=0}^{(N/2)-1} \left[\left(x(n) - x\left(n + \frac{N}{2}\right) \right) W_N^n \right] W_{N/2}^{kn}, \quad (3.74)$$

si se definen las subsecuencias de $N/2$ puntos $a_1(n)$ y $a_2(n)$ como:

$$a_1(n) = x(n) + x\left(n + \frac{N}{2}\right), \quad a_2(n) = \left[\left(x(n) - x\left(n + \frac{N}{2}\right) \right) W_N^n \right] W_{N/2}^{kn}, \quad n = 0, 1, \dots, \frac{N}{2} - 1.$$

entonces (3.73) y (3.74) pueden representarse por

$$X(2k) = \sum_{n=0}^{(N/2)-1} a_1(n) W_{N/2}^{kn} \quad y \quad (3.75)$$

$$X(2k+1) = \sum_{n=0}^{(N/2)-1} a_2(n) W_{N/2}^{kn}, \quad k = 0, 1, \dots, \frac{N}{2} - 1. \quad (3.76)$$

Este procedimiento puede ser repetido a través del decimado de las DFTs de $\frac{N}{2}$ puntos $X(2k)$ y $X(2k+1)$ en nuevas subsecuencias. Al igual que la FFT DIT-2, el proceso entero toma $\log_2(N)$ etapas de decimado, donde cada una de ellas involucra $\frac{N}{2}$ operaciones *butterflies* DIF-2 como la mostrada en la figura 3.5.

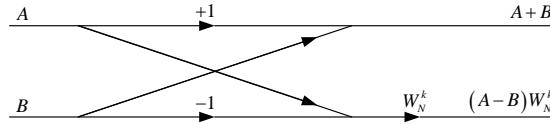


Figura 3.5: Estructura del *butterfly* DIF-2.

Analizando el flujograma para una FFT DIF-2 de 8-puntos mostrado en la figura 3.6, se puede notar que —a diferencia del algoritmo DIT-2— la secuencia de entrada debe ser alimentada en orden natural y el reordenamiento ahora se debe realizar en la secuencia de datos de salida.

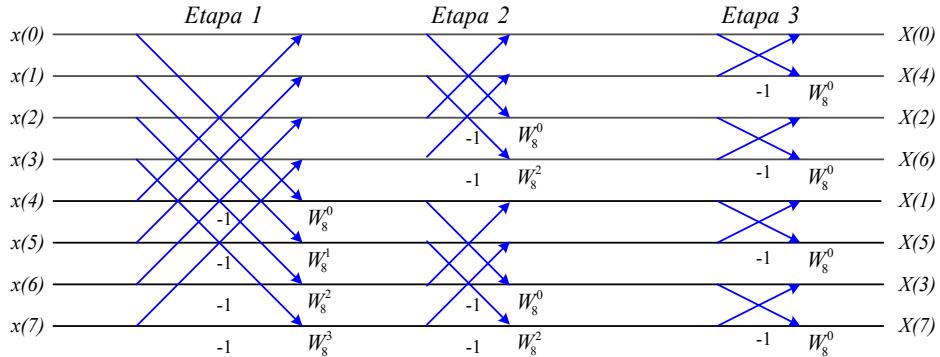


Figura 3.6: Flujo de datos de una FFT decimada en frecuencia de 8-puntos.

3.4.4. Algoritmo FFT radix-4 decimado en tiempo (DIT-4)

Cuando el número de muestras de entrada en la DFT es una potencia de 4, i.e. $N = 4^v$, donde $v \in \mathbb{Z}$, siempre puede aplicarse un algoritmo FFT *radix-2*. Sin embargo, para este caso es más eficiente desde el punto de vista computacional, recurrir a un algoritmo *radix-4* para efectuar la transformación.

Para su análisis, por principios de cuentas se divide la secuencia de entrada de N -puntos en cuatro subsecuencias, $x(4n)$, $x(4n + 1)$, $x(4n + 2)$ y $x(4n + 3)$, donde $n = 0, 1, \dots, \frac{N}{4} - 1$. Luego, se obtiene la DFT de cada una de tales subsecuencias, la cual viene dada por

$$F(l, q) = \sum_{m=0}^{(N/4)-1} x(l, m) W_{N/4}^{mq} \quad (3.77)$$

donde

$$l = 0, 1, 2, 3, \quad q = 0, 1, \dots, \frac{N}{4} - 1,$$

y

$$x(l, m) = x(4m + l), \quad X(p, q) = x\left(\frac{N}{4}p + q\right).$$

Ahora, las cuatro DFTs de $\frac{N}{4}$ -puntos deben combinarse para producir la DFT completa de N -puntos. Esto se logra con la siguiente expresión:

$$X(p, q) = \sum_{l=0}^3 \left[W_N^{lq} F(l, q) \right] W_{N/4}^{lp} \quad p = 0, 1, 2, 3. \quad (3.78)$$

Representando (3.78) de forma matricial, se obtiene

$$\begin{bmatrix} X(0, q) \\ X(1, q) \\ X(2, q) \\ X(3, q) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} W_N^0 F(0, q) \\ W_N^q F(1, q) \\ W_N^{2q} F(2, q) \\ W_N^{3q} F(3, q) \end{bmatrix}. \quad (3.79)$$

La ecuación (3.79) define la operación atómica fundamental del algoritmo FFT DIT-4. Esta es conocida como *dragonfly* y su estructura puede visualizarse en la figura 3.7

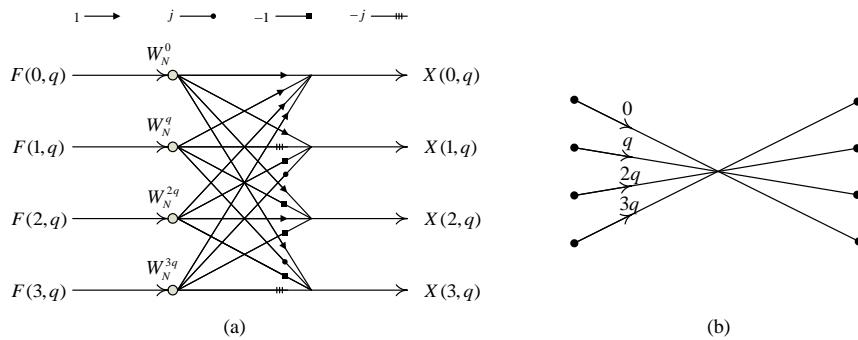


Figura 3.7: Estructura del *dragonfly* DIF-4: a) Completa, b) Representación simplificada.

Se puede apreciar que en cada *dragonfly* se encuentran involucradas tres multiplicaciones complejas y doce sumas complejas. Además, a lo largo del algoritmo se ejecutan $\log_4(N)$ etapas de decimado y en cada una de ellas se procesan $\frac{N}{4}$ *dragonflies*.

3.5. Algoritmos para la convolución y correlación

La convolución y la correlación son dos operaciones de uso intensivo para el procesamiento digital de señales en telecomunicaciones. Las propiedades y aplicaciones de ambas operaciones son distintas, a pesar de la gran similitud existente en las ecuaciones que las representan (la diferencia radica en el signo de uno de los operandos). No obstante, esto implica que en esencia, la convolución y correlación pueden calcularse mediante el mismo algoritmo, introduciendo mínimas modificaciones.

3.5.1. Convolución discreta

De forma general, la convolución es la operación matemática que de forma lineal, define como dos funciones, continuas o discretas se combinan para generar una nueva función de salida. Sin embargo, ya en la práctica lo que en realidad se procesa y combina son secuencias discretas de longitud finita para formar una tercera. En estos términos, la convolución $a(n)$ de dos secuencias $r(n)$ y $g(n)$ está dada por

$$a(n) = \sum_{m=0}^{N-1} r(m)g(m-n) = r(n) \otimes g(n). \quad (3.80)$$

Desafortunadamente, calcular directamente (3.80) involucra un gran número de operaciones, el cual es proporcional a N^2 . Pero, si se considera la propiedad que establece que la convolución en el dominio del tiempo es equivalente a la multiplicación en el dominio de la frecuencia, (3.80) se transforma en:

$$A(k) = R(k) G(z), \quad (3.81)$$

donde $A(k)$, $R(k)$ y $G(k)$, son las DFTs de $a(n)$, $r(n)$ y $g(n)$. Debido a la periodicidad introducida por la DFT, la convolución descrita en 3.81 es de tipo circular y no regular.

También es posible calcular la convolución lineal o regular usando la DFT. Si las secuencias a procesar tienen longitudes N_r y N_g , la longitud de la convolución lineal tendrá $N_r + N_g + 1$ muestras. Por lo tanto, antes de calcular la DFT, se debe hacer *padding* con suficientes ceros cada secuencia para que cada una tenga una longitud de $N_r + N_g + 1$.

3.5.2. Correlación discreta

La correlación es una operación matemática que permite cuantificar el grado de similitud entre dos señales, aunque aparentemente no haya evidencias de coincidencia temporal entre ellas. Para calcularla, se usa un proceso parecido a la convolución; de tal

forma que, la correlación $z(n)$ existente entre las secuencias de $r(n)$ y $g(n)$ se representa con

$$z(n) = \sum_{m=0}^{N-1} r(m)g(m+n) = r(n) \otimes \otimes g(n). \quad (3.82)$$

De igual manera que en la convolución circular, el cálculo de la correlación es más sencillo en el dominio de la frecuencia, salvo por un paso previo de conjugación en una de las secuencias. La correlación circular de las secuencias $r(n)$ y $g(n)$ cuando cuenta con longitudes iguales N , se denota por:

$$Z(k) = R(k) G(z)^*. \quad (3.83)$$

Asimismo, la correlación lineal puede ser encontrada con un procedimiento similar al expuesto para la convolución regular.

3.6. Conclusiones del capítulo

De lo expuesto en este capítulos se concluye lo siguiente:

- El análisis matemático de los algoritmos involucrados en un sistema de comunicación con IT, permite desarrollar modelos de simulación que los integren de forma correcta dentro del sistema con la finalidad de mejorar su desempeño.
- Debido a que no siempre se pueden tener las estadísticas de la señal a expandir, pero si se puede contar con alguna otra información tal como: la duración máxima, ciertas características de la respuesta al impulso y ancho de banda máximo; se pueden proponer bases que son construidas de manera sencilla con la ayuda de esta información y que sigan expandiendo estas señales.
- Las bases universales pueden construirse en función del conocimiento de la duración máxima y la respuesta al impulso del filtro que limita su ancho de banda.
- El uso de expansión en bases mejora los resultados en el proceso de estimación de un canal invariante en el tiempo.
- Los algoritmos de la FFT permiten reducir los tiempos de procesamiento para las transformaciones entre los dominios temporal y frecuencial, así como para realizar análisis espectrales.
- El cálculo de la convolución y correlación discreta es más eficiente si se realiza en el dominio de la frecuencia.

Capítulo 4

Desarrollo de un transmisor ST/DDST configurable

4.1. Introducción

En este capítulo se describirán los pormenores concernientes al proceso de trasladar cada uno de los algoritmos involucrados en un sistema de comunicaciones con IT a su correspondientes arquitecturas de *hardware*. Dicho procedimiento conocido como **mapeo de algoritmo a arquitectura** (*algorithm-to-architecture mapping*), permite que el algoritmo en cuestión pueda ser implementado en forma eficiente ya sea en términos de velocidad o en área consumida. Asimismo, facilita que los diseños puedan ser replicados por cualquiera que requiera de su uso.

4.2. Procedimiento de mapeo de algoritmo a arquitectura

En sentido amplio, este proceso está conformado por los siguientes fases:

1. Generación del modelo de oro (*Golden model*).
2. Adecuación del algoritmo (*Algorithm transformation*).
3. Análisis de punto fijo (*Fixed-point analysis*).
4. Diseño de la arquitectura de hardware (*Hardware architecture design*).
5. Implementación y verificación de la arquitectura (*Architecture implementation and verification*).

Vale la pena resaltar, que este proceso puede tener una naturaleza iterativa en cualquiera de sus fases, ya que ha menudo se requieren hacer ajustes para obtener los resultados deseados.

4.2.1. Generación del modelo de oro

El objetivo que se persigue aquí es el de generar una referencia para el algoritmo que pueda emplearse en pasos posteriores para evaluar otras figuras de mérito. Para esto, se debe programar el algoritmo objeto de estudio, empleando preferentemente un lenguaje de alto nivel, en aritmética de punto flotante y con sus ecuaciones originales.

4.2.2. Adecuación del algoritmo

La finalidad de este punto es modificar las ecuaciones originales o secuencia de pasos del algoritmo para reducir su complejidad sin alterar el resultado final. No existe una regla de dedo para aplicarlo, únicamente el entendimiento profundo y un análisis exhaustivo del algoritmo ayudará a identificar si los siguientes artificios (por citar algunos) pueden ayudar a replantearlo.

- Factorizaciones o aproximaciones matemáticas.
- Simplificaciones o propiedades matemáticas.
- Transformaciones o cambios de dominios.
- Sistolización de operaciones de cálculo intensivo.
- Sustitución de divisiones y multiplicaciones con operaciones conjuntas de desplazamientos y sumas/restas.

4.2.3. Análisis de punto fijo

La elección natural de emplear aritmética de punto flotante en las arquitecturas de PDS regularmente es desechada. Esto se debe a que es posible alcanzar desempeños similares si se sustituyen los cálculos con su representación de punto fijo. También, éstos últimos ocupan menos tiempo de procesamiento y son menos complejos de implementar. En consecuencia, su consumo de área y potencia son menores.

Bajo la luz de lo expuesto, en este trabajo se ha optado por incluir aritmética de punto fijo en las arquitecturas de procesamiento digital de señales a diseñar, lo cual requiere que un análisis del algoritmo sea llevado a cabo en esos términos para encontrar la longitud y el formato de palabra correcto. Para cuantificar los efectos de la aritmética de punto fijo en un algoritmo, se usa la figura de mérito SQNR, que mide la calidad de una señal analógica después de ser sometida a un proceso de cuantización.

El análisis de punto fijo se divide en los pasos que a continuación se enumeran:

- (a) Medición del rango dinámico: Se define como rango dinámico a la diferencia entre los valores máximos y mínimos que toma la variable. De tal forma, que resulta imprescindible dimensionar el rango dinámico de todas las variables en punto flotante que son manipuladas por el algoritmo, para después calcular la cantidad de bits que

se le deben asignar tanto a la parte entera como a la fraccionaria para representarlas en punto fijo.

- (b) Selección del formato de palabra: Existen diversas formas para obtener el formato de palabra, ha continuación se detalla el que se aplica en esta tesis [65]. La cantidad de bits Q_I para la representación entera de una palabra en punto fijo se relacionada directamente con el rango dinámico de la variable en cuestión, por medio de la expresión

$$Q_I = \lfloor \log_2(\max(\text{abs}(\alpha_{\max}), \text{abs}(\alpha_{\min}))) \rfloor + 2, \quad (4.1)$$

donde α_{\max} y α_{\min} son los valores máximo y mínimo del rango dinámico de la variable en punto flotante y $\lfloor \cdot \rfloor$ es el operador *floor*. La parte fraccional define la resolución ε de una variable en punto fijo. La resolución es el cambio más pequeño que se puede tener debido a los niveles de cuantización y queda expresada como:

$$\varepsilon = \frac{1}{2^{Q_F}}. \quad (4.2)$$

Por lo que, al despejar Q_F que representa el número de bits fraccionales para un resolución dada, se tiene que:

$$Q_F = \left\lceil \log_2 \left(\frac{1}{\varepsilon} \right) \right\rceil, \quad (4.3)$$

donde $\lceil \cdot \rceil$ denota el operador *ceil*.

Cabe señalar que la resolución de la parte fraccional depende de las necesidades del sistema. En general, se debe hacer una elección cuidadosa debido a que una mayor resolución desembocará en un aumento en la anchura de la palabra (Q_W).

- (c) Cuantificación de la relación señal a ruido de cuantización (SQNR): El proceso de cuantizar un valor de punto flotante genera un error que puede ser visto como ruido que contamina al valor original, por ende el SQNR es la figura de mérito preferida para medir la calidad del proceso de cuantificación de una variable de punto flotante. Debido que la mayoría de los algoritmos operan con datos expresados como matrices y vectores, la SQNR vendrá expresada como un promedio dado por:

$$S Q N R = 10 \log_{10} \left[\frac{\sum_{n=0}^{N-1} a(n)^2}{\sum_{n=0}^{N-1} (b(n) - a(n))^2} \right], \quad (4.4)$$

donde $a(n)$ es una señal original en punto flotante, $b(n)$ es su contraparte en punto fijo y N es la cantidad de elementos que las conforman. En particular, se debe monitorear el SQNR en todas las variables del algoritmo y ajustar la longitud de la palabra en aquellas donde el SQNR sea muy pobre y sacrificar bits en donde sea muy elevado. La finalidad es alcanzar un SQNR objetivo en los resultados de salida con un equilibrio en las longitudes de palabras de las variables.

- (d) Efectos del redondeo: El inconveniente de usar aritmética de punto fijo es que no todos los valores pueden ser representados de manera exacta. Cuando esto sucede, el método de redondeo se utiliza para convertir el valor a un número representable con la consabida pérdida de precisión. La elección del tipo de redondeo que se pretenda usar en el algoritmo impactará directamente en su desempeño en términos de SQNR, área y velocidad de procesamiento. Así, un redondeo sencillo implicará un diseño más rápido, con menor consumo de área y con un SQNR menor que si se utilizará un redondeo más complejo. Los tipos de redondeo más empleados son el truncamiento, redondeo hacia cero, redondeo hacia más/menos infinito y el redondeo al más cercano.

4.2.4. Diseño de la arquitectura de hardware

Toda vez que se han dimensionado las longitudes y los formato de palabra correspondientes a las variables del algoritmo, lo siguiente es realizar un diagrama a bloques donde se visualice el recorrido que deben seguir los datos desde la entrada hasta la salida de algoritmo y que además capture su esencia en un nivel alto de abstracción. Cada bloque deberá representar un proceso funcional que derive en una transformación de los datos. Posteriormente, se tiene que ir incrementando el nivel de detalle inicial subdividiendo cada bloque en módulos, estableciendo una jerarquía. Cada módulo a su vez se tendrá que subdividir cuantas veces sea necesario hasta que sea sencillo de diseñar o su funcionalidad se relacione con algún componente u operación primitiva. De esta forma se tendrá una primera versión de lo que se conoce como la ruta de datos (*datapath*) de la arquitectura, la cual se tendrá que complementar con su correspondiente unidad de control.

Dependiendo del criterio de optimización: velocidad o consumo de área; que se deseé aplicar a la arquitectura inicial, ésta se podrá refinar a través de las técnicas que a continuación se mencionan.

- (I) Paralelismo: Permite la ejecución simultánea de varias operaciones, siempre y cuando no existan una dependencia de datos entre ellas. Esto permite incrementar significativamente la velocidad de operación de la arquitectura pero el consumo de área también se verá afectado en la misma proporción.
- (II) Encauzamiento (*pipelining*): Cuando en un proceso de la arquitectura se ejecuta una secuencia de operaciones para obtener un resultado, es posible introducir una etapa de *pipeline* que permita traslapar la ejecución de tales operaciones de forma que, ya no será necesario esperar a que se termine de procesar dicha secuencia para poder introducir un nuevo dato al proceso. Vale señalar que la etapa de *pipeline* no mejorará el tiempo para procesar cada dato, pero sí incrementará la cantidad de operaciones que se ejecutan, con lo que el rendimiento de la arquitectura será mayor con un costo mínimo en el consumo de área.

- (iii) Reconfigurabilidad: No existe una definición concreta para éste término, pero está estrechamente relacionado con la capacidad que se le puede otorgar a una arquitectura para modificar en tiempo de ejecución su ruta de datos. Esto con el objetivo de cambiar su funcionalidad y usando los mismos bloques o módulos de *hardware*. Esto disminuirá el consumo del área en la arquitectura pero las funcionalidades con las que cuente ahora serán excluyentes.
- (iv) Reusabilidad: Cuando alguno de los bloques o módulos que cumplan una misma función aparece más de una vez dentro de una arquitectura, se recomienda diseñar solo uno y usarlo las veces que sea necesario, en lugar de diseñar varios de ellos idénticos. Esto disminuirá drásticamente el consumo de área en la arquitectura pero depreciará la velocidad de operación.
- (v) Portabilidad: A menos que se indique lo contrario, la descripción de la arquitectura no debe asociarse con opciones o componentes de alguna tecnología específica de un fabricante. Esto garantizará su independencia y asegurará el que pueda ser implementada en varias plataformas de diseño (*EDA tools*) de cualquier fabricante sin realizar cambios en la descripción.

4.2.5. Implementación y verificación de la arquitectura

Toda vez que se tiene definida la arquitectura de *hardware* definitiva, lo que procede es describirla e implementarla a nivel de lógica de transferencia de registros (RTL) usando un lenguaje de descripción de *hardware* (HDL). El entorno de desarrollo —que el fabricante de la tecnología pone a disposición del arquitecto — juega un papel fundamental al facilitar estos procesos. Después, la arquitectura ya sintetizada tiene que pasar por un proceso de verificación antes de que sea declarada lista para usarse. Esto involucra estudiar su comportamiento en condiciones controladas para:

- * Verificar y asegurar que todas las funciones para las que fue diseñada sean llevadas a cabo. Esto se conoce como verificación estática.
- * Verificar y asegurar que todas las secuencias involucradas en cada una de las funciones se ejecuten correctamente. Esto se conoce como verificación por comportamiento dinámico, e implica la generación de vectores de entradas específicos variantes con el tiempo, que se aplican a la arquitectura para monitorear las salidas generadas partir de ellos.
- * Comprobar el comportamiento temporal: De manera adicional, es posible elaborar pruebas para observar las variaciones de las señales en puntos seleccionados, medir retardos entre eventos específicos, ancho de pulsos, etc.

La verificación se fundamenta casi siempre en bancos de pruebas (*test-benches*), que básicamente son un entorno de simulación alrededor de la arquitectura. Su labor consiste

en injectar un conjunto de estímulos específicos en las entradas y verificar que el resultado obtenido en las salidas sea el esperado. Por otro lado, es importante resaltar que a pesar de lo exhaustiva que pueda llegar a ser una verificación, nunca se logra al 100 % cuando se trata de arquitecturas complejas.

4.3. Arquitectura digital de un transmisor ST/DDST configurable

Antes de iniciar esta sección, es prudente mencionar que la forma que se ha elegido para exponer los pormenores relativos al diseño de cada una de las arquitecturas objeto de estudio de esta tesis, comenzará precisamente con la presentación del diagrama de la arquitectura final obtenida. Esto se hace con la finalidad de clarificar detalles específicos en cada uno de los bloques y de ninguna manera contraviene a lo mencionado en la sección 4.2, que fue el procedimiento que se siguió. Otro aspecto importante a resaltar corresponde al hecho de restringir de que P y N sean una potencia de dos, esto con el fin de que cualquier operación de división pueda ser replanteada con desplazamientos.

La arquitectura simplificada para el transmisor con IT propuesto en la sección 2.3.1 se muestra en la figura 4.1. Su característica de configurabilidad permite modificar su funcionalidad para poder transmitir bloques de datos, ya sea con ST o DDST junto con su prefijo cíclico, empleando constelaciones QAM de 4, 16 ó 64 puntos. Asimismo en la figura 4.1 se pueden identificar los seis módulos de *hardware* que conforman la arquitectura, los cuales son: el adecuador de símbolos, un mapeador, el transformador de la secuencia de datos, un generador de direcciones y una unidad de control. El proceso completo para generar un bloque de datos con IT a través de tales módulos es descrito a continuación.

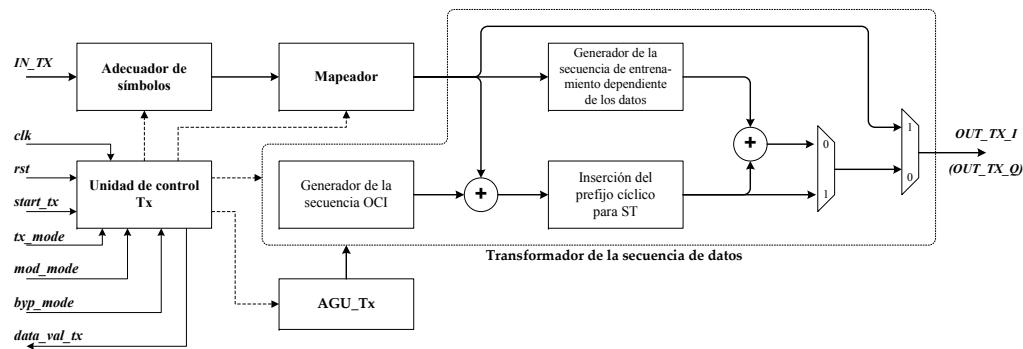


Figura 4.1: Arquitectura simplificada de *hardware* del transmisor ST/DDST configurable.

En el momento en que se activa la señal *start_tx*, el transmisor verifica el valor de sus señales de entrada y establece su modo de operación acorde a lo que se presenta en la tabla 4.1. Hecho esto, los N símbolos de un bloque son leídos en forma serial a través

Entradas de control				Modo de transmisión	Constelación
byp_mode	tx_mode	mod_mode			
0	0	0	0	ST	--
0	0	0	1	ST	QPSK
0	0	1	0	ST	QAM-16
0	0	1	1	ST	QAM-64
0	1	0	0	DDST	--
0	1	0	1	DDST	QPSK
0	1	1	0	DDST	QAM-16
0	1	1	1	DDST	QAM-64
1	x	0	0	Sin IT	--
1	x	0	1	Sin IT	QPSK
1	x	1	0	Sin IT	QAM-16
1	x	1	1	Sin IT	QAM-64

Tabla 4.1: Tabla de verdad del transmisor ST/DDST.

del bus de entrada *IN_TX* y transferidos al adecuador de símbolos que los convierte en las salidas de direccionamiento *addr_I* y *addr_Q*. A partir del valor que se presenta en estas salidas, el mapeador genera un símbolo complejo de la secuencia $b(k)$, lo normaliza dependiendo del modo de operación y el orden de la constelación, y lo envía al siguiente módulo en dos buses independientes, uno para la componente real y otro para la imaginaria. En consecuencia, todas las operaciones ejecutadas en la arquitectura a partir del mapeador serán sobre señales con valores complejos. En el modo de operación ST, el transformador de la secuencia de datos tomará cada uno de los símbolos de $b(k)$ y les sumará el símbolo de entrenamiento que le corresponde de la secuencia $c(k)$. Sin embargo, si una transmisión con DDST toma lugar, se deberá sumar también el símbolo en turno de la secuencia $e(k)$. Finalmente, una vez procesados los N símbolos, el prefijo cíclico de longitud P se incorporará al bloque de datos y los $N + P$ símbolos serán transmitidos uno a uno. La bandera *data_val_tx* se activará para indicar que el primer dato válido está presente en los buses de salida *OUT_TX_I* y *OUT_TX_Q* y el transmisor queda habilitado para el procesamiento de un nuevo bloques de datos.

Ahora, los detalles específicos para el diseño de la arquitectura individual de cada uno de los módulos del transmisor se describen en las siguientes subsecciones

4.3.1. El adecuador de símbolos

El diseño de este módulo está en mayor medida supeditado a la forma operacional dispuesta para el mapeador. Por tal motivo, en este apartado es inevitable adelantar aspectos referentes a este último para comprender cómo se diseñó. Por principios de cuentas, ya que el transmisor debe operar bajo un esquema de modulación QAM con tres diferentes ordenes de constelación, la solución obvia de usar una tabla de translación o LUT (*look-up table*) para cada una de ellas, se descartó tempranamente por ser inefi-

ciente en términos de consumo de área. Por tal razón, una de las adecuaciones que se plantea en el mapeador se fundamenta en el hecho de que las constelaciones 16-QAM y 4-QAM (QPSK) están inmersas dentro de la de 64-QAM, tal como lo ilustra la figura 4.2c. Esto permite que únicamente sea necesaria dicha constelación junto con una correcta adecuación de los símbolos para la operación del mapeador. Dicha adecuación es

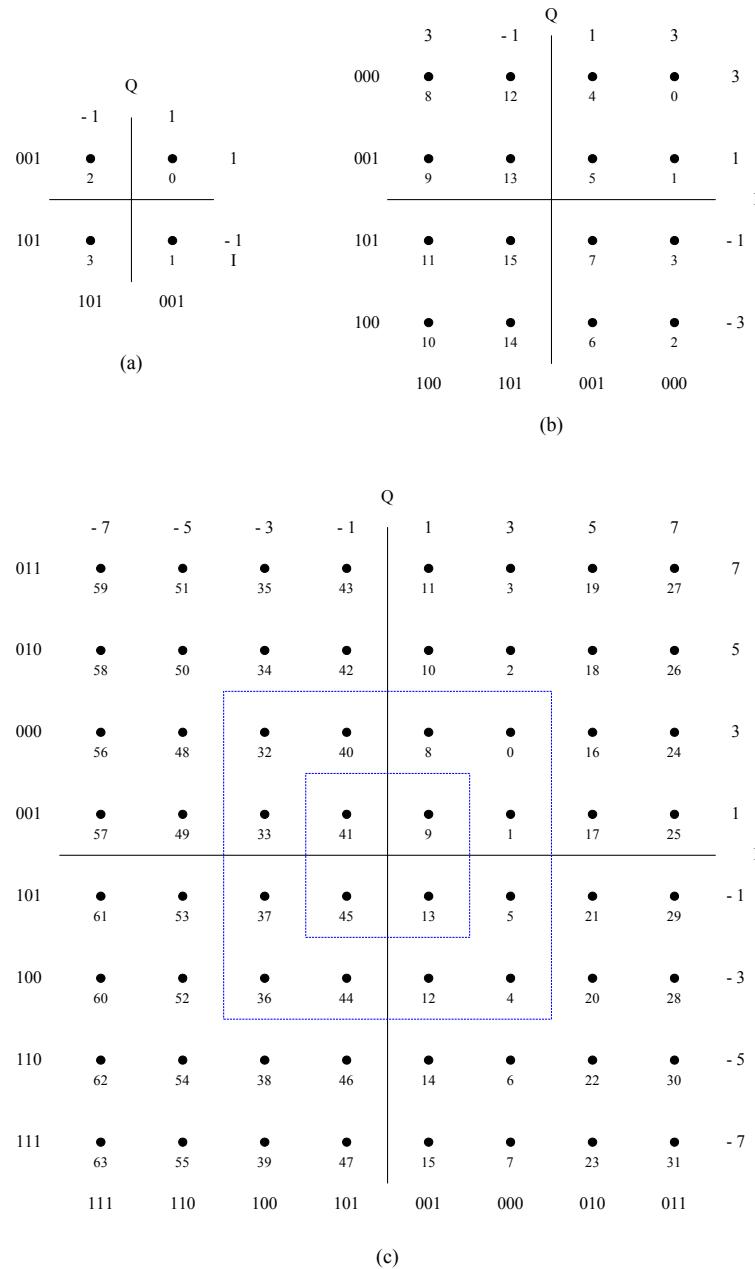
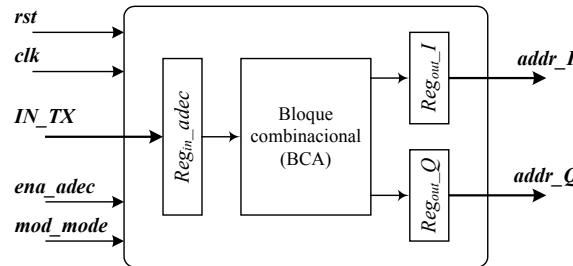


Figura 4.2: Constelaciones con las que puede operar el transmisor ST/DDST; a) 4-QAM, b) 16-QAM, c) 64-QAM (se delimitan también las constelaciones anteriores).

necesaria debido a que —como se puede apreciar en la figura 4.2— no todos los puntos situados en la misma posición en las tres constelaciones mapean al mismo símbolo complejo de salida; p. ej., mientras que en QPSK, el punto 2 de la constelación se mapeará al símbolo complejo $-1 + j$, 16-QAM lo hará al $3 - 3j$ y en 64-QAM al símbolo $1 + 5j$.



(a)

Símbolo de entrada 4-QAM			Posición en la const. 64-QAM		Salida en código Gray de la posición en la const. 64-QAM							
Decimal	Binario		Decimal	Componente I				Componente Q				Decimal
	b_1	b_0		s_5	s_4	s_3	s_2	s_1	s_0	I	Q	
0	0	0	9	0	0	1	0	0	0	1	1	1
1	0	1	13	0	0	1	1	0	0	1	1	-1
2	1	0	41	1	0	1	0	0	0	1	-1	1
3	1	1	45	1	0	1	1	0	0	1	-1	-1

(b)

Símbolo de entrada 4-QAM					Posición en la const. 64-QAM		Salida en código Gray de la posición en la const. 64-QAM						
Decimal	Binario				Decimal	Componente I				Componente Q			Decimal
	b_3	b_2	b_1	b_0		s_5	s_4	s_3	s_2	s_1	s_0	I	Q
0	0	0	0	0	0	0	0	0	0	0	0	0	3
1	0	0	0	1	1	0	0	0	0	0	0	1	3
2	0	0	1	0	4	0	0	0	1	0	0	0	-3
3	0	0	1	1	5	0	0	0	1	0	0	1	-1
4	0	1	0	0	8	0	0	1	0	0	0	0	3
5	0	1	0	1	9	0	0	1	0	0	0	1	1
6	0	1	1	0	12	0	0	1	1	0	0	0	-3
7	0	1	1	1	13	0	0	1	1	0	0	1	-1
8	1	0	0	0	32	1	0	0	0	0	0	0	3
9	1	0	0	1	33	1	0	0	0	0	0	1	-3
10	1	0	1	0	36	1	0	0	1	0	0	0	-3
11	1	0	1	1	37	1	0	0	1	0	0	1	-1
12	1	1	0	0	40	1	0	1	0	0	0	-1	3
13	1	1	0	1	41	1	0	1	0	0	0	1	1
14	1	1	1	0	44	1	0	1	1	0	0	-1	-3
15	1	1	1	1	45	1	0	1	1	0	0	1	-1

(c)

Figura 4.3: El adecuador de símbolos; a) Diagrama, b) Tabla de verdad para adecuación de símbolos QPSK, c) Tabla de verdad para adecuación de símbolos 16-QAM.

La figura 4.3a muestra el diagrama del módulo de adecuación de símbolos, que en esencia es un bloque combinacional (BCA) situado entre un registro de entrada (Reg_{in_adec}) y dos de salida (Reg_{out_I} y Reg_{out_Q}). Analizando las tablas de verdad presentadas en las figuras 4.3b y 4.3c, —las cuales corresponden a la adecuación de los símbolos QPSK y 16-QAM a la constelación 64-QAM, respectivamente— resulta sencillo derivar las siguientes expresiones lógicas para las salidas $addr_I$ y $addr_Q$ del bloque combinacional.

$$addr_I = \begin{cases} conc(b_1, 0, 1), & \text{para 4-QAM} \\ conc(b_3, 0, b_2), & \text{para 16-QAM} \\ conc(b_5, b_4, b_3), & \text{para 64-QAM} \end{cases} \quad (4.5)$$

$$addr_Q = \begin{cases} conc(b_0, 0, 1), & \text{para 4-QAM} \\ conc(b_1, 0, b_0), & \text{para 16-QAM} \\ conc(b_2, b_1, b_0), & \text{para 64-QAM} \end{cases} \quad (4.6)$$

donde $conc(\cdot)$ denota el operador lógico de concatenación.

4.3.2. El mapeador

Tal como se estableció en el apartado 4.3.1, para el diseño de éste módulo solo se requiere de la constelación 64-QAM. Para construirla, en este trabajo se propone un esquema eficiente en términos de consumo de memoria, ya que únicamente se almacenan los valores $(1, 3, 5, 7, -1, -3, -5, -7)$ que puede tomar cada una de las componentes I, Q. Adicionalmente, para generar el símbolo complejo de salida, el mapeador debe contar con una etapa que normalice las señales de salida de la constelación en base a dos criterios: el orden de la constelación QAM y la cantidad de potencia que se asigna a cada una de las secuencias involucradas en la transmisión. Su implementación directa en *hardware* implica dos multiplicadores, mismos que pueden reducirse a sólo uno por medio de las siguientes adecuaciones.

La constante de normalización para una constelación QAM viene dado por

$$Cte_Norm_QAM = \begin{cases} 1/\sqrt{2}, & \text{para 4-QAM} \\ 1/\sqrt{10}, & \text{para 16-QAM} \\ 1/\sqrt{42}, & \text{para 64-QAM.} \end{cases} \quad (4.7)$$

Por otro lado, en lo que concierne a las potencias de las secuencias, se tiene que cumplir que $\sigma_b^2 + \sigma_c^2 = 1$ para ST y $\sigma_b^2 + \sigma_c^2 + \sigma_e^2 = 1$ para DDST; donde σ_e^2 denota la potencia de la secuencia de entrenamiento dependiente de los datos. Por tal motivo, se precisa una expresión para σ_b^2 que la relacione con ambos modos de operación del transmisor. Así, en [66] se menciona que:

$$\sigma_b^2 = \frac{(1 - \sigma_c^2) N_P}{N_P - 2\alpha_{IT} + \alpha_{IT}^2}, \quad (4.8)$$

donde $\alpha_{IT} = 0$ para ST y 1 para DDST. De ésta última ecuación es posible representar los términos correspondientes a cada uno de los modos de operación de transmisor, con lo que:

$$\sigma_{bST}^2 = 1 - \sigma_c^2 \quad \sigma_{bDDST}^2 = \frac{N_P}{N_P - 2\alpha_{IT} + \alpha_{IT}^2}.$$

En consecuencia, para obtener una constante única de normalización para el mapeador (*Cte_Norm_Mapp*), es suficiente con combinar (4.7) y (4.8), esto es

$$Cte_Norm_Mapp = \sigma_b \times Cte_Norm_QAM. \quad (4.9)$$

En la figura 4.4 se muestra la estructura final del mapeador, donde se puede apreciar que para la LUT de las constelaciones se emplea una ROM de doble puerto 8×4 bits, mientras que el tamaño de la ROM para la LUT con las constantes de normalización, dependerá de los modos de operación del transmisor y el numero de bits con los que se quiera cuantizar la constante en cuestión.

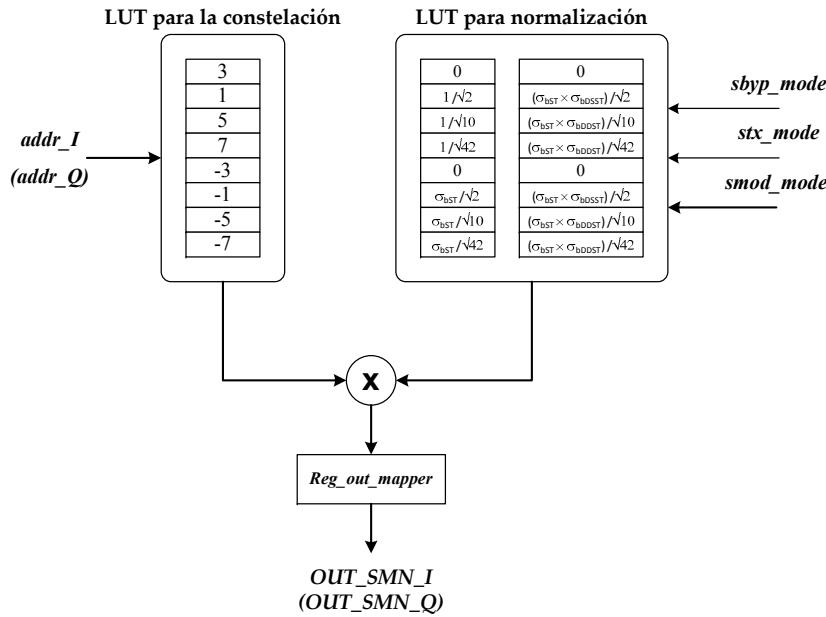


Figura 4.4: Arquitectura interna del mapeador.

Debido a que en el diseño del transmisor se estableció que $\sigma_c^2 = 0.2$ [66] y $Q_W = 16$ bits, fue necesario disponer de una ROM de 16×16 bits en conjunto con los valores para las constantes de normalización que se muestran en la tabla 4.2. Un multiplicador combina los valores enviados desde las LUT en cada instante de símbolo y el resultado se almacena en el registro de salida *Reg_out_mapper*.

Modo de operación	Expresión	Nombre de la constante	Valor decimal	Valor hexadecimal	Q _I	Q _F
Mapeador sin IT	$1/\sqrt{2}$	Cte_Norm_Qam4	0.7071	h'5A82	1	15
	$1/\sqrt{10}$	Cte_Norm_Qam16	0.3162	h'287A	1	15
	$1/\sqrt{42}$	Cte_Norm_Qam64	0.1543	h'13C0	1	15
Mapeador con ST	$\sigma_{bST}/\sqrt{2}$	Cte_Norm_QamS4	0.6324	h'50F4	1	15
	$\sigma_{bST}/\sqrt{10}$	Cte_Norm_QamS16	0.2828	h'2434	1	15
	$\sigma_{bST}/\sqrt{64}$	Cte_Norm_QamS64	0.1380	h'11AA	1	15
Mapeador con DDST	$(\sigma_{bST} \times \sigma_{bDSST})/\sqrt{2}$	Cte_Norm_QamD4	0.6375	h'5198	1	15
	$(\sigma_{bST} \times \sigma_{bDSST})/\sqrt{10}$	Cte_Norm_QamD16	0.2851	h'247D	1	15
	$(\sigma_{bST} \times \sigma_{bDSST})/\sqrt{64}$	Cte_Norm_QamD64	0.1391	h'11CE	1	15

Tabla 4.2: Cuantización de las constantes del mapeador para $\sigma_c^2 = 0.2$ y $Q_W = 16$ bits.

4.3.3. Módulo transformador de secuencia

En términos de *hardware*, el transformador de secuencia es el módulo de mayor complejidad del transmisor IT, esto debido a su característica de configurabilidad que le permite conmutar entre los dos distintos modos de operación. Por tal motivo, su diseño se ha particionado en tres submódulos, cuyas arquitecturas individuales se describen a continuación.

4.3.3.1. Generador de la secuencia de entrenamiento

Retomando lo expuesto en 3.2.10 y analizando (3.41), se percibe que los parámetros σ_c , N y P necesarios para construir la secuencia OCI son conocidos *a priori* y permanecen invariantes durante la operación del sistema. Por tal motivo, la secuencia de entrenamiento permanece constante y como tal, sus valores pueden ser calculados *off-line*, cuantizados y posteriormente almacenados en una LUT. La tabla 4.3 resume los resultados de este procedimiento para generar la secuencia de entrenamiento con $\sigma_c^2 = 0.2$,

Elemento de la secuencia OCI	Valor decimal		Valor decimal (normalizado a σ_c)		Valor hexadecimal		Formato	
	Real	Imag	Real	Imag.	Real	Imag.	Q _I	Q _F
c(0)	1	0	0.4472	0	h'393E	h'0000	1	15
c(1)	0.3827	0.9239	0.1711	0.4132	h'15E8	h'34E3	1	15
c(2)	-1	0	-0.4472	0	h'C6C2	h'0000	1	15
c(3)	0.9239	-0.3827	0.4132	-0.1711	h'34E3	h'EA18	1	15
c(4)	-1	0	-0.4472	0	h'C6C2	h'0000	1	15
c(5)	0.3827	0.9239	0.1711	0.4132	h'15E8	h'34E3	1	15
c(6)	1	0	0.4472	0	h'393E	h'0000	1	15
c(7)	0.9239	-0.3827	0.4132	-0.1711	h'34E3	h'EA18	1	15

Tabla 4.3: Cuantización de la secuencia de entrenamiento para $\sigma_c^2 = 0.2$, $P = 8$, $N = 512$ y $Q_W = 16$ bits.

$N = 512$, $P = 8$ [66] y $Q_W = 16$ bits, misma que fué almacenada en una ROM de 8×32 bits, similar a la presentada en la figura 4.5. Dicha memoria es leída N_p veces con la finalidad de que la secuencia OCI sea del mismo tamaño N que la de datos y así puedan superponerse elemento a elemento con la ayuda de un sumador complejo.

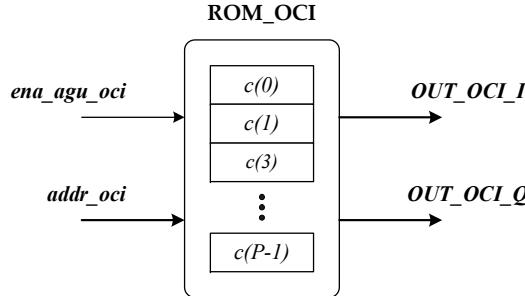


Figura 4.5: ROM para la LUT con la secuencia de entrenamiento.

4.3.3.2. Submódulo de inserción del prefijo cíclico

Existen varios problemas en el diseño de este bloque derivados de la naturaleza del prefijo cíclico y la posición que toma lugar en la trama final. Analizando la figura 4.6 —la cual corresponde a la secuencia ST con prefijo cíclico— se puede evidenciar lo siguiente:

1. A consecuencia de que el CP está conformado de los últimos P elementos de la secuencia ST, sólo puede ser generado de la secuencia en cuestión hasta que ésta haya sido procesada en su totalidad.
2. Debido a que en todo bloque de datos a transmitir lo primero que se envía es el CP, resulta inevitable usar un elemento de memoria temporal para almacenar la secuencia de datos que le precede y de esta forma prevenir pérdida de datos.

Ante lo mencionado, una memoria de $N + P$ localidades es una opción para guardar toda la trama (datos más CP) y después leerla en su totalidad, a costa de introducir una latencia de $N + P$ ciclos. Por tal motivo, para disminuir en lo posible la latencia y el tamaño de la memoria, la solución fue usar una RAM de $N \times 2Q_W$ bits (RAM_CP en la figura

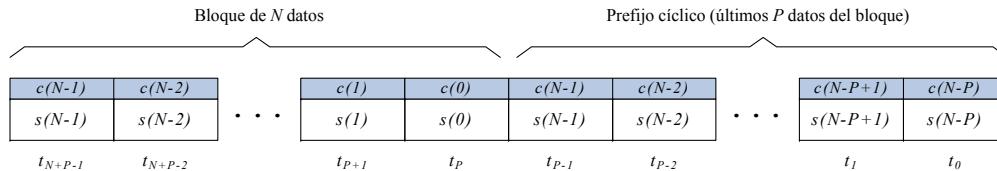


Figura 4.6: Secuencia ST con prefijo cíclico incorporado.

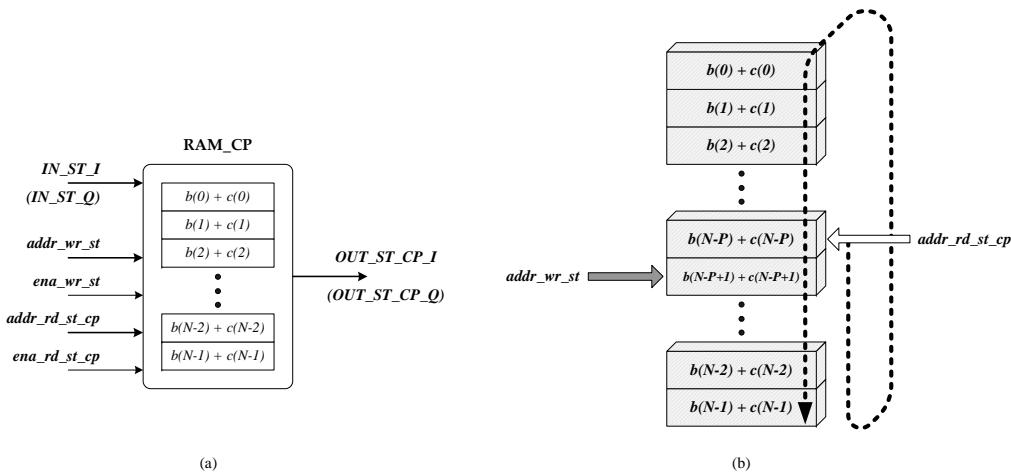


Figura 4.7: Submódulo de inserción de prefijo cíclico; a) Implementación con memoria RAM con buses de direccionamiento separado para lectura y escritura, b) Proceso de almacenamiento de la secuencia de datos e inserción del prefijo cíclico.

4.7a) con buses de direccionamiento separado para lectura y escritura ($addr_rd_st_cp$ y $addr_wr_st$, respectivamente en la figura 4.7a). Esta característica permite leer el primer dato del prefijo cíclico ($b(N - P) + c(N - P)$) un ciclo después de que ha sido grabado en la memoria y enviarlo por la salida $OUT_ST_CP_I$ no importando que aún falten datos de la secuencia por almacenarse, tal como se ilustra en la figura 4.7b. Una vez que finalice la escritura/lectura de los P últimos elementos de la secuencia de datos que corresponden al CP, se deshabilitará la operación de escritura en la RAM_{CP}, el apuntador $addr_rd_st$ se posicionará en la primera localidad y se procederá a la lectura de la memoria en su totalidad para enviar ahora la secuencia de datos. Este esquema permite un ahorro de P localidades en el tamaño de la RAM y una reducción de $2P$ ciclos en la latencia.

4.3.3.3. Submódulo generador de la secuencia de entrenamiento dependiente de los datos

La media cíclica de la secuencia de datos $b(k)$ definida por (3.3), es la operación en la que se fundamenta el diseño de éste submódulo. En ella se tienen que realizar los procesos que se muestran en la figura 4.8 y que se describen a continuación:

1. La secuencia $b(k)$ se reordena en una matriz de $P \times N_P$ elementos acorde a la regla de correspondencia $[\mathbf{B}_{SDD}^T]_{i,j} = [\mathbf{b}]_{iP+j} = b(iP + j)$.
2. En cada una de las P filas de la matriz \mathbf{B}_{SDD}^T , se suman uno a uno los N_P elementos que las conforman. Los resultados obtenidos en cada fila se dividen entre N_P para obtener los elementos de $e(j)$.

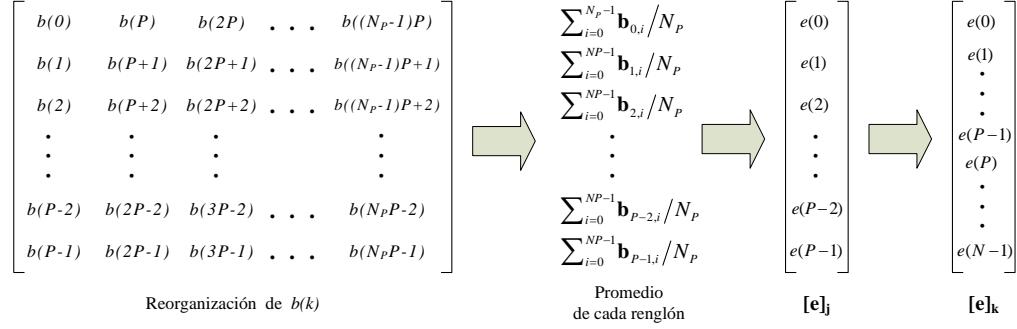


Figura 4.8: Proceso para generar la secuencia de entrenamiento dependiente de los datos a partir de la secuencia $b(k)$.

- Para que la SDD $e(k)$ tenga una longitud final de N , N_p réplicas de la secuencia $e(j)$ tienen que concatenarse y así pueda superponerse elemento a elemento a $b(k)$ y $c(k)$.

Se puede notar que, de forma similar con lo que sucede en la inserción del CP, sólo es posible generar la SDD hasta que toda la secuencia $b(k)$ ha sido procesada. Nuevamente no hay forma de evitar la latencia (de hasta N ciclos) ocasionada por esto. Consecuentemente, para mitigar estos inconvenientes se propone un esquema novedoso para generar la SDD a través de la arquitectura presentada en la figura 4.9a, en donde por principio de cuentas se evita la reorganización de $b(k)$ en la matriz \mathbf{B}_{SDD}^T . Esto se logra realimentando —a través del registro de desplazamiento lb_delay_sdd — los datos de una de las salidas de la RAM (RAM_SDD) de doble puerto hacia el sumador complejo. La realimentación produce un retraso de P símbolos que permite alinear los datos que corresponden a ca-

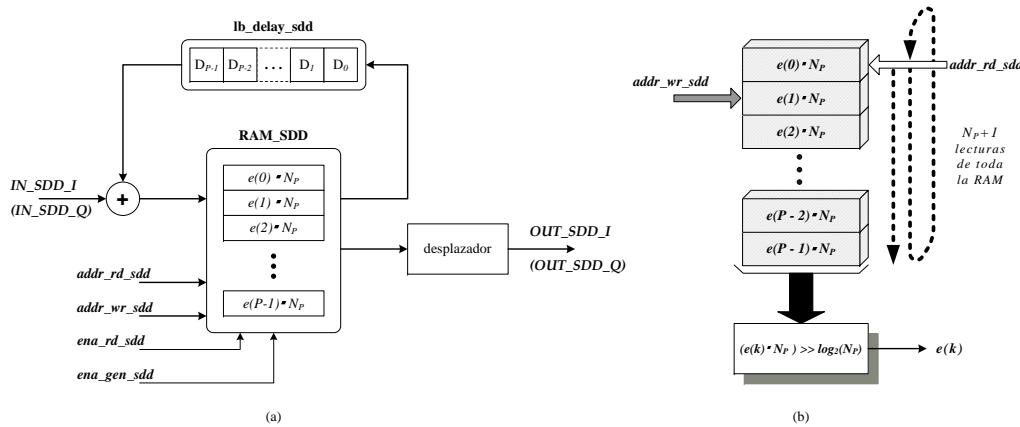


Figura 4.9: Submódulo generador de la secuencia de entrenamiento dependiente de los datos; a) Arquitectura simplificada, b) Proceso de replicación de $e(j) \cdot N_p$ y división entre N_p para formar $e(k)$.

da renglón y de esta forma puedan ser sumados “al vuelo” conforme van llegando al generador SDD sin detener su flujo. Por consiguiente, la sumatoria del primer renglón se obtiene P ciclos antes de que se termine de procesar $b(k)$. Luego entonces, un ciclo después se activa el bus de direcciones $addr_rd_sdd$ del segundo puerto de salida de RAM_CP y se posiciona en la localidad inicial para proseguir con las $N_P + 1$ lecturas de toda la memoria (figura 4.9b). La primera de ellas corresponde al CP y las restantes son para expandir la secuencia $e(j)$ escalada por N_P a su tamaño final N . Finalmente, el escalamiento en los datos de $e(k)$ se suprime dividiendo entre N_P a cada uno de sus elementos. Un bloque *desplazador* aplica $\log_2(N_P)$ desplazamientos aritméticos hacia la izquierda a cada dato y envía los resultados de cada componente a las salidas OUT_SDD_I y OUT_SDD_Q respectivamente.

4.3.4. Generador de direcciones (AGU_Tx) y unidad de control

Es el módulo encargado de generar los valores de direcciones válidas para las memorias ROM_OC, RAM_CP y RAM_SDD del transmisor. Para esto, dispone de dos contadores módulo N ; Count_agu_1 y Count_agu_2, el primero se activa al inicio de la operación de transmisor y el segundo lo hace N ciclos después. El bus de direcciones $addr_wr_st$ se construye directamente con la salida proveniente de Count_agu_1, mientras que sus $\log_2(P)$ bits menos significativos (LSB) extraídos con el bloque Slice_bus_1 sirven para generar tanto a $addr_oci$ como a $addr_wr_sdd$, tal como lo sugiere la arquitectura de la figura 4.10. Un procedimiento similar se usa con la salida de Count_agu_2, y con la ayuda del bloque Slice_bus_2 para obtener los buses $addr_rd_st_cp$ y $addr_rd_sdd$.

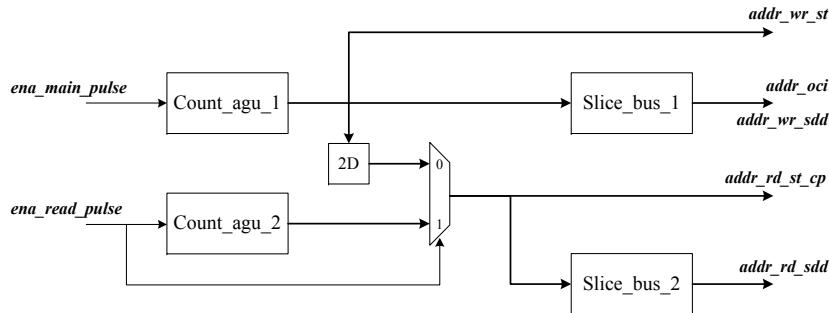


Figura 4.10: Estructura interna del generador de direcciones del transmisor ST/DDST.

En lo que concierne a la unidad de control, su implementación está basada en una máquina de estados finitos (FSM) con tres estados. Ella es la encargada de generar la señalización necesaria para todas las secuencias de eventos dentro de la arquitectura del transmisor dependiendo de número de muestra, las señales a su entrada; p. ej. las entradas que definen el modo de operación (*tx_mode*) o el orden de la constelación QAM (*mod_mode*), y el estado en cuestión.

4.4. Resultados de implementación y simulación

Se describió en su totalidad a nivel RTL usando Verilog HDL, un transmisor ST/DDST con parámetros de operación de $N = 512$ símbolos, $CP = 8$, $P = 8$, $\sigma_c^2 = 0.2$, con una $Q_W = 16$ bits en la mayoría de las señales y se sintetizó en un FPGA Virtex-5 de Xilinx bajo las siguientes consideraciones:

- Se habilitó los valores de síntesis por defecto (*default settings*) y no se seleccionó alguna restricción de usuario (*user constraints*).
- No se usó ningún componente pre-diseñado proveniente de alguna librería de módulos.
- Se aplicó un esquema de redondeo de truncamiento simple ya que no requiere *hardware* adicional para su implementación.
- Todas las señales se representaron en punto fijo con aritmética signada en complemento a dos.

Los resultados de la síntesis del transmisor ST/DDST propuesto se resumen en la tabla 4.4. En ella se exponen el total disponible de cada uno de los recursos del FPGA Virtex-5, la cantidad consumida por la arquitectura y el porcentaje que esto representa. Analizándola, se observa una frecuencia de operación de 160 MHz con un consumo de recursos muy reducido, con lo que queda de manifiesto que la arquitectura presenta una excelente relación velocidad–área. Por otro lado, no fue posible comparar los resultados de síntesis del transmisor ST/DDST con las únicas implementaciones reportadas en la literatura [42, 43]. En la primera, a causa de que el diseño del transmisor es *full-software*, no existe un proceso de síntesis como tal y en la segunda porque sólo se implementó la parte receptora.

Recursos del FPGA	Registros (<i>Slice registers</i>)	LUTs (<i>Slice LUTs</i>)	Pares LUT-FF (<i>Fully used LUT-FF pairs</i>)	IOBs (<i>Bonded IOs</i>)	BRAMs (<i>Block RAMs</i>)
Total	69120	69120	444	640	148
Usado	141	437	134	46	4
Porcentaje	<1%	<1%	30%	7%	2%

Frecuencia de operación: 160.12 MHz

Tabla 4.4: Resultados de síntesis para el transmisor ST/DDST con $\sigma_c^2 = 0.2$, $P = 8$, $N = 512$ y $Q_W = 16$ bits.

En sentido amplio, el procedimiento de verificación para el *hardware* del transmisor ST/DDST —así como el de la mayoría de las arquitecturas planteadas en este trabajo de tesis— queda conceptualizado en el diagrama de la figura 4.11. En el caso específico del transmisor la verificación se realiza tanto a nivel funcional como de su rendimiento en punto fijo en términos del SQNR. Para tal fin, se ejecuta un conjunto de simulaciones

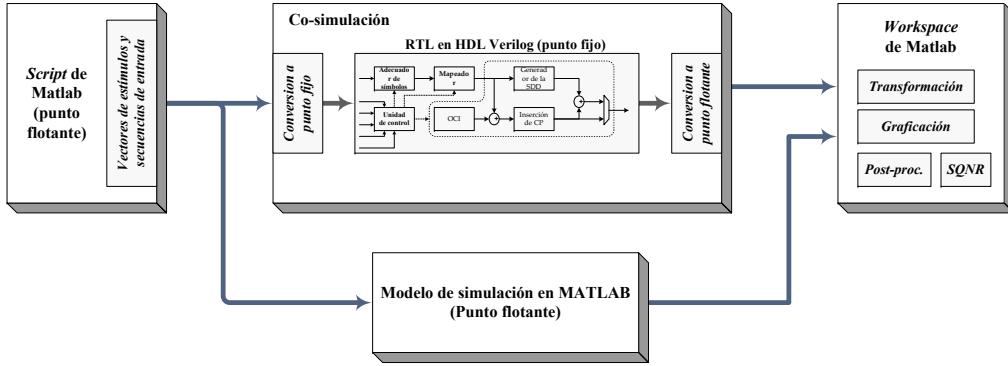


Figura 4.11: Esquema general de verificación para las arquitecturas de *hardware*.

Monte Carlo, generando aleatoriamente en cada una de ellas la secuencia $b(k)$ a transmitir y comparando los resultados contra los obtenidos en punto flotante con el modelo de oro de simulación.

Así, la figura 4.12 muestra la gráfica de la secuencia de entrenamiento con potencia σ_c^2 obtenida tanto por el *hardware* del transmisor operando en modo ST como por el modelo de simulación de Matlab. Se puede apreciar que la diferencia entre ellas es imperceptible al grado de que las trazas en las gráficas se superponen. El mismo comportamiento se observa en las gráficas de la figura 4.13 al comparar ahora la secuencia de salida $s(k)$. Es de resaltarse el grado de fidelidad que se logra al generar $s(k)$ con la arquitectura del transmisor a pesar de emplear aritmética de precisión finita.

Ante la imposibilidad de poder verificar en las señales de la figura 4.13 que el *hardware* del transmisor (en modo ST) ha incorporado en forma correcta la señal de entrenamiento $c(k)$ en los datos $b(k)$, se transformó la secuencia de salida ST $s(k)$ al dominio

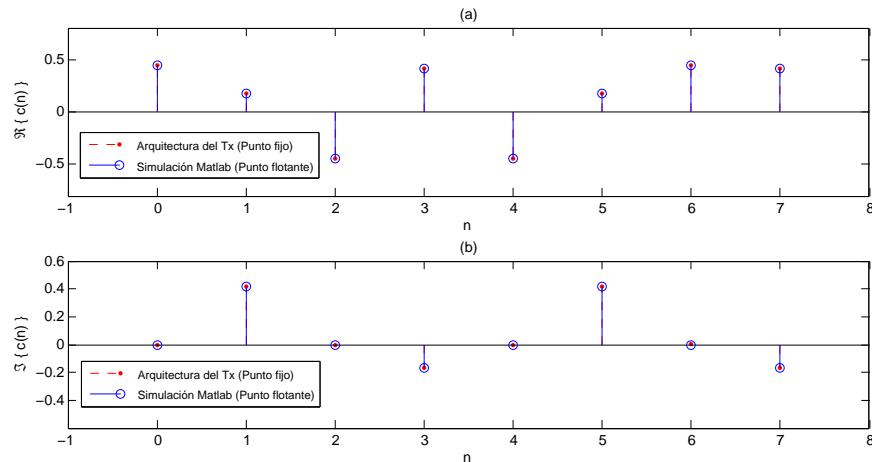


Figura 4.12: Secuencia de entrenamiento obtenida con el *hardware* del transmisor y por medio de simulación con Matlab; a) Parte real, b) Parte imaginaria.

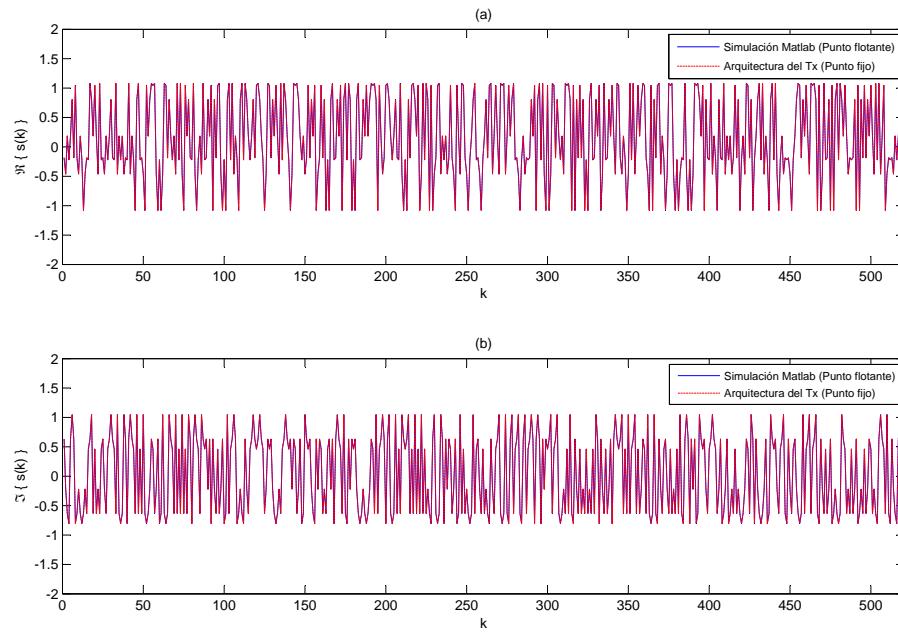


Figura 4.13: Secuencia de salida ST obtenida tanto con la arquitectura del transmisor como por medio del modelo de simulación con Matlab; a) Parte real, b) Parte imaginaria.

de la frecuencia por medio de la DFT. Esto arroja como resultado la señal que se muestra en la figura 4.14, donde se observa como la energía de $b(k)$ se dispersa en todas las componentes de frecuencia mientras para el caso de la secuencia de entrenamiento, la energía se concentra únicamente en las P componentes de frecuencia equiespaciadas.

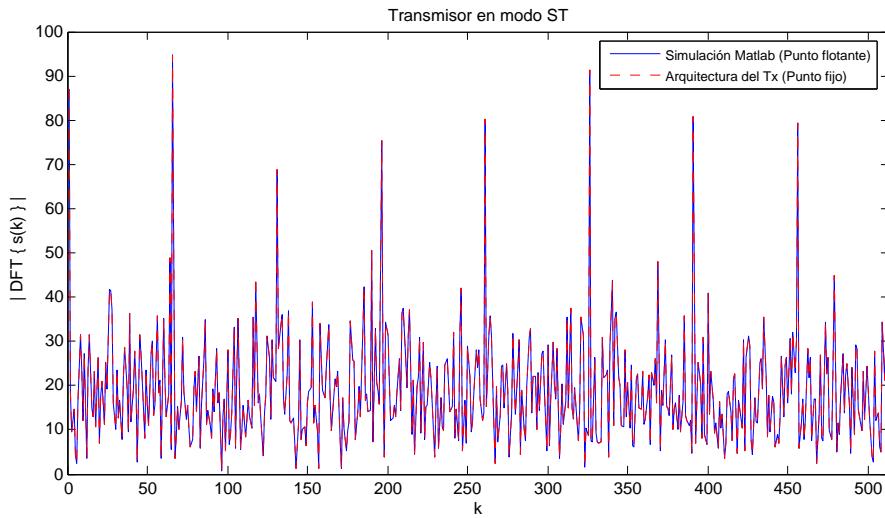


Figura 4.14: Transformada discreta de Fourier de la secuencia ST del transmisor.

De forma similar, se continuó con la verificación del modo de operación DDST del transmisor. Por consiguiente, fue necesario corroborar que la secuencia de entrenamiento dependiente de los datos $e(n)$ —clave en este modo de funcionamiento— se estuviera generando correctamente en la arquitectura. Al respecto, la figura 4.15 presenta gráficamente la SDD generada tanto por el *hardware* como por el modelo de Matlab. Cabe señalar que visualmente no es posible determinar diferencias significativas entre ambas señales. Similarmente, las gráficas para la secuencia de salida $s(k)$ en modo DDST presentadas en la figura 4.16 se traslanan entre sí, sin exhibir discrepancias relevantes que permitan inferir un mal funcionamiento del transmisor. Asimismo, en la gráfica de la figura 4.17 correspondiente a la transformada discreta de Fourier de la secuencia de salida DDST, se identifica claramente los pilotos de la secuencia OCI. No obstante a diferencia del modo ST, los pilotos ademas de estar equi-espaciados, todos tienen la misma energía, prueba inequívoca de que el transmisor esta superponiendo correctamente $c(k)$ y $e(k)$ en $b(k)$.

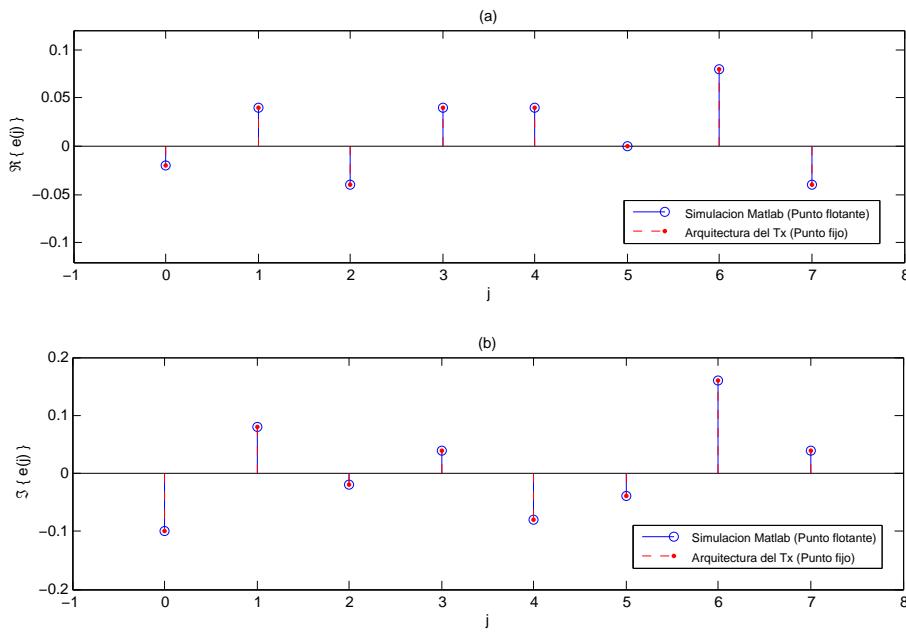


Figura 4.15: Secuencia dependiente de los datos $e(n)$ obtenida tanto con la arquitectura del transmisor como por el modelo de simulación; a) Parte real, b) Parte imaginaria.

En referencia a las gráficas anteriores, queda de manifiesto que el desempeño funcional de la arquitectura es el adecuado, ya que es capaz de reproducir las señales en forma similar a las que se obtienen con el modelo de oro de simulación. No obstante, aún resta cuantificar a través del SQNR el grado de fidelidad de la señales de salida de la arquitectura del transmisor. Esto implica una verificación más exhaustiva, ya que al ser los datos $b(k)$ una variable aleatoria, se necesita un número representativo de realizaciones para tener un resultado confiable. Luego entonces, los histogramas de la figura 4.18

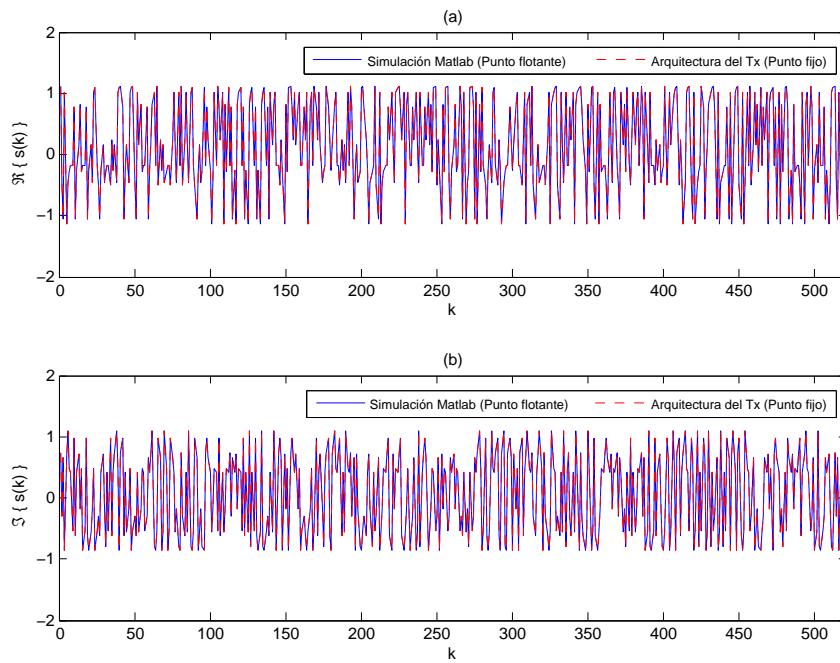


Figura 4.16: Secuencia de salida DDST obtenida tanto con la arquitectura del transmisor como por medio del modelo de simulación con Matlab; a) Parte real, b) Parte imaginaria.

muestran el resultado para el SQNR del *hardware* del transmisor para 100 simulaciones Monte Carlo, eligiendo el orden de la constelación en forma aleatoria. Al principio considerando solo un modo de operación a la vez y al final estableciéndolo igualmente aleatoriamente.

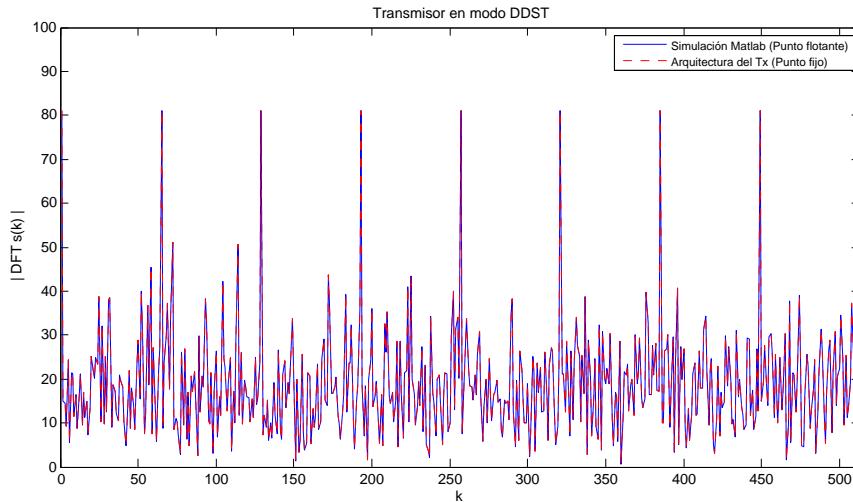


Figura 4.17: Transformada discreta de Fourier de la secuencia DDST del transmisor.

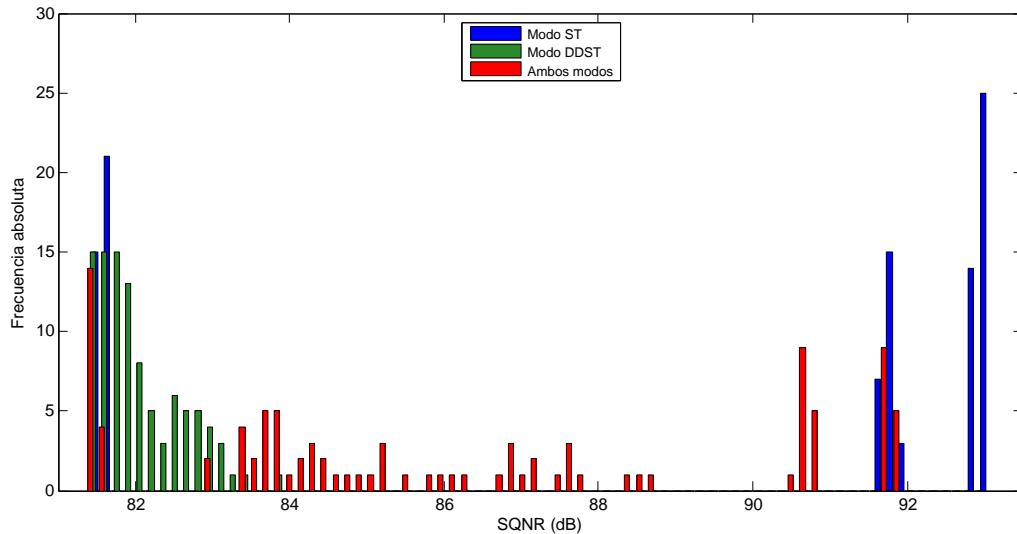


Figura 4.18: Histograma del SQNR de la arquitectura del transmisor ST/DDTS para 100 simulaciones Monte Carlo.

Estas últimas gráficas revelan en el *hardware* un rendimiento en el SQNR de 82 a 92 dB sin importar el modo de operación del transmisor. De igual forma, los resultados en cada caso nos dan un SQNR promedio de 88.51 dB para el modo ST, 85.46 dB para el DDST y de aproximadamente 86.80 dB en forma combinada, valores bastante respetables si se tiene en cuenta que un SQNR de 40 dB se considera como el mínimo aceptable.

Conclusiones del capítulo

En este capítulo se expusieron las diversas problemáticas que se tuvieron que enfrentar en el desarrollo del transmisor ST/DDST y como cada una de ellas fue resuelta manteniendo siempre como meta de diseño el tener un buen *trade-off* entre rendimiento y utilización de *hardware*. De hecho, tiene mucho que ver en cómo cada uno de las pequeñas pero novedosas mejoras de diseño —introducidas en los diversos módulos del transmisor— al final ya vistas en conjunto logran que:

- El consumo total de memoria sea reducido drásticamente al grado de que solo se necesiten $3P + N$ localidades.
- Se tenga un rendimiento excelente de SQNR superior a los 80 dB sin incrementar en exceso la longitud de palabra.
- El alto grado de reusabilidad con el que cuenta el sistema final permite integrar en una sola arquitectura configurable tres ordenes de constelación QAM y tres

modalidades de operación, todo funcionando a una velocidad de 160 MHz con un consumo simbólico de recursos del FPGA Virtex-5.

Todo lo anteriormente expuesto quedó plasmado en el artículo:

Romero-Aguirre, E.; Parra-Michel, R.; Carrasco-Alvarez, R.; Orozco-Lugo, A.G., Configurable transmitter and systolic channel estimator architectures for data-dependent superimposed training communications systems, *International Journal of Reconfigurable Computing*, 2012, Hindawi.

Capítulo 5

Desarrollo de arquitecturas de PDS para un receptor DDST

5.1. Introducción

Como se mencionó en párrafos anteriores, la parte receptora en cualquier sistema inalámbrico es de mucho mayor complejidad que la transmisora. En el caso de un receptor DDST, la aseveración anterior no es la excepción y puede corroborarse de inmediato al observar su arquitectura (propuesta) en la figura 5.1, la cual está diseñada para afrontar la mayoría de las perturbaciones presentadas en la sección 2.3.2. Para esto, el receptor debe inicialmente recibir un bloque de $2(N + P)$ muestras y almacenarlo en el *buffer* principal. A partir de dicho bloque debe efectuar un estimado burdo de desplazamiento de CD, un estimado del CFO y compensarlos. Su siguiente tarea consiste en encontrar el desfase (en muestras) existente entre transmisor-receptor a nivel secuencia de entrenamiento. Hecho esto, debe también estimar el desfase que tiene con el transmisor pero ahora a nivel trama y corregirlo. En este punto, el receptor ya está en condiciones de encontrar la CIR del canal junto con el desplazamiento de CD, el cual es removido de las muestras con la ayuda de un sumador. Los coeficientes del canal se alimentan al ecualizador para reestablecer los símbolos en la constelación y estos se envían posteriormente al *buffer* temporal 2. Al final, el detector lee las muestras almacenadas en dicho *buffer* con el objetivo de recuperar cada uno de los datos originalmente transmitidos.

El tiempo y la cantidad de procesamiento para que el receptor DDST recupere la señal transmitida va en relación directa con el número de perturbaciones que se requiera estimar y compensar. Esto se acentúa si se toma en cuenta que, tal como lo sugiere la figura 5.1, para iniciar los procesos de cada módulo, es absolutamente necesario que todas las módulos previos hayan finalizado sus tareas. Esta gran dependencia quita la posibilidad de que varios módulos puedan trabajar en paralelo y así disminuir los tiempos de procesamiento. Tal estructura altamente secuencial de procesos en el receptor se conoce

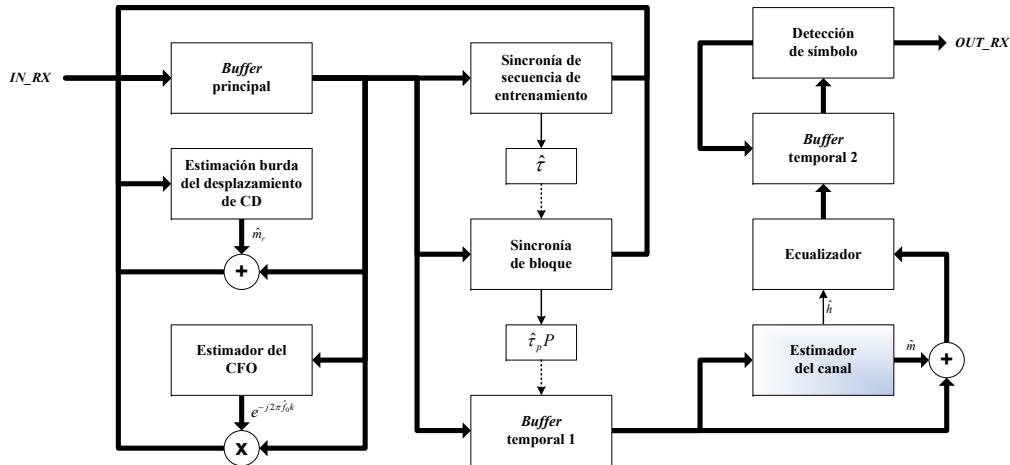


Figura 5.1: Arquitectura simplificada de un receptor DDST. (Por legibilidad se ha omitido los módulos generadores de direcciones y de control).

como cadena de algoritmos de estimación.

Precisamente esta diferencia tan marcada es la que permite que el desarrollo de la arquitectura final del receptor pueda partitionarse, en forma más directa y sin ambigüedades, en los módulos que se indican en la figura 5.1. Posteriormente, para el diseño individual de cada uno de ellos se usará el método planteado en 4.2, tal y como se hizo con el transmisor.

En primer lugar —debido a su relevancia y por estar presente en la mayoría de los receptores inalámbricos invariablemente del método de entrenamiento— se desarrolló la arquitectura digital para el módulo de estimación de canal. Por otro lado, al ser operaciones de PDS de uso intensivo, la FFT y la correlación también son consideradas en este capítulo.

5.2. Arquitecturas de *hardware* de estimadores de canal para DDST bajo sincronía perfecta y sin desplazamiento de CD

En esta apartado de describirá a detalle el desarrollo de tres arquitecturas para estimación de canal en DDST. A pesar de que todas ellas están basadas en el algoritmo presentado en 3.2.2, sus paradigmas de diseño son distintos y básicamente enfocados a su futura reutilización en los subsecuentes módulos del receptor DDST. La primera arquitectura, denominada como **estimador de canal con alimentación de datos desde memoria** (CEDAM), el *hardware* ejecuta su procesamiento a partir de un bloque de datos que ha sido almacenado previamente en memoria. En la segunda, a la cual nos referiremos en lo subsecuente como **estimador de canal con alimentación de datos al**

vuelo (CEO), la estimación del canal se lleva a cabo al mismo tiempo que el flujo de datos se está recibiendo y además de forma paralela, tales datos se van almacenando en memoria. En la tercera, un arreglo sistólico de procesadores es el encargado de ejecutar todos los cálculos necesarios en el algoritmo. Por tal motivo, la arquitectura se referencia en este trabajo como **estimador de canal DDST sistólico** (SYSDCE). Vale la pena mencionar que en todos ellos se considera que P debe ser una potencia de dos, N es un múltiplo de P , $P = L$ y todos las operaciones se realizan sobre señales con valores complejos.

5.2.1. Arquitectura digital del estimador CEDAM

La arquitectura simplificada para este estimador se muestra en la figura 5.2 y está conformada de cuatro módulos un *buffer* de memoria (MB), un estimador de la media cíclica paralelo (PCME), un multiplicador matriz-vector sistólico (SMVM) y una LUT para \mathbf{C}^{-1} . El proceso para calcular el canal estimado a través de cada uno de tales módulos se describe a continuación. Tan pronto se activa la señal de entrada *start_cedam*, el módulo

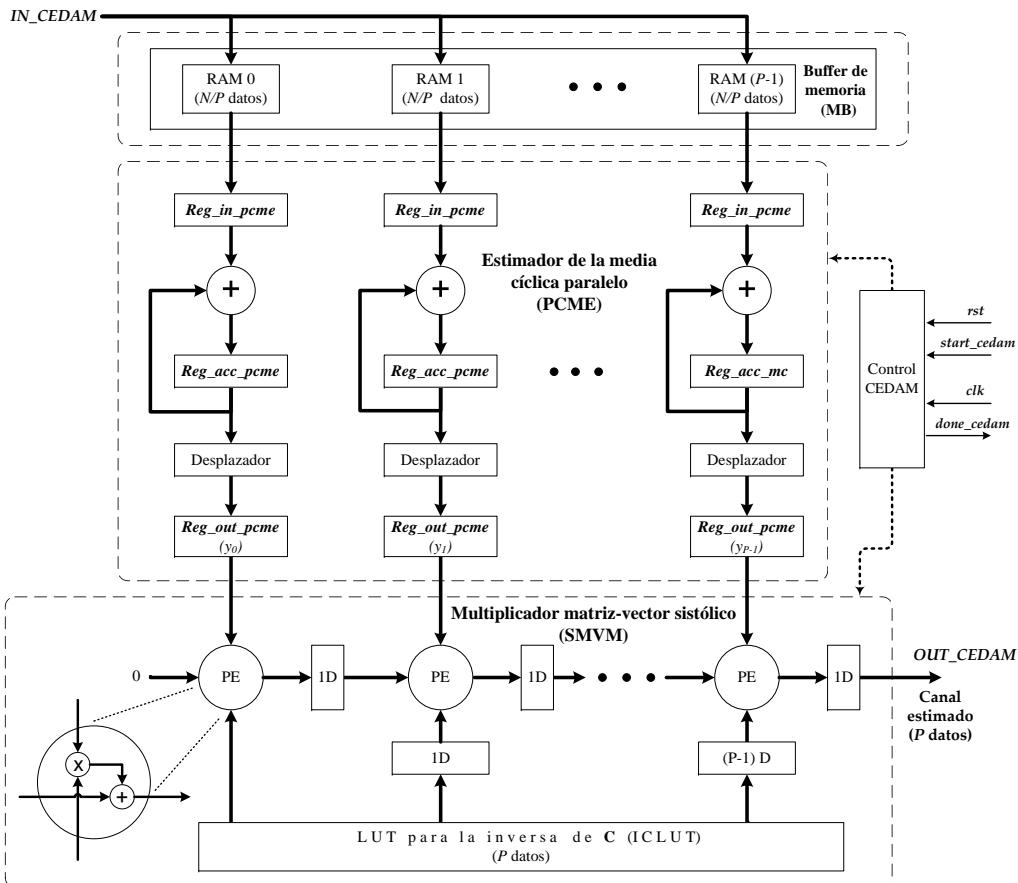


Figura 5.2: Arquitectura simplificada del estimador de canal CEDAM.

lo PCME durante N_P ciclos, procesa en paralelo P datos a la vez de la secuencia $x(k)$, tomando ventaja de que éstos ya estaban almacenados en MB. Una vez que la media cíclica \bar{y} de la secuencia recibida $x(k)$ ha sido estimada, el módulo SMVM realizará el producto expresado en (3.7). Transcurridos $P + 1$ ciclos, la salida *done_cedam* se activa y uno por uno los valores del canal estimado $\hat{\mathbf{h}}$ se envían al bus de salida *OUT_CEDAM*. Los detalles funcionales y de diseño individual de los componentes de la arquitectura del CEDAM se exponen en las siguientes párrafos.

5.2.1.1. Buffer de memoria (MB)

Está compuesto de un arreglo de P memorias RAM de doble puerto, cada una con profundidad de N_P localidades, organizadas como un banco de memorias, tal como el que se muestra en la figura 5.3. En ciertas partes del receptor DDST, las secuencias de datos deben almacenarse conforme se van recibiendo. Este respaldo es necesario para evitar pérdidas de datos en caso de que otros módulos tengan necesidad de modificarlos durante el algoritmo. Por otro lado, tomando en cuenta las características del banco de memoria, se diseñó un novedoso esquema de direccionamiento *hard-wired* para el MB. De forma que, como se indica en la figura 5.4, los $\log_2(N)$ bits correspondientes al bus de direcciones del MB se dividen en dos partes. Los primeros P LSB bits se emplean para seleccionar a una memoria en particular del banco y los bits restantes se destinan para direccionar individualmente cada una de las localidades en la memoria seleccionada. Por lo tanto, el conjunto de N datos almacenados puede ser visualizado como una matriz de tamaño $N_P \times P$, en la cual cada memoria dentro del banco almacena una columna de dicha matriz, tal como lo sugiere la figura 5.3.

5.2.1.2. Estimador de la media cíclica paralelo (PCME)

Este submódulo está diseñado con un enfoque masivamente-paralelo, debido a que está conformado de P bloques que calculan P medias aritméticas en forma simultánea. Cada bloque promediador está conformado de un registro de entrada (*reg_in_pcme*), un sumador complejo, un registro acumulador (*reg_acc_pcme*) y un desplazador, interco-

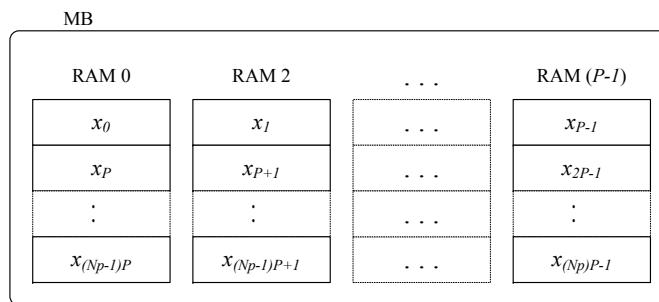


Figura 5.3: Organización de los datos en el *buffer* de memoria.

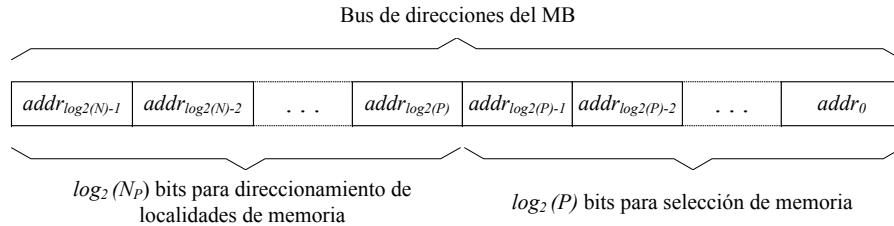


Figura 5.4: Esquema de direccionamiento *hard-wired* para el *buffer* de memoria.

nectados tal como se muestra en la figura 5.2. Una vez que la secuencia de datos se ha reordenado y guardado en forma de matriz en MB, el PCME toma ventaja de su estructura paralela y lee N_P datos de cada una de las memorias para obtener la sumatoria de ellos. Desde el punto de vista del algoritmo, esto equivale a sumar los elementos de la columnas de la matriz. Luego, los resultados de las sumatorias son divididos entre N_P con la ayuda del bloque desplazador. Finalmente, los P resultados correspondientes a $\hat{\mathbf{y}}$ son transferidos a la vez al submódulo SMVM.

5.2.1.3. Multiplicador matriz-vector sistólico (SMVM)

La operación fundamental que debe ejecutar el CEDAM es el producto matriz-vector expresado en (3.7). Desde el punto de vista de consumo de *hardware*, esta parte del diseño es una de las más críticas en la arquitectura del estimador. La estrategia obvia de acelerar su cálculo consiste en ejecutar tantas operaciones en paralelo como sea posible, pero esto consumiría una gran cantidad de recursos del FPGA. Por lo tanto, se decidió optar por un diseño basado en un arreglo sistólico de procesadores, similar al que se presenta en [67, Cap. 3], en aras de obtener un buen rendimiento y evitar un consumo excesivo de *hardware*.

En la esquina inferior izquierda de la figura 5.2 se muestra el elemento procesador (PE), el cual es la operación de PDS atómica dentro del SMVM y está compuesto de un multiplicador complejo y un sumador complejo que procesan tres flujos de datos: los del PCME, los provenientes de la LUT para \mathbf{C}^{-1} y los producidos por el PE adyacente previo. En el diseño del SMVM se consideró que el número de PEs necesarios (tamaño del arreglo sistólico) es de P , lo cual está acorde con las dimensiones de la matriz \mathbf{C}^{-1} y el vector $\hat{\mathbf{y}}$ respectivamente. De forma similar al diseño presentado en [67]; se usó un vector de proyección $\mathbf{d} = [10]^T$ en conjunto con un vector de calendarización $\mathbf{d} = [11]^T$. Con lo que, el periodo del *pipeline* es igual a uno y el tiempo de cálculo para una multiplicación matriz-vector (MVM) completa es de $2P - 1$ ciclos; no obstante, el primer resultado está disponible a partir de $P + 1$ ciclos.

5.2.1.4. LUT para la inversa de C (ICLUT)

Los valores de la matriz circulante \mathbf{C}^{-1} son constantes que pueden precalcularse *offline* y guardarse en forma de LUT en una ROM. De igual importancia es la propiedad de las matrices circulantes que nos señala que únicamente son necesarios los datos de la primera columna para generar dicha matriz, debido a que las columnas restantes son versiones rotadas de la primera. En consecuencia, en un paradigma de diseño tradicional para la LUT; una ROM de tamaño de P localidades y con múltiples puertos de lectura sería suficiente. Sin embargo, dicha memoria sería sintetizada por cualquier herramienta de compilación como un arreglo de P ROMs de puerto sencillo, por lo que el número total de localidades se incrementaría a P^2 . Por tal motivo, se desarrolló una solución ingeniosa con un arreglo de P registros interconectados en configuración de *buffer* circular (figura 5.5). Este diseño aporta un ahorro de $P(P - 1)$ localidades de memoria. Para su funcionamiento, sólo se almacenan los valores de la primera fila de \mathbf{C}^{-1} en los registros. Eventualmente, basta con aplicar una rotación al *buffer* circular en cada ciclo de reloj, para que las salidas del registro cambien su valor, tal como lo indica la temporización mostrada en la figura 5.5.

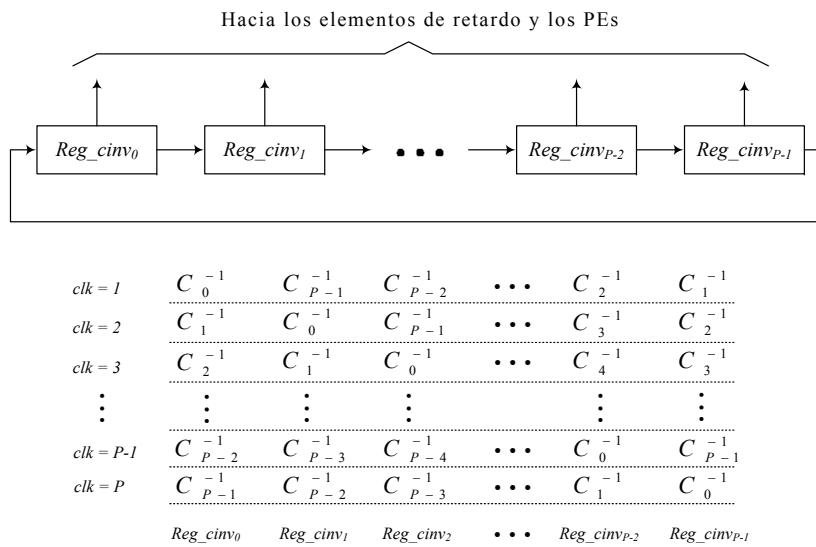


Figura 5.5: Arreglo de registros de la LUT para \mathbf{C}^{-1} y sus correspondientes valores de salida.

5.2.2. Arquitectura digital del estimador CEOF

Una variante del estimador CEDAM se desarrolló al usar un enfoque de diseño distinto, al sustituir el módulo PCME por un estimador de la media cíclica serial (SCME), teniendo como resultado la arquitectura que se observa en la figura 5.6. El módulo SCME no necesita esperar hasta que los $N + P$ datos de entrada sean recibidos y almacenados en MB. Por el contrario, durante $N + P$ ciclos, la media cíclica es calculada al “vuelo”

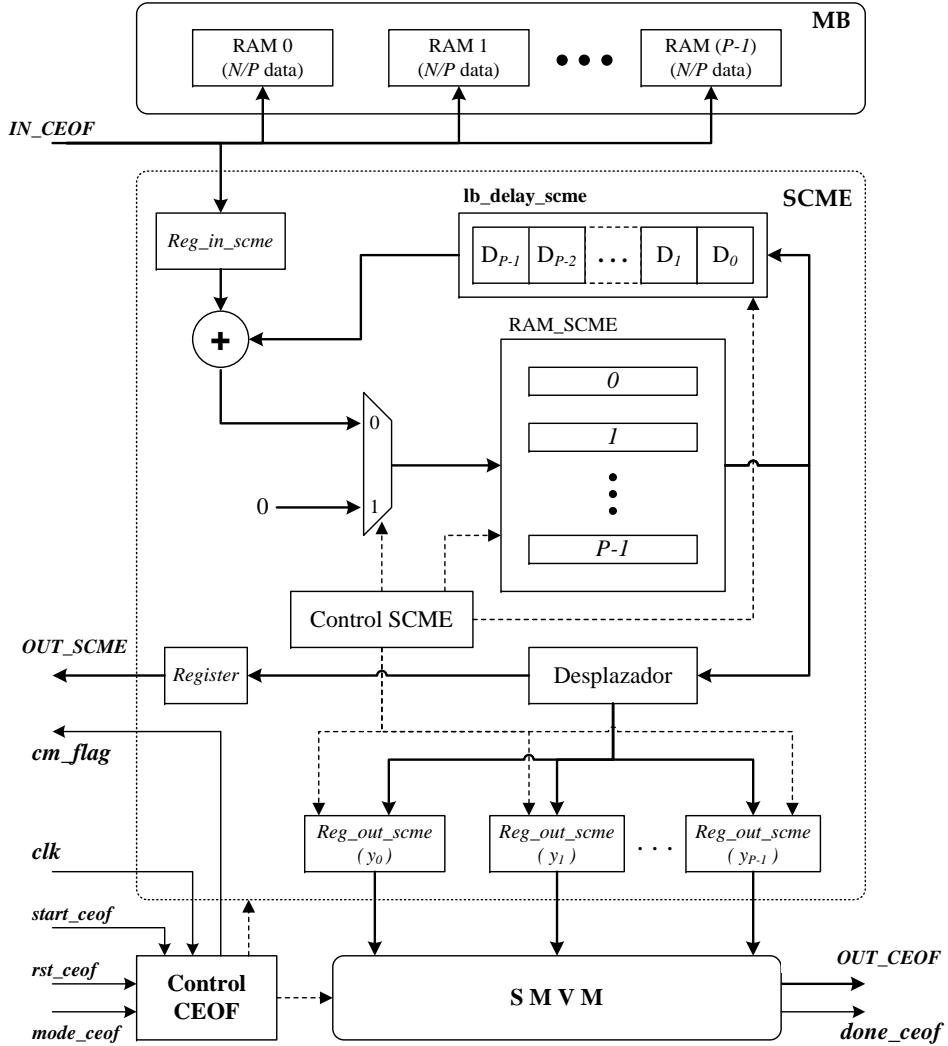


Figura 5.6: Arquitectura de *hardware* del estimador de canal CEOF.

conforme los datos van arribando. Al mismo tiempo, la secuencia de entrada se almacena y queda disponible para otros módulos. Además, el módulo SCME debe ser capaz de identificar los primeros P datos que corresponden al prefijo cíclico y excluirlos al estimar la media cíclica. Una memoria RAM de puerto sencillo (RAM_SCME) de P localidades es la encargada de guardar las P sumatorias y cuando los cálculos finalizan, sus localidades junto con las del registro de desplazamiento (lb_delay_scme) deben ser puestas a cero. El MB no interviene en los cálculos y el módulo SMVM es idéntico al que se usó en la arquitectura CEDAM. Adicionalmente, el estimador CEOF ofrece dos características funcionales útiles:

1. Puede operar en modo de flujo continuo cuando un *buffer* adicional de profundidad $2P$ se interconecta a su bus de entrada IN_CEOF.

2. Puede ser configurado para estimar ya sea los coeficientes de la CIR del canal o sólo la media cíclica, dependiendo del valor presente en su entrada de control *mode_ceof*. En éste último caso, los resultados se envían en forma serial al bus de salida *OUT_SCME* al mismo tiempo que se activa la señal *cm_flag*.

5.2.3. Diseño de un estimador de canal sistólico para DDST

Una característica que distingue al *hardware* de los estimadores descritos en 5.2.1 y 5.2.2, es el hecho de que el cálculo de la MVM se lleva a cabo en un arreglo sistólico de procesadores. Con lo que la idea principal ahora es la de reusar dicho arreglo para estimar la media cíclica de los datos también. Por lo tanto, el algoritmo entero para la estimación del canal se ejecutará únicamente en un arreglo sistólico, similar al descrito en 5.2.1.3, gracias a una sencilla pero novedosa modificación en su diseño.

5.2.3.1. Adecuación del algoritmo de estimación de la media cíclica

El siguiente análisis describe como la media cíclica es obtenida usando un arreglo sistólico de procesadores que calculan una multiplicación matriz–vector. Por principios de cuentas, considérese que la ecuación original (5.1) puede redefinirse con la siguiente expresión matricial, i.e.

$$\mathbf{y} = \mathbf{J}\mathbf{x}, \quad (5.1)$$

donde \mathbf{y} es el vector columna con los valores de la media cíclica de \mathbf{x} y \mathbf{J} viene dado por

$$\mathbf{J} = \frac{1}{N_P} (\mathbf{1}_{N_P}^T \otimes \mathbf{I}_P), \quad (5.2)$$

donde $\mathbf{1}_{N_P}$ es un vector columna de longitud N_P con todos sus elementos igual a uno, \mathbf{I}_P es la matriz identidad de tamaño $P \times P$ y \otimes es el producto Kronecker de dos matrices. Puede notarse que, no obstante (5.1) ya representa una MVM; su implementación resultaría demasiada complicada a causa del producto de Kronecker que tiene involucrado. Con el fin de evitar tal operador, la misma ecuación puede reformularse como

$$\mathbf{y} = \frac{1}{N_P} (\mathbf{\Xi} \mathbf{1}_{N_P}), \quad (5.3)$$

donde $\mathbf{\Xi}$ es una matriz de tamaño $P \times N_P$ la cual esta representada por

$$\mathbf{\Xi} = \begin{bmatrix} [\mathbf{x}]_0 & [\mathbf{x}]_P & [\mathbf{x}]_{2P} & \cdots & [\mathbf{x}]_{(N_P-1)P} \\ [\mathbf{x}]_1 & [\mathbf{x}]_{P+1} & [\mathbf{x}]_{2P+1} & \cdots & [\mathbf{x}]_{(N_P-1)P+1} \\ [\mathbf{x}]_2 & [\mathbf{x}]_{P+2} & [\mathbf{x}]_{2P+2} & \cdots & [\mathbf{x}]_{(N_P-1)P+2} \\ \vdots & \vdots & \vdots & & \vdots \\ [\mathbf{x}]_{P-1} & [\mathbf{x}]_{2P-1} & [\mathbf{x}]_{3P-1} & \cdots & [\mathbf{x}]_{N_PP-1} \end{bmatrix}. \quad (5.4)$$

Sin embargo, una arquitectura sistólica para el cálculo de (5.4) podría ser impráctica bajo la óptica del consumo de *hardware* ya que serían necesarios N_P PEs. Este problema se

conoce como *problem-size dependent array*, donde un algoritmo requiere de un arreglo de procesadores cuyo tamaño depende de la complejidad del problema que requiere ser resuelto.

A pesar de todo, aún es posible mapear el algoritmo de la media cíclica a un arreglo sistólico de procesadores de menor tamaño usando el método de particionamiento [68, Cap. 13]. Considere que la matriz Ξ se partitiona en bloques cuyo tamaño se escogen para concordar con un arreglo de procesadores de tamaño P , luego entonces (5.4) llega a ser

$$\Xi = \left[\mathbf{B}_0 | \mathbf{B}_1 | \cdots | \mathbf{B}_{\frac{N_p}{P}-1} \right], \quad (5.5)$$

donde

$$\mathbf{B}_i = \begin{bmatrix} [\mathbf{x}]_{iP^2} & [\mathbf{x}]_{iP^2+P} & [\mathbf{x}]_{iP^2+2P} & \cdots & [\mathbf{x}]_{iP^2+(P-1)P} \\ [\mathbf{x}]_{iP^2+1} & [\mathbf{x}]_{iP^2+P+1} & [\mathbf{x}]_{iP^2+2P+1} & \cdots & [\mathbf{x}]_{iP^2+(P-1)P+1} \\ [\mathbf{x}]_{iP^2+2} & [\mathbf{x}]_{iP^2+P+2} & [\mathbf{x}]_{iP^2+2P+2} & \cdots & [\mathbf{x}]_{iP^2+(P-1)P+2} \\ \vdots & \vdots & \vdots & & \vdots \\ [\mathbf{x}]_{iP^2+P-1} & [\mathbf{x}]_{iP^2+2P-1} & [\mathbf{x}]_{iP^2+3P-1} & \cdots & [\mathbf{x}]_{iP^2+P^2-1} \end{bmatrix},$$

$$\text{para } i = 0, \dots, \frac{N_p}{P} - 1.$$

De igual forma, $\mathbf{1}_{N_p}$ se partitiona en $\frac{N_p}{P}$ vectores columna unitarios de longitud P . Sustituyendo (5.5) en (5.3), la media cíclica con particionamiento queda concisamente expresada como:

$$\mathbf{y} = \frac{1}{N_p} (\mathbf{B}_0 \mathbf{1}_P + \mathbf{B}_1 \mathbf{1}_P + \cdots + \mathbf{B}_{\frac{N_p}{P}-1} \mathbf{1}_P). \quad (5.6)$$

Por lo tanto, el arreglo de PEs procesará un par de bloques: \mathbf{B} y $\mathbf{1}_P$, uno tras otro en forma secuencial junto con los resultados parciales.

5.2.3.2. Arquitectura de hardware y funcionamiento

En la figura 5.6, se presenta la propuesta de arquitectura denominada como SYSDCE para estimación de canal en DDST. Su operación fundamental es la MVM, la cual se lleva a cabo de forma sistólica dentro de un arreglo de procesadores. Asimismo, en dicha figura también es posible identificar con claridad sus cuatro módulos funcionales: un multiplicador matriz–vector modificado (MSMVM), un alimentador de datos (DATINF), una LUT para la matriz \mathbf{C}^{-1} (ICLUT) y un control. En términos generales, el funcionamiento del estimador SYSDCE se divide en tres fases: almacenamiento de la secuencia de entrada, estimado de la media cíclica y cálculo de canal estimado.

En el instante que se activa la señal de entrada *star_sysdce*, $N + P$ datos, correspondientes a la secuencia de entrada con prefijo cíclico incorporado, son leídos desde el bus *IN_SYSDCE*. Despues de extraer los símbolos del CP, el resto de la secuencia es reordenada y almacenada en el banco de memoria del DATINF. Cuando este proceso finaliza, el control configura al módulo MSMVM y durante N_p ciclos lee P datos a la vez provenientes del DATINF y estima su media cíclica. Hecho esto, el vector $\hat{\mathbf{y}}$ obtenido

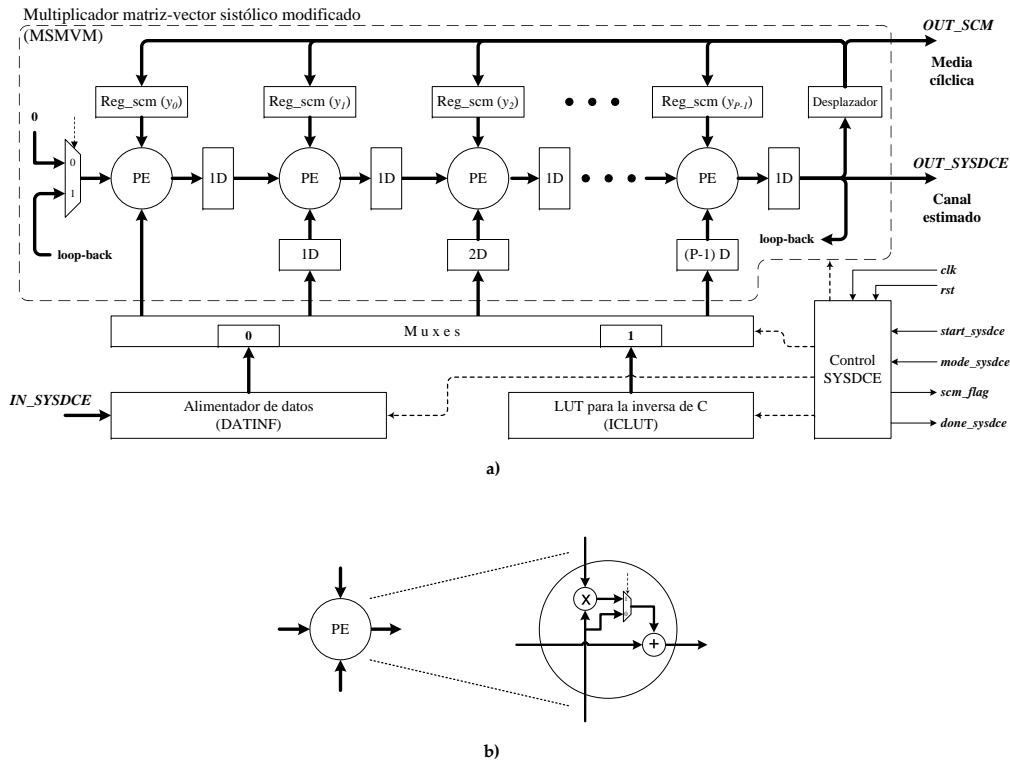


Figura 5.7: Estimador de canal sistólico para un receptor DDST; a) Arquitectura de *hardware*, b) Estructura interna de su PE.

junto con los datos del ICLUT son alimentados nuevamente al módulo MSMVM para ejecutar la estimación del canal (ec. 3.7). Finalmente, transcurridos $P + 1$ ciclos, la salida *done_sysdce* se activa y uno por uno los coeficientes del canal estimado $\hat{\mathbf{h}}$ se envían al bus de salida *OUT_SYSDCE*.

Es preciso señalar que el estimador SYSDCE —al igual que el CEOF— puede configurarse para estimar sólo la media cíclica si la señal de control *mode_sysdce* es puesta a cero. En dicho caso, la salida *scm_flag* se activa para indicar que los resultados válidos están disponibles en el bus *OUT_SCM*. Así, el estimador de canal queda preparado para el procesamiento de otra secuencia de datos.

A continuación se procede en explicar a más detalle cada uno de los módulos de la arquitectura SYSDCE, excepto el de la LUT para \mathbf{C}^{-1} que es exactamente igual al descrito previamente en 5.2.1.4.

5.2.3.3. Multiplicador matriz–vector sistólico modificado (MSMVM)

La arquitectura del MSMVM (figura 5.7a) es una variante del módulo SMVM. Por consiguiente, gran parte de lo que se ha explicado en 5.2.1.3, igualmente aplica para este caso. Dos grandes diferencias radican en la estructura del PE (figura 5.7b) y en la inclu-

sión de una realimentación de datos (*loop-back*). Las razones detrás de la modificación de PE quedan claras al analizar (5.6) donde la expresión que se encuentra entre paréntesis representa una sumatoria de N_P MVMs. Pero, resulta innecesario en cada MVM efectuar cada uno de los productos involucrados, ya que como se asentó en 5.2.3.1, $\mathbf{1}_P$ es un vector con elementos de valor 1. Por tal razón, la estructura original del PE se modificó con un multiplexor. Esto permite que el PE pueda realizar todas las multiplicaciones triviales a través de un *bypass* directo de los datos de entrada del multiplicador complejo hacia el sumador complejo. La función del *loop-back* es realimentar los resultados parciales al arreglo de procesadores.

5.2.3.4. Alimentador de datos (DATINF)

Como regularmente ocurre en la mayoría de los arreglos sistólicos, el MSMSVM requiere que los datos —que se alimentarán a cada uno de sus PEs— estén en un orden preestablecido antes de sean procesados. En la aproximación propuesta, el módulo DATINF es el encargado de realizar esta tarea. El está conformado por un arreglo de P memorias, cada una de las cuales tiene un profundidad de N_P localidades, organizadas como un banco de memorias, tal como se muestra en la figura 5.8. El módulo DATINF lee $N + P$ datos desde el bus *IN_SYSDEC*, identifica y remueve los P datos del prefijo cíclico. Subsecuentemente, reorganiza la secuencia en $\frac{N_P}{P}$ bloques de tamaño $P \times P$ con la finalidad de formar $\mathbf{B}_0, \mathbf{B}_1, \dots, \mathbf{B}_{\frac{N_P}{P}}$. Por lo tanto, los N datos quedan dispuestos de manera semejante a una matriz de $N_P \times P$, donde cada memoria en particular dentro del banco tiene en su haber una columna de cada bloque y estos se encuentran almacenados consecutivamente uno tras otro, tal como se visualiza en la figura 5.8.

Entre tanto, cada dato que se recibe en la entrada del DATINF tiene asociado tres direcciones que definen su posición dentro del banco de memoria y que son: el número de bloque (*blk_num*), el número de memoria (*mem_num*) y la dirección de la localidad dentro de la memoria (*mem_addr*). El módulo DATINF debe generar tales direcciones usando las siguientes expresiones:

$$blk_num = \left\lfloor \frac{k \times N_P}{N \times P} \right\rfloor, \quad k = 0, 1, \dots, N - 1, \quad (5.7a)$$

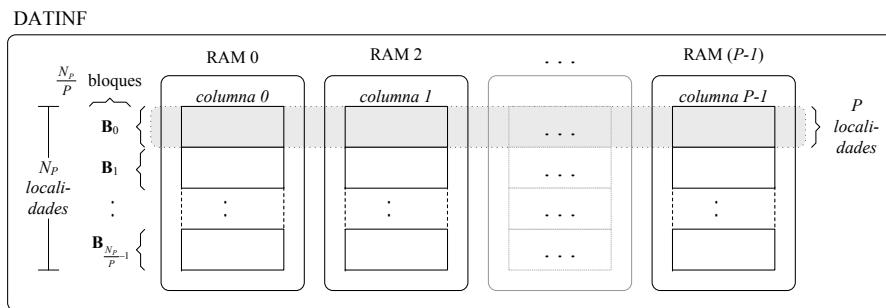


Figura 5.8: Banco de memoria y organización de los datos en el DATINF.

$$mem_num = \left\lfloor \frac{k \times N_p - blk_num \times N \times P}{N} \right\rfloor, \quad (5.7b)$$

$$mem_addr = (k \text{ mód } P) + (P \times blk_num), \quad (5.7c)$$

donde k es el k -ésimo elemento de \mathbf{x} (sin CP) y $\lfloor \cdot \rfloor$ denota el operador *floor*.

Para minimizar el consumo de *hardware*, nuevamente un direccionamiento de tipo *hard-wired* se diseñó para el banco de memorias. Como se detalla en la figura 5.9, los $\log_2 N$ bits del bus de direcciones del DATINF son divididos en tres partes. Los primeros $\log_2 \frac{N}{N_p}$ bits más significativos (MSB) se usan para seleccionar el bloque, los siguientes $\log_2 \frac{N_p}{P}$ bits son para seleccionar una memoria específica en el banco y los $\log_2(P)$ restantes determinan la dirección individual de cada una de las localidades dentro de la memoria seleccionada.

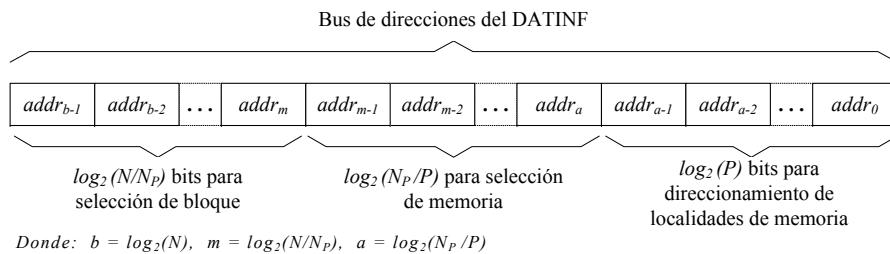


Figura 5.9: Direccionamiento de tipo *hard-wired* para el banco de memoria del alimentador de datos.

5.2.4. Resultados de implementación en FPGA

En esta apartado se expone un análisis comparativo entre los tres estimadores de canal. Todos ellos fueron descritos a nivel en RTL usando Verilog HDL. Sus parámetros de operación se establecieron $N = 512$, con un $CP = P$ y $P = 8$ y se sintetizaron en un FPGA Virtex-5 usando las mismas consideraciones mencionadas en 4.4.

Para definir las longitudes y formatos de palabra convenientes para las arquitecturas de los estimadores, fue necesario en primera instancia caracterizar el comportamiento de la secuencia $x(k)$ que se recibe en el receptor debido a la naturaleza aleatoria de los coeficientes del canal inalámbrico. Por tal motivo, se realizó un análisis estadístico de dicha secuencia usando 312×10^6 datos obtenidos vía 10^6 simulaciones Monte Carlo. Así, fue posible establecer que usando una longitud de palabra (Q_W) de 16 bits —de los cuales 4 corresponden a la parte entera (Q_I) y 12 a la fraccionaria (Q_F)— son suficientes para cuantizar a los datos de la secuencia recibida $x(k)$ y cubrir todo su rango dinámico. En la tabla 5.1 se muestra un resumen detallado de las longitudes y formatos de palabra empleados en cada uno de los estimadores de canal implementados. Además, como se puede visualizar en dicha tabla, ninguna de las operaciones matemáticas ejecutadas

	Módulo aritmético	CEDAM			CEO			SYSDE		
		Q _W	Q _I	Q _F	Q _W	Q _I	Q _F	Q _W	Q _I	Q _F
PCME / SCME	Entradas del sumador complejo	16	4	12	16	4	12	-	-	-
	Salidas del sumador complejo	22	10	12	22	10	12	-	-	-
	Entradas del desplazador	22	10	12	22	10	14	24	9	15
	Salidas del desplazador	16	2	14	16	2	14	16	2	14
SMVM / MSMVM	Entradas del multiplicador complejo	16	2 / 1	14 / 15	16	2 / 1	14 / 15	16 / 19	2 / 4	14 / 15
	Salidas del multiplicador complejo	18	3	15	18	3	15	18	3	15
	Entradas del sumador complejo	18	3	15	18	3	15	23	8	15
	Salidas del sumador complejo	18	3	15	18	3	15	23	8	15
IC-LUT	Salidas del ICLUT	16	1	15	16	1	15	16	1	15

Tabla 5.1: Longitudes y formatos de palabra usados en las arquitecturas CEDAM, CEOF y SYSDCE.

en las arquitecturas emplean precisión completa con la finalidad de evitar un consumo desmedido de recursos del FPGA.

La tabla 5.2 resume los resultados de la síntesis de los tres estimadores de canal. Los valores en los paréntesis al lado de cada *item* indican el total disponible del recurso en el FPGA. En lo que respecta a la frecuencia de operación, se nota que la diferencia entre los estimadores CEDAM y CEOF es marginal a pesar de que el primero es masivamente-paralelo. En cambio, para el SYSDCE la frecuencia disminuye hasta los 115 MHz debido a un incremento en la ruta crítica por parte del alimentador de datos. Un común denominador en los tres radica en su bajo consumo de registros y LUTs. La diferencia tan marcada en el renglón de consumo de BRAMs tiene su razón en el hecho de que tanto el alimentador de datos y el MB se consideran parte del estimador en las arquitecturas CEDAM y SYSCE, caso contrario del CEOF donde no se depende del MB. También, todos ellos consumen la misma cantidad de módulos DSP48Es.

Ahora, es importante determinar un aproximado de los ciclos necesarios en cada una de las arquitecturas de los estimadores para obtener tanto la media cíclica como los coeficientes de la CIR. En el caso de la arquitectura CEDAM, el número de ciclos necesarios para obtener el canal estimado esta dado por

$$cedam_ciclos_{\hat{h}} = (N + P) + (N_P + 2P - 1). \quad (5.8)$$

El primer termino en (5.8) corresponde a los ciclos que se consumen por el almacenamiento previo de los datos en el MB y se deben incluirse puesto que el CEDAM sólo funciona bajo esa condición. En cambio, el estimador CEOF para estimar la media cíclica en particular requiere

$$ceof_ciclos_{\hat{y}} = N + P, \quad (5.9)$$

es decir, los resultados son obtenidos en instante justo que se recibe el último dato de la secuencia de entrada. Este mismo estimador, para la tarea de estimación de canal consume

$$ceof_ciclos_{\hat{h}} = ceof_ciclos_{\hat{y}} + 2P - 1. \quad (5.10)$$

En ese mismo tenor, para estimar la media cíclica en la arquitectura SYSDCE son necesarios:

$$sysdce_ciclos_{\hat{y}} = (N + P) + (N_P + P - 1). \quad (5.11)$$

El primer término en la ec. (5.11) corresponde a la fase de almacenamiento y el segundo a las $\frac{N}{P}$ MVM operaciones involucradas. Luego entonces, la cantidad de ciclos para la estimación final está dada por

$$sysdce_ciclos_{\hat{h}} = sysdce_ciclos_{\hat{y}} + (2P - 1). \quad (5.12)$$

Por otro lado, vale la pena mencionar que resulta difícil comparar los estimadores diseñados con los reportados en otros trabajos previos [42, 43], debido a diferencias en los paradigmas, tecnología, condiciones de verificación, etc. La comparación contra el estimador de [42], no fue posible por ser un sistema *full-software*. Algo similar ocurre con el diseño híbrido reportado en [43], donde del módulo de estimación, únicamente la media (aritmética en este caso) fue trasladada a un coprocesador. Los reordenamientos en los datos y la operación MVM descrita en (3.7) fueron implementados en software. Con lo que, la comparación solo se hará entre dicho coprocesador y los estimadores propuestos.

De igual importancia, para evaluar y comparar el desempeño de los estimadores, se definen las siguientes métricas. El tiempo de procesamiento (PT), es el tiempo que transcurre desde el inicio de un estimado de la media cíclica o del canal hasta que se finaliza

Estimador de canal	CEDAM	CEOFS	SYSDE
Secuencia de entrada (sin CP)	(N)	512	512
Frecuencia de operación	(MHz)	161.520	161.520
Registros (<i>Slice registers</i>)	(69120)	2371 (3%)	917 (1%)
LUTs (<i>Slice LUTs</i>)	(69120)	929 (1%)	937 (1%)
BRAMs (<i>Block RAMs</i>)	(148)	8 (5%)	1 (<1%)
Módulos DSP48Es	(64)	32 (50%)	8 (5%)

Tabla 5.2: Resultados de síntesis para los estimadores de canal propuestos ($N = 512$, $P = 8$ y $CP = P$).

Estimador de canal	Modo de operación	Secuencia de entrada	Ciclos/ media cíclica	Ciclos/ estimación de canal	CMT (us)	CET (us)	TP (MS/s)	TP/area (MS/s/slices)
CEDAM	—	512	—	599	—	3.708	138.06	41.83e3
CEOFL	Media cíclica	512+8 (CP)	520	—	3.246	—	157.73	73.22e3
CEOFR	Estimación de canal	512+8 (CP)	—	537	—	3.352	152.72	70.90e3
SYSDEC	Media cíclica	512+8 (CP)	591	—	5.128	—	101.06	25.625e3
SYSDFR	Estimación de canal	512+8 (CP)	—	606	—	5.258	98.91	24.996e3
Promediador [30]	Coprocésador	512+8 (CP)	2238	—	20	—	26.29	—

Tabla 5.3: Tabla comparativa del rendimiento en los estimadores propuestos.

su cálculo. El rendimiento (*throughput*-TP) por área permite en una arquitectura relacionar a través de un cociente, el comportamiento combinado de rendimiento-consumo, un valor más alto es indicativo de una mejor implementación bajo esos términos.

Analizando la tabla 5.3, se advierte que aunque el CEOF tiene el mejor desempeño en casi todas las métricas comparadas, no existe una diferencia drástica con respecto al CEDAM. El estimador SYSDEC es el que menor desempeño tiene de los tres, sin embargo todas ellas son mucho mejores con respecto al coprocesador para el cálculo de la media aritmética usado en [43].

5.2.5. Resultados de implementación en ASIC

Los códigos RTL usados para validar las implementaciones en FPGA de los estimadores de canal CEDAM y CEOF fueron sintetizados (con un mínimo de modificaciones), enrutados (*place&routed* - P&R) e implementados en tecnología de 90 nm de tipo *generic CMOS standard cell* de nueve capas (Synopsys SAED 90 nm). La figura 5.10 ilustra los esquemas de diseño (*layouts*) de las arquitecturas mencionadas y la tabla 5.4 resume su comparación en términos de:

- Área de silicio usada.
- Complejidad representada en número de compuertas *nand2* equivalentes (asumiendo que una compuerta *nand2* ocupa $5.8 \mu\text{m}^2$).
- Frecuencia máxima de operación sostenible después de P&R.
- Potencia dinámica reportada por la herramienta de análisis de tiempo estático (asumiendo una tasa de commutación de 10 % en las compuertas).

Puede notarse que los resultados presentados en la tabla 5.4 son representativos de un proceso de síntesis y P&R típico con las opciones de ejecución por defecto en la herramienta de diseño. No se activó ninguna opción para incrementar el nivel de esfuerzo en dichos procesos que permitieran mejorar los parámetros mostrados en la tabla 5.4. En una comparación justa, también puede observarse que la arquitectura CEOF es 34 % más

		CEDAM	CEOOF	$\Delta\%$
Área	(μm^2)	823,096	543,544	-33.96%
Complejidad	(Compuertas)	141,913	93,715	-33.96%
Frecuencia máxima	(MHz)	187	245	31.02%
Potencia dinámica	(mW)	3.70	2.74	-25.83%
Condiciones de operación: 1.2V, 25 °C. Modelo de interconexión: <i>balance tree</i>				

Tabla 5.4: Comparación de las implementaciones VLSI de los estimadores CEDAM y CEOOF.

pequeña en área comparada con el estimador CEDAM. Asimismo, es 31 % más rápida y consume 25 % menos potencia. Por lo que, la arquitectura CEOOF es más eficiente bajo estos términos.

Con el objetivo de hacer un uso más eficiente del área, los ocho bancos de RAM estática de 64 localidades por 32 bits (fácilmente identificables en la figura 5.10) de la arquitectura CEDAM, fueron implementados como *hard macros cells* usando las librerías de la tecnología de 90 nm. Con lo que, de los $823,096 \mu\text{m}^2$ del área total usada, la memoria consume alrededor de $334,518 \mu\text{m}^2$, lo cual representa el 40.64 %. A diferencia de lo que sucede en los FPGA's, en este caso no aplica hacer la comparación únicamente con la lógica aleatoria de las arquitecturas, debido a que las memorias consumen grandes porciones de área. Por otro lado, debido a que su tamaño es pequeño (8 localidades de 32 bits) en la arquitectura CEOOF la memoria se sintetizó usando *Flip-Flops*.

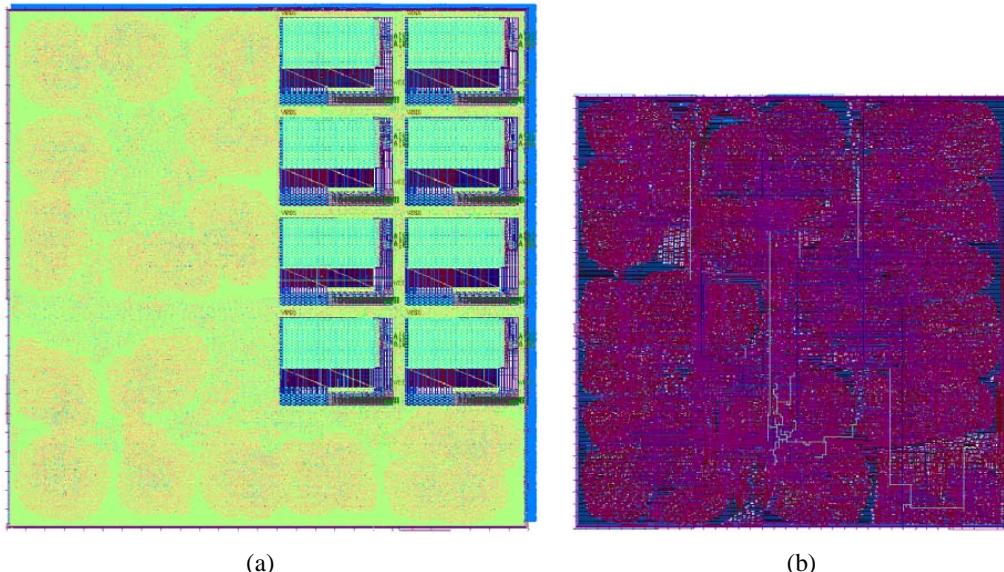


Figura 5.10: Esquemas de diseño VLSI de los estimadores de canal propuestos; a) CEDAM, b) CEOOF.

Cabe señalar que no es posible establecer una comparación justa entre las implementaciones en ASIC y FPGA a causa de la diferencia de tecnologías (90 nm vs 65 nm). Lo anterior aunado al uso de componentes tales como los módulos DSP48 y BRAMs que están diseñados *ad-hoc* para aumentar el rendimiento y eficiencia en el FPGA [69].

5.2.6. Resultados de simulación

La validación funcional de las arquitecturas se concede comparando sus resultados con el modelo de oro en punto flotante programado en Matlab. Así, se consideró el siguiente escenario:

- (i) La longitud de la secuencia de datos de entrada es de $N = 512$ símbolos 4-QAM.
- (ii) La longitud del prefijo cíclico es de $CP = 8$.
- (iii) $c(k)$ es una secuencia OCI de periodo $P = 8$ y $\sigma_c^2 = 0.2$.
- (iv) El canal es Rayleigh, selectivo en frecuencia y de tipo *block-flat-fading*.
- (v) La longitud del canal es $M = 8$ y se genera al azar en cada realización, donde la parte real e imaginaria de $h(k)$ son variables aleatorias independientes e idénticamente distribuidas con función de densidad de probabilidad (PDF) Gaussiana de media cero. Su energía se normaliza a uno.
- (vi) A cada realización del canal se le añade ruido blanco Gaussiano con media cero.

Por otro lado, en las tres arquitecturas propuestas se utilizó un esquema de verificación semejante al que se muestra en la figura 5.11, con el objetivo de corroborar su correcto funcionamiento y evaluar métricas como el error cuadrático medio (MSE) del canal estimado y el desempeño en punto fijo en términos de SQNR. Como se puede observar, en la verificación no se utilizó el *hardware* del transmisor ST/DDST, a pesar de contar ya con él. En caso contrario, la medición del SQNR en los estimadores se hubiera

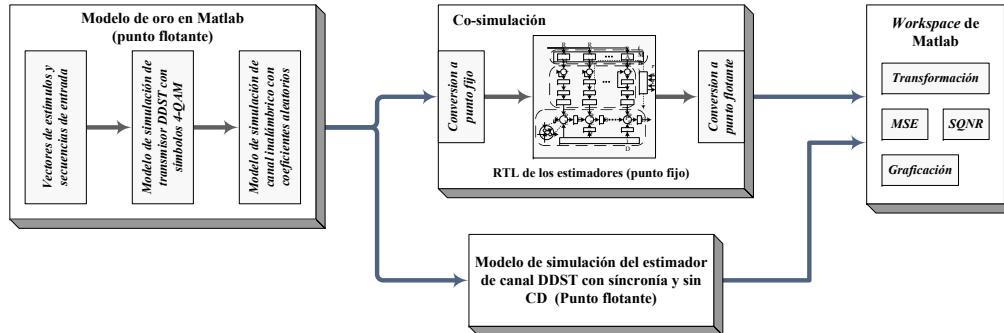


Figura 5.11: Diagrama a bloques del esquema de verificación para las arquitecturas de los estimadores CEDAM, CEOF y SYSDCE.

visto afectada por el error de cuantización que introduciría el transmisor y por ende, ya no correspondería sólo a ellos. Adicionalmente, al ser la secuencia de entrada una variable aleatoria, se requiere hacer una cantidad significativa de simulaciones Monte Carlo para que las mediciones del MSE y SQNR sean confiables y representativas.

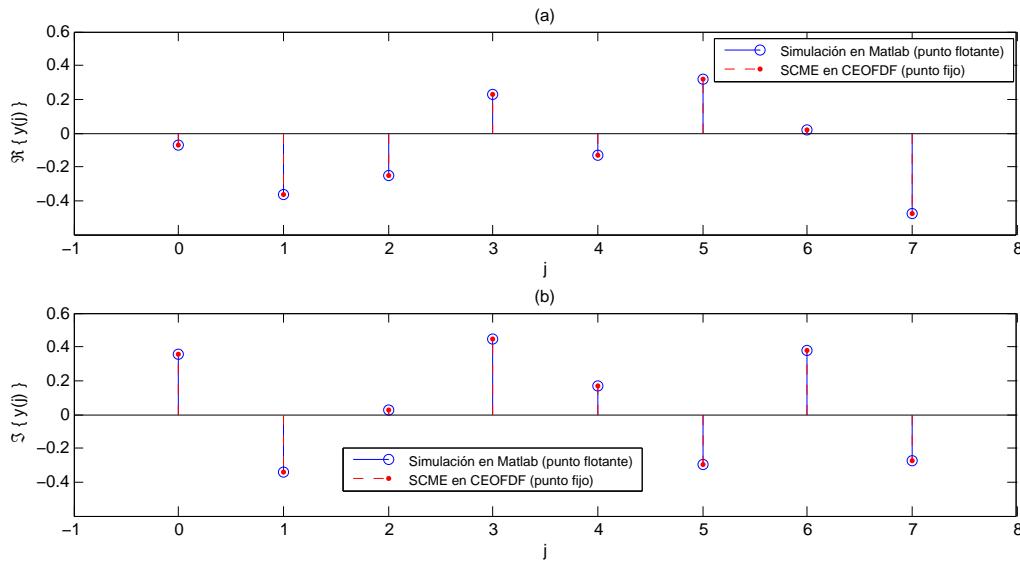


Figura 5.12: Estimación de la media cíclica realizada por el módulo SCME del estimador CEOF *versus* modelo de simulación en Matlab; a) Parte real, b) Parte imaginaria.

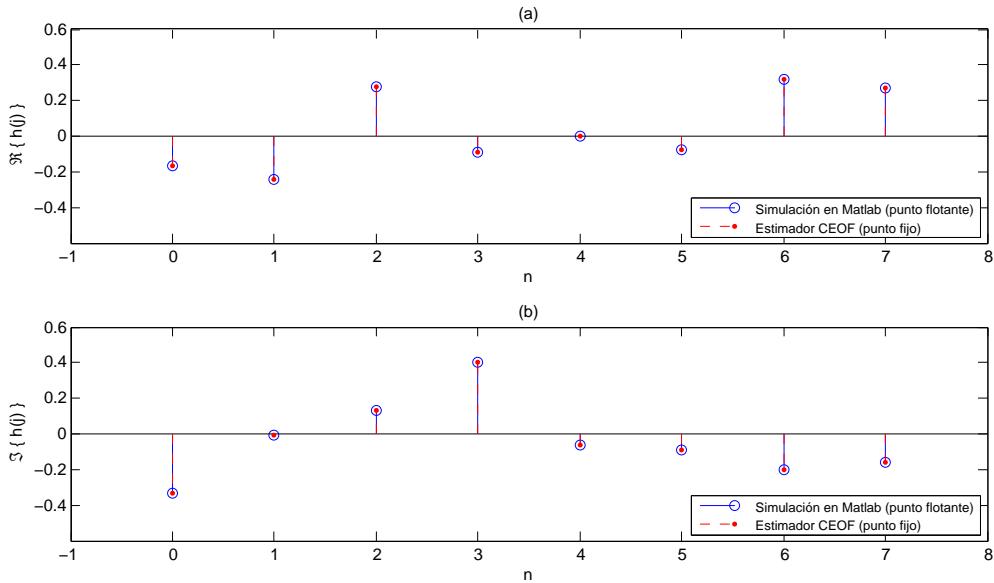


Figura 5.13: Estimación del canal realizada por el estimador CEOF *versus* modelo de simulación en Matlab; a) Parte real, b) Parte imaginaria.

Las gráficas de la figura 5.12 muestran \hat{y} obtenida con el módulo SCME del estimador CEOF y con el modelo de Matlab. Se observa que la arquitectura es capaz de estimar correctamente los valores de la media cíclica, a tal punto que su traza se superpone a la obtenida con el modelo en punto flotante. También es evidente al observar la figura 5.13, que los coeficientes estimados de la respuesta al impulso del canal obtenidos con la arquitectura del CEOF tienen un comportamiento similar cuando se grafican y se comparan con su contraparte resultante de la simulación en Matlab.

La figura 5.14 muestra el MSE del canal estimado, el cual es promediado sobre 300 simulaciones Monte Carlo para cada valor de SNR. Notese que el MSE de las tres arquitecturas de estimadores de canal es muy cercano entre sí y casi indistinguible con respecto a la traza que representa el MSE obtenido con modelo de oro de simulación en punto flotante.

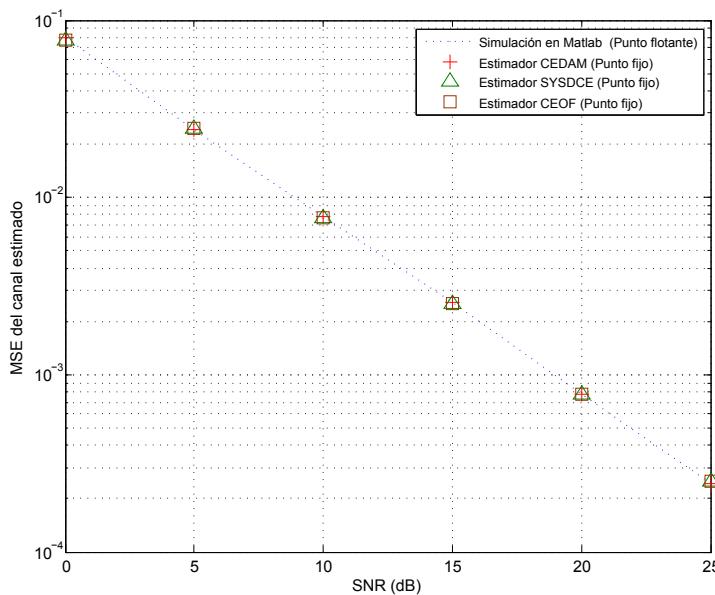


Figura 5.14: Rendimiento del MSE de las arquitecturas *versus* el obtenido con el modelo de simulación en Matlab.

Para cuantificar la diferencia entre los valores tanto de la media cíclica estimada como el de los coeficientes del canal estimado, obtenidos con el modelo de simulación en punto flotante y el resultante en cada una de las tres arquitecturas desarrolladas; se promedió en ellas el SQNR individual obtenido en cada una de las 300 simulaciones Monte Carlo para cada valor de SNR del sistema de comunicación.

Se puede observar en la figura 5.15, que los histogramas* para las arquitecturas CEDAM y CEOF son idénticos debido a que se usan las mismas longitudes y formatos de palabra. Su rango de SQNR para la media cíclica y estimación de canal es de 12 dB y su valores medios toman lugar alrededor de 80 dB y 65 dB, respectivamente. En lo que

*Se uso la regla de Sturges [70] para calcular el número de clases en los histogramas de este apartado

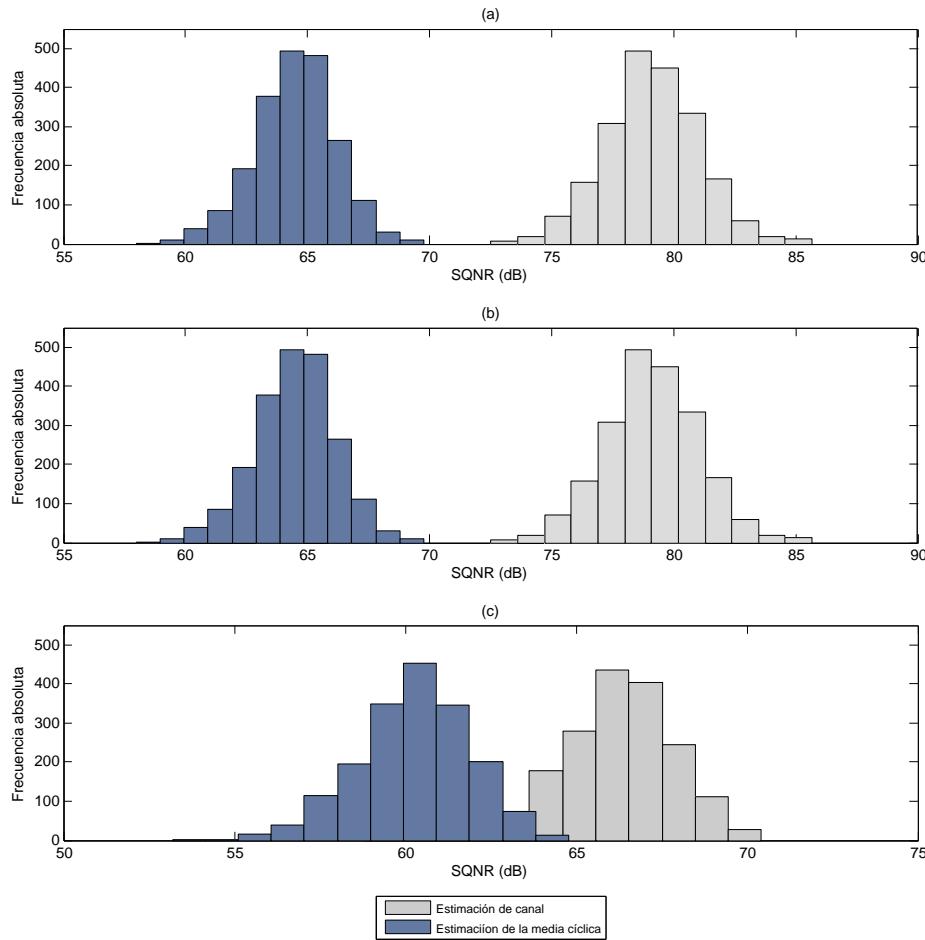


Figura 5.15: Histogramas del rendimiento SQNR para 300 simulaciones Monte Carlo de las arquitecturas de estimadores de canal DDST; a) CEDAM, b) CEOF, c) SYSDCE.

respecta a la arquitectura SYSDCE, el rango sigue siendo el mismo pero el rendimiento del SQNR es en promedio menor en casi 12 dB en la media cíclica y 4 dB para la estimación de canal. No obstante, todos ellos sobrepasan el umbral máximo del rango de operación (0-30 dB) del sistema.

5.3. Arquitectura digital de un estimador de canal para DDST con corrección de sincronía y estimación del desplazamiento de CD.

En este apartado se presenta la forma en la que el estimador CEOF —el cual funciona bajo la presunción de sincronía perfecta y ausencia de desplazamiento de CD— puede usarse como componente clave en el diseño de un estimador *full-hardware* que sea capaz de estimar el canal inalámbrico usando DDST bajo condiciones de falta de

sincronía entre el receptor-transmisor y considerando un desplazamiento de CD desconocido. Esta nueva propuesta de arquitectura, llamada en lo subsecuente “Extended CEOF” (ECEO) tiene su fundamento en los algoritmos descritos en los apartados 3.2.6 y 3.2.7. Los pormenores de su diseño se presentan a continuación.

5.3.1. Adecuaciones del algoritmo

La principal ventaja del algoritmo usado en el estimador ECEO es que realiza conjuntamente la corrección de sincronía y la estimación del desplazamiento de CD. De forma general, hay cuatro tareas que el algoritmo debe ejecutar: TSS, BS, estimación del desplazamiento de CD y estimación verdadera del canal. Todas ellas son procedimientos altamente iterativos y secuenciales. Esto significa que ningún proceso nuevo puede iniciarse si el actual no ha finalizado, dicha característica ocasiona cuellos de botella debido a la gran dependencia entre procesos. En consecuencia, se deben realizar adecuaciones a las ecuaciones originales del algoritmo con el objetivo de hacer eficiente —en términos de rendimiento y consumo de *hardware*— su mapeo a una arquitectura digital.

5.3.1.1. Adecuación de las ecuaciones para TSS

Como se estableció en la sección 3.2.6, el primer estimado de la CIR es sólo una versión con desplazamiento circular del estimado real del canal. Por consiguiente, una de las posibles permutaciones corresponde a la estimación del canal bajo TSS, así el problema se reduce a encontrar la permutación correcta. En (3.29), la permutación originalmente está definida en la matriz \mathbf{C}^{-1} antes de su multiplicación por la media cíclica, lo cual requiere la ejecución de P productos matriz-vector. Sin embargo, ya que \mathbf{C}^{-1} es una matriz circulante, fue posible evitar $P - 1$ multiplicaciones matriz-vector al usar la siguiente propiedad:

$$\text{circshift}_\tau(\mathbf{C}^{-1})\mathbf{y} = \text{circshift}_\tau(\mathbf{C}^{-1}\mathbf{y}). \quad (5.13)$$

Por otra parte, se sabe que la norma de un vector $\mathbf{v} \in \mathbb{C}$ de longitud N se define por

$$\|\mathbf{v}\| = \sqrt{\sum_{k=0}^{N-1} (Re([\mathbf{v}]_k)^2 + Im([\mathbf{v}]_k)^2)}. \quad (5.14)$$

Analizando (3.29), puede notarse que su argumento queda en términos de una substracción y por ende puede ser reescrita como:

$$\hat{\tau} = \arg \min_{\tau} \left\| \tilde{\mathbf{h}}_\tau - \bar{\tilde{\mathbf{h}}}_\tau \right\|, \quad (5.15)$$

donde $\tilde{\mathbf{h}}_\tau = (\text{circshift}_\tau(\mathbf{C}^{-1})\hat{\mathbf{y}})_{[P-M]}$, $\bar{\tilde{\mathbf{h}}}_\tau = [\bar{\tilde{h}}_\tau, \tilde{h}_\tau, \dots, \tilde{h}_\tau]^T$ y siendo $\bar{\tilde{h}}_\tau$ el valor medio de los elementos de $\tilde{\mathbf{h}}_\tau$. Esto implica que para la ejecución de la substracción compleja $(\tilde{\mathbf{h}}_\tau - \bar{\tilde{\mathbf{h}}}_\tau)$ están involucradas dos lecturas del mismo vector, ya que es necesario esperar

a que primero se obtenga $\tilde{\mathbf{h}}_\tau$. Por lo tanto, para evitar lo anterior y calcular la norma en un solo paso, se modificó y aplicó la factorización propuesta en [43]

$$\| \mathbf{h} - \tilde{\mathbf{h}} \| = \sqrt{A - 2B + C}, \quad (5.16)$$

donde

$$A = \sum_{k=M}^{P-1} (Re([\mathbf{h}]_k)^2 + Im([\mathbf{h}]_k)^2) \quad (5.17a)$$

$$B = Re(\bar{h}) \sum_{k=M}^{P-1} Re([\mathbf{h}]_k) + Im(\bar{h}) \sum_{k=M}^{P-1} Im([\mathbf{h}]_k) \quad (5.17b)$$

$$C = (P - M) (Re(\bar{h})^2 + Im(\bar{h})^2). \quad (5.17c)$$

Pero es evidente que:

$$Re(\bar{h}) = \frac{\sum_{k=M}^{P-1} Re([\mathbf{h}]_k)}{P - M} \quad \text{e} \quad Im(\bar{h}) = \frac{\sum_{k=M}^{P-1} Im([\mathbf{h}]_k)}{P - M}, \quad (5.18)$$

con lo que sustituyendo (5.18) en (5.17c) y después de algunas manipulaciones algebraicas se tiene que $C = B$. De este modo, (5.16) queda definida en forma concisa por

$$\| \mathbf{h} - \tilde{\mathbf{h}} \| = \sqrt{A - B}. \quad (5.19)$$

Mas aún, debido a que $\| \mathbf{h} - \tilde{\mathbf{h}} \| \geq 0$ y tomando ventaja de que la raíz cuadrada de un número positivo es una función estrictamente monótonica ascendente, entonces $\sqrt{a} < \sqrt{b}$ puede sustituirse por $a < b$. Por lo tanto, en cada una de las iteraciones puede omitirse el cálculo de la raíz cuadrada sin afectar el resultado de la comparación.

5.3.1.2. Adecuación de las ecuaciones para BS

Del mismo modo que para TSS, el funcional que se debe evaluar para corrección de BS puede reformularse como:

$$\hat{\tau}_P = \arg \min_{\tau_P} \left\| \tilde{\mathbf{h}}_{\tau_P} - \tilde{\mathbf{h}}_{\tau_P} \right\|, \quad (5.20)$$

donde $\tilde{\mathbf{h}}_{\tau_P} = (\mathbf{C}^{-1} \hat{\mathbf{y}}(\tau_p))_{[P-M]}$, $\tilde{\mathbf{h}}_{\tau_P} = [\tilde{h}_{\tau_P}, \tilde{h}_{\tau_P}, \dots, \tilde{h}_{\tau_P}]^T$ y siendo \tilde{h}_{τ_P} el valor medio de los elementos de $\tilde{\mathbf{h}}_{\tau_P}$. Luego entonces, dicho algoritmo involucra las mismas operaciones que el de TSS salvo las siguientes excepciones:

- El número de iteraciones es N_P en lugar de P .

- Al contrario de lo que sucede en TSS, en BS es absolutamente necesario estimar la media cíclica $\tilde{y}(\tau_p)$ en cada iteración, por lo que ya no es posible aplicar (5.13). En consecuencia, resulta inevitable realizar N_p estimaciones del canal —que conllevan igual número de multiplicaciones matriz-vector— para encontrar el vector de media cíclica que minimiza (3.32).

Es importante resaltar, que se usa nuevamente (5.19) para calcular la norma en (3.32).

5.3.1.3. Adecuación de las ecuaciones para estimación del desplazamiento de CD

Analizando la ec. (5.17c), es posible apreciar que otra de las ventajas de la factorización aplicada para el cálculo de la norma, consiste en que el cálculo de B lleva inmerso el estimado del desplazamiento de CD definido en (3.36). Por lo que el mismo flujo de datos se aprovecha para hacer ambos cálculos en forma simultánea.

5.3.2. Descripción de la arquitectura del estimador de canal ECEOOF

La figura 5.16 muestra la arquitectura simplificada final para este estimador. En ella se pueden visualizar varios módulos que pueden agruparse acorde a su funcionalidad dentro de cuatro dominios que son:

1. Recepción de datos.
2. Estimación de media cíclica y canal.
3. Estimación de retardo de sincronía y desplazamiento de CD.
4. Estimación verdadera de canal.

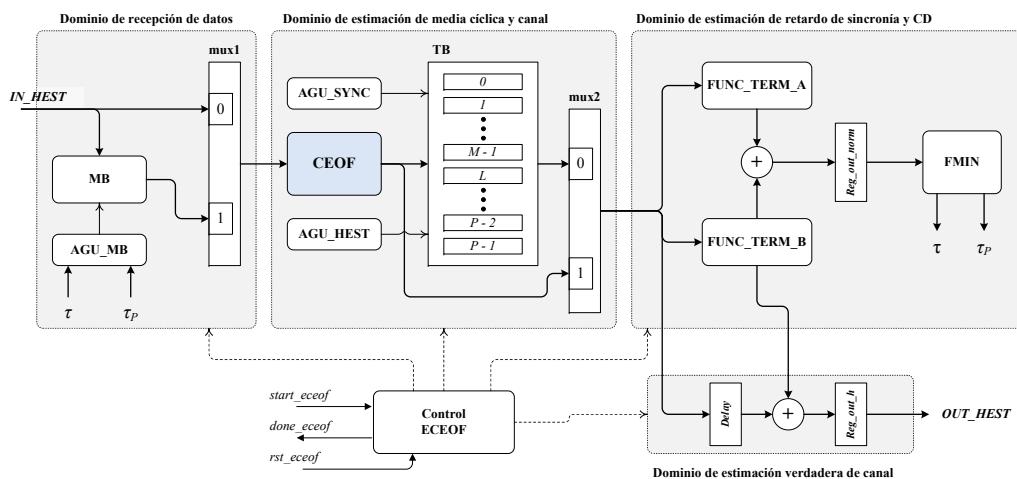


Figura 5.16: Arquitectura simplificada de *hardware* del estimador de canal ECEOOF.

El flujo de datos a través de estos dominios durante la ejecución del algoritmo se detalla a continuación. Tan pronto se active la señal de entrada *start_ceeof*, la secuencia de entrada de longitud $2(N + P)$ se proporciona al estimador por medio de su bus de entrada *IN_HEST* y se almacena en MB. Simultáneamente, las primeras N muestras de dicha secuencia se alimentan al módulo CEOF —a través del bus de entrada 0 del MUX1— para realizar la primera estimación del canal.

Una vez finalizada dicha tarea, los P coeficientes estimados se almacenan en la memoria TB (*temporal buffer*) y el módulo AGU_SYNC inicializa el apuntador de memoria *mptr_sync_addr* con $P - M - 1$. Con la finalidad de encontrar τ , el siguiente proceso iterativo toma lugar entre los dominios 2 y 3.

- (a) Las últimas $P - M$ localidades de la memoria TB son leídas por el dominio 2 para la evaluación del funcional TSS.
- (b) En el módulo FMIN, se comparan el resultado previo del funcional y el actual. Sólo el menor de los dos se elige y almacena.
- (c) En el módulo AGU_SYNC se incrementa en uno el valor del apuntador *mptr_sync_addr*.

Los pasos anteriores se ejecutan P veces. El número de iteración donde se calcula el valor mínimo para (3.29) corresponde a τ . Luego, para propósitos de corrección de TSS, el módulo AGU_MB emplea el valor de τ para incorporar un desplazamiento en las direcciones que genera. Con lo que, el apuntador de memoria *mptr_mb_addr* se actualiza con $P + \tau$. Siguiente, un proceso iterativo nuevo inicia con los dominios 1-3 para BS.

- (a) El módulo CEOF lee un bloque de N datos a partir de la localidad *mptr_mb_addr* – 1 hasta la *mptr_mb_addr* + $N - 1$ del MB, para estimar tanto \mathbf{y}_{τ_P} como $\tilde{\mathbf{h}}_{\tau_P}$.
- (b) Los últimos $P - M$ elementos del canal estimado son transferidos directamente al dominio 3 para el cálculo del funcional de BS.
- (c) En el módulo FMIN, se comparan el resultado previo del funcional y el actual. Sólo el menor de los dos se elige y almacena.
- (d) En cada iteración, se incrementa en P el valor del apuntador *mptr_sync_addr* en el módulo AGU_SYNC.

Estos pasos se ejecutan N_P veces. El número de iteración donde se genera el valor mínimo para (3.32) corresponde a τ_P . Se obtiene entonces el desplazamiento final para el direccionamiento con $\tau_s = \tau + P\tau_P$ y con este valor se actualiza el apuntador *mptr_mb_addr*. Una nueva estimación de la media cíclica y del canal —ya con la sincronía corregida pero aún con desplazamiento de CD— inicia con la lectura de un bloque de N datos del MB ($[\mathbf{x}]_{mptr_mb_addr+P-1}$ a $[\mathbf{x}]_{mptr_mb_addr+N+P-1}$) por parte del estimador CEOF. Los primeros M coeficientes del canal estimado son enviados al registro *delay* para retrasar su flujo, mientras que los últimos $P - M$ coeficientes son transferidos, vía

entrada 0 del MUX2, al módulo *func_term_B* para estimar el desplazamiento de CD. Finalmente, la estimación verdadera del canal se obtiene cuando el valor estimado del desplazamiento de CD es sustraído de cada uno de los datos provenientes del registro *delay*. Los coeficientes de la CIR estimada son enviados al bus de salida *OUT_HEST* y se activa la señal de salida *done_eceof*.

5.3.3. Reutilización del módulo MB y el estimador CEOF

Observando detenidamente la figura 5.16, se pueden identificar tanto al módulo MB como el estimador CEOF, los cuales han sido previamente descritos en el apartado 5.2. Estos han sido reutilizados para el diseño del estimador ECEOFO con las (pequeñas) modificaciones que ha continuación se exponen.

El módulo MB

Su función es almacenar el bloque de $2(N + P)$ datos de entrada. A partir de ellos y con la ayuda del AGU_SYNC direccionar los bloques de N datos que serán leídos por el módulo CEOF. A diferencia de su antecesor (sección 5.2.1.1), ahora se diseñó para operar bajo un esquema de almacenamiento y lectura lineal, aprovechando la virtud del módulo CEOF de procesar flujos de datos *on-the-fly* sin necesidad de ningún tipo de reordenamiento previo.

Estimador CEOF

Es el módulo angular sobre el cual radica el diseño de la arquitectura del estimador ECEOFO. Su función es realizar las $N_P + 2$ estimaciones necesarias tanto de la media cíclica como del canal. Su diseño tuvo que modificarse para procesar secuencias de entradas de longitud N en vez de las $N + P$ (datos más prefijo cíclico) muestras con los que originalmente operaba (sección 5.2.2). Es decir, se eliminó la fase de detección y supresión del prefijo cíclico.

5.3.4. Generador de direcciones de lectura para el MB (AGU_MB)

Las fases de estimación/corrección del retardo de sincronía de trama y la de estimación verdadera del canal requieren del uso intensivo de las $2(N + P)$ muestras de entrada previamente almacenadas en el MB. Para esto, la unidad de control global necesita habilitar al AGU_MB para que genere las direcciones necesarias para la lectura de $N_P + 1$ bloques de N datos del MB. Por lo que, tal como lo sugiere la figura 5.17, el AGU_MB calcula los distintos valores del apuntador de memoria *mptr_mb_addr* combinando las señales de entrada *in_tau* e *in_taus*. A partir de ahí, con la ayuda de un contador binario ascendente módulo N , genera las direcciones válidas para extraer cada uno de los bloques datos del MB.

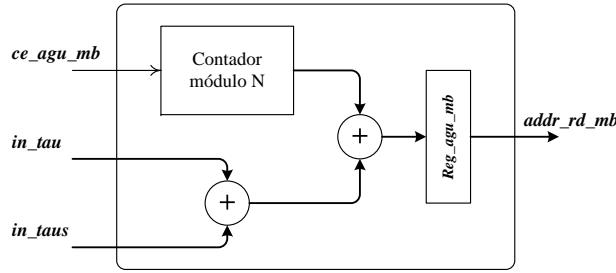


Figura 5.17: Arquitectura interna del módulo AGU_MB.

5.3.5. Memoria de almacenamiento temporal (TB)

Consiste de una RAM de doble puerto con una profundidad de P localidades. En ella se almacenan los coeficientes de la primera estimación del canal en las localidades generadas por el AGU_HEST. El direccionamiento a sus localidades esta manejado por medio de dos generadores de direcciones; AGU_HEST para operaciones de escritura y AGU_SYNC para las de lectura.

5.3.6. Generador de direcciones de escritura para el TB (AGU_HEST)

Este módulo se encarga de generar los valores de direcciones válidos para todas las operaciones de almacenamiento de datos en el TB. Básicamente está conformado por un contador binario ascendente módulo P configurado para funcionar en modo de conteo libre (*free running*). La habilitación del conteo está determinada por el nivel lógico presente en la señal de salida *done_ceof* del estimador CEOF.

5.3.7. Generador de direcciones de lectura para el TB (AGU_SYNC)

El AGU_SYNC genera los valores de direcciones válidos para todas las operaciones de lectura del TB. Esto incluye aquellas con las que se llevan a cabo los P desplazamientos circulares de los últimos $P - M$ coeficientes del primer estimado del canal, que

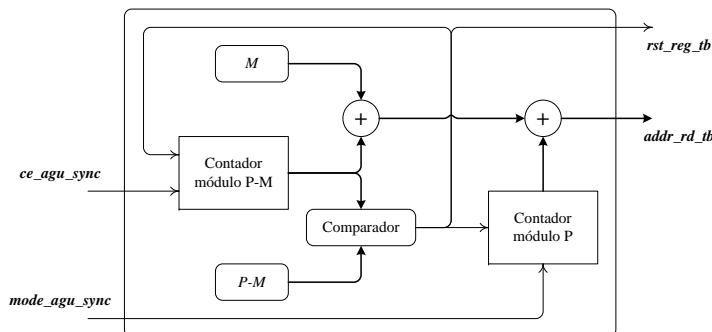


Figura 5.18: Arquitectura interna del módulo AGU_SYNC.

se requieren para corrección de TSS. Su estructura interna (figura 5.18) está compuesta por dos contadores ascendente conectados en cascada: el primero se encarga de que siempre sean accesadas $P - M$ localidades del TB. El segundo, con conteo módulo P , configura el desplazamiento necesario que se debe aplicar en cada iteración para generar los desplazamientos circulares.

5.3.8. Arquitectura para el cálculo de la norma Euclíadiana al cuadrado

La norma Euclíadiana es la operación fundamental en los algoritmos de corrección de TSS y BS. Sin embargo, por las razones ya descritas en el apartado 5.3.1, las ecuaciones (5.17a), (5.17b) y (5.19) se usaron en lugar de las originales en el diseño de una arquitectura para su cálculo. Los módulos de *hardware* FUNC_TERM_A (figura 5.19) y FUNC_TERM_B (figura 5.20) fueron diseñados para cada uno de los sumandos de (5.19). Para permitir su funcionamiento en paralelo ambos módulos se interconectaron al mismo flujo de datos ocasionando que el cálculo de (5.19) sea efectivamente acelerado. Además, seis etapas de encauzamiento (*pipeline*) fueron insertadas entre las operaciones matemáticas inmersas en tales módulos con el objetivo de reducir su ruta crítica. Adicionalmente, los módulos FUNC_TERM_A y FUNC_TERM_B cuentan cada uno con un secuenciador (secuenciador A y B), con los que se controla la transferencia de datos en cada una de las etapas del *pipeline*, tal como lo indican las figuras 5.19b y 5.20b. Con esta estrategia de temporización se contrarrestan las latencias adicionales introducidas por el *pipeline* y se incrementa el flujo de datos que procesan ambos módulos de la norma.

5.3.9. El módulo FMIN

En este módulo se procesan los vectores de datos de longitud P y N_P provenientes del módulo de la norma y se identifica al elemento con el menor valor en cada uno de ellos. En su diseño se considera lo expuesto en el apartado 5.3.1.1, donde se concluye que los elementos de los vectores están restringidos a tomar sólo valores reales y positivos. En la figura 5.21, se puede apreciar su composición interna. Dos registros internos *Reg_min_ant* y *Reg_ind_ant* inicializan todos sus bits a un valor de 1 lógico. Seguido, el valor de cada uno de los elementos que conforman a cada vector se alimenta al FMIN través de su bus de entrada *dat_in_fmin* y se compara uno a uno con el contenido almacenado en *Reg_min_ant*. El valor que resulte menor se usa para actualizar dicho registro y su posición dentro del arreglo hace lo mismo con el registro *Reg_ind_ant*. Al final, sólo el índice del elemento con menor valor dentro del vector en cuestión se entrega a través del bus de salida *ind_out_fmin*.

5.3.10. Resultados de implementación en FPGA

A través de un análisis estadístico de 316×10^6 datos, se caracterizó a la secuencia $x(k)$ que se recibe en el receptor y se monitoreó el valor de las variables relevantes dentro del algoritmo para determinar sus rangos dinámicos. El resultado de este experimento

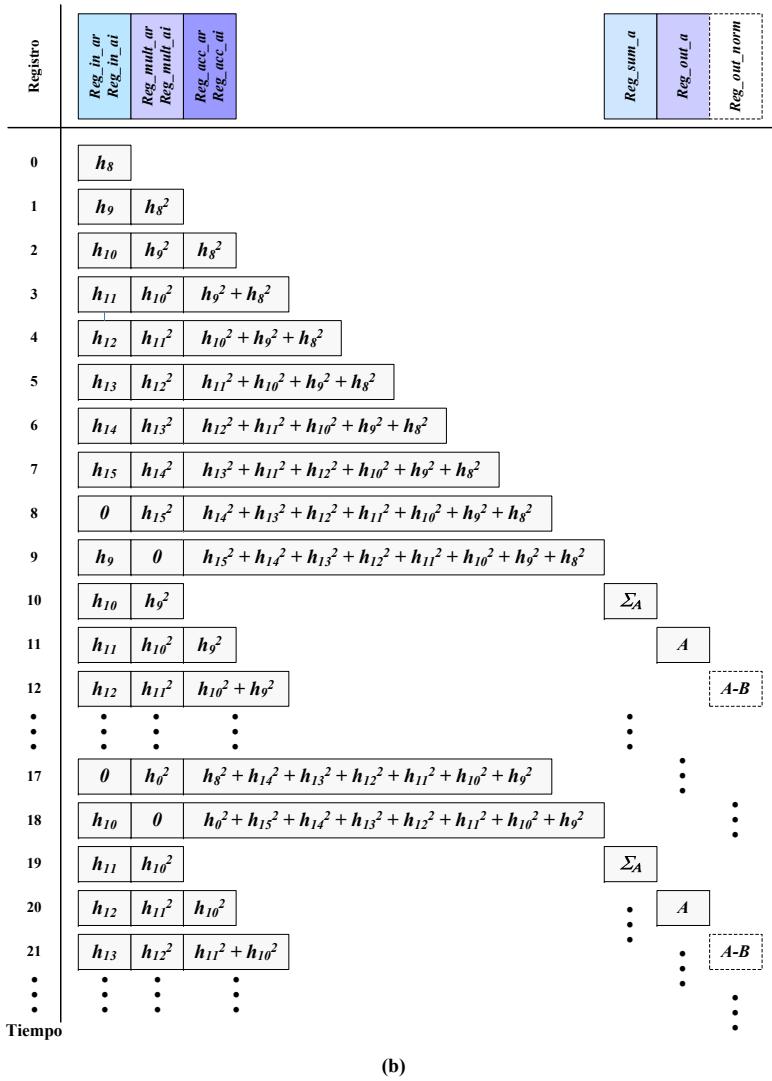
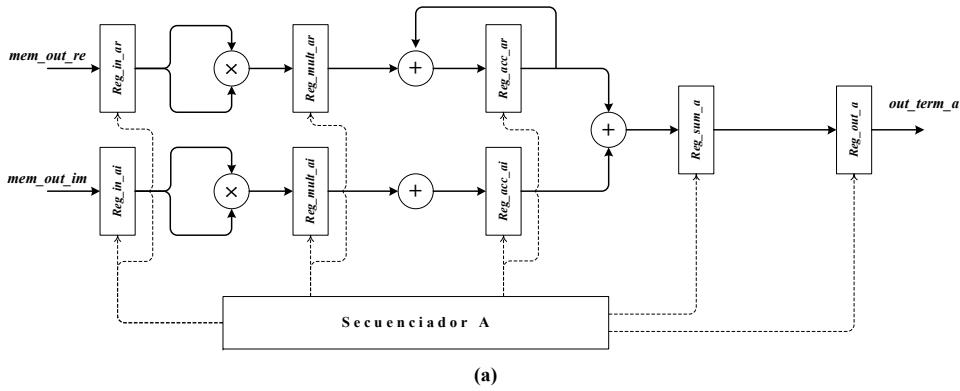


Figura 5.19: El módulo FUNC_TERM_A; a) Arquitectura, b) Temporización.

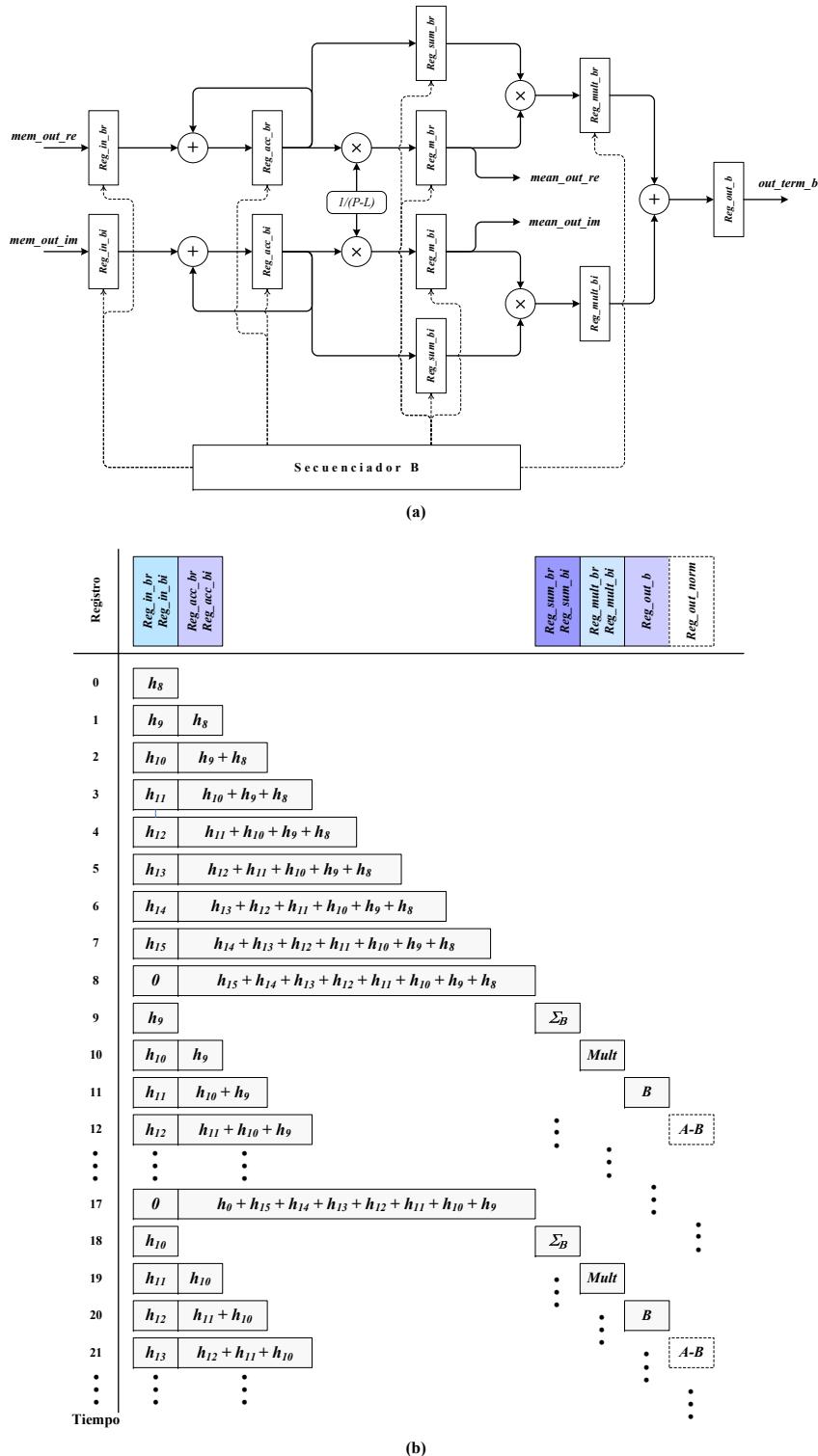


Figura 5.20: El módulo FUNC_TERM_B; a) Arquitectura, b) Temporización.

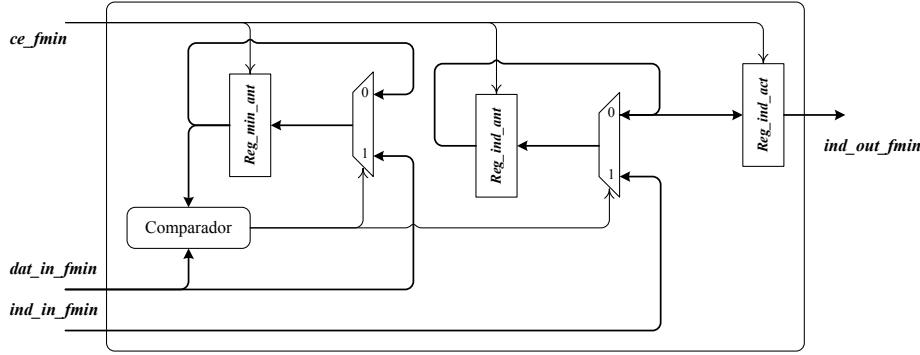


Figura 5.21: Arquitectura interna del módulo FMIN.

puede visualizarse en la tabla 5.5. Asimismo, tomando como referencia los resultados obtenidos en la implementación de los estimadores de canal previos (subsección 5.2) y sabiendo de antemano que el rango de operación del sistema va a situarse entre 0–25 dB, se pudo corroborar con el análisis de punto fijo que si se utiliza precisión comple-

	MB	TSS	Rango dinámico				
			Variable		Parte real		
			Máximo	Mínimo	Máximo	Mínimo	
		Secuencia recibida	$x(k)$	5.5049	-4.8782	4.9499	-5.6020
	TSS	Media cíclica	$y(\tau)$	1.9091	-1.1462	1.4297	-1.3519
		Desplazamiento de CD	m_{τ}	0.5047	-0.1761	0.2148	-0.4654
		Estimación de canal	h_{τ}	1.3438	-1.0712	1.1322	-1.2714
	Funcional TSS	Término A	A_{τ}	4.4224	0.0040	—	—
		Término B	B_{τ}	2.5611	1.3052×10^{-5}	—	—
		Funcional TSS	$A_{\tau} - B_{\tau}$	3.4975	3.8210×10^{-5}	* 3.9431×10^{-5}	** 1.2208×10^{-6}
	BS	Media cíclica	$y(\tau_P)$	2.0909	-1.2991	1.6044	-1.5707
		Desplazamiento de CD	m_{τ_P}	0.3418	-0.0538	0.0702	-0.2493
		Estimación de canal	h_{τ_P}	0.5783	-0.4014	0.3329	-0.4726
	Funcional BS	Término A	A_{τ_P}	1.4086	3.6747×10^{-4}	—	—
		Término B	B_{τ_P}	1.2981	8.6746×10^{-7}	—	—
		Funcional BS	$A_{\tau_P} - B_{\tau_P}$	0.4176	2.3575×10^{-5}	* 2.4392×10^{-5}	** 8.1659×10^{-7}
	Estimación verdadera de canal	Media cíclica	$y(\tau, \tau_P)$	1.9163	-1.0707	1.4073	-1.3621
		Desplazamiento de CD	m	0.3147	-0.0222	0.0622	-0.2201
		Estimación de canal	h	1.3641	-0.9836	1.0903	-1.2196

Tabla 5.5: Rangos dinámicos de las variables relevantes del algoritmo para el estimador ECEOOF. (Nota *: Es el segundo valor mínimo (antecesor); Nota **: Es la diferencia entre el valor mínimo y su antecesor).

ta en todas las operaciones del algoritmo se obtiene un rendimiento SQNR demasiado holgado (> 77 dB). Por tal motivo y dada la complejidad que implica la arquitectura de estimador ECEOOF, se optó por usar precisión limitada en la mayoría de las operaciones y truncamientos en las longitudes de palabra de los resultados intermedios. Evidentemente lo anterior va encaminado a tener un ahorro substancial de *hardware* a costa de degradar el SQNR del sistema, pero con la precaución de que en promedio ésta métrica se mantenga siempre por encima del rango de operación del sistema. La tabla 5.6, muestra las longitudes y formatos de palabra que se emplearon en el diseño de la arquitectura del estimador ECEOOF.

Variable		Q_w	Q_I	Q_F
Secuencia recibida	$x(k)$	12	4	8
Funcional TSS/BS	$A - B$	24	3	21
Desfase TSS	τ	4	4	—
Desfase BS	τ_P	5	5	—
Desplazamiento de CD	m	16	1	15
Media cíclica	$y(\tau, \tau_P)$	12	2	10
Canal estimado	h	16	2	14

Tabla 5.6: Longitudes y formatos de palabra empleados en las variables relevantes de la arquitectura ECEOOF.

Un aspecto importante identificado con la ayuda del análisis de punto fijo, es que tanto el funcional para TSS como el BS son muy susceptibles a las variaciones derivadas de la precisión finita de sus argumentos. Esto requiere particular atención debido a la diferencia que existe entre sus rangos dinámicos y al hecho de que sea un sólo módulo el encargado de evaluar a ambos. Por ende, la cantidad de bits para cuantizar la parte entera de la salida del módulo de *hardware* de la norma cuadrada lo determina el rango dinámico del funcional TSS por ser el mayor. En forma similar, la cantidad de bits para la parte fraccionaria la establece el funcional BS, por ser el que presenta la diferencia más pequeña entre el valor mínimo y el segundo valor mínimo del funcional. Retomando de la tabla 5.5 tales valores y sustituyendo en (4.1) y (4.3), se tiene que:

$$Q_I = \left\lfloor \log_2 [\max(3.4975, 3.8210 \times 10^{-5})] \right\rfloor + 2 = 3 \text{ bits},$$

$$Q_F = \left\lceil \log_2 \left(\frac{1}{2.3275 \times 10^{-7}} \right) \right\rceil = 21 \text{ bits} \quad \text{y} \quad Q_w = 24 \text{ bits}.$$

Toda vez definido las longitudes y formatos de palabra del estimador de canal ECEOOF, su arquitectura de *hardware* fue descrita a nivel RTL usando Verilog HDL para su posterior síntesis en un FPGA Virtex-5. Las condiciones empleadas en la herramienta de

desarrollo son similares a las enunciadas en la subsección 4.4. Sus parámetros de operación del estimador fijaron para procesar longitudes de bloques de $2(N + P)$ símbolos. El periodo de la secuencia de entrenamiento fue puesto a $P = 16$ con una potencia de $0.167\sigma_s$, además $N = 512$ con una longitud de prefijo cíclico $CP = 16$. En la tabla 5.7 se presentan los resultados de síntesis para la arquitectura del estimador ECEOOF en dos modelos de FPGA de la familia Virtex-5. En ambos se alcanzan frecuencias de operación superiores a los 130 MHz con consumos reducidos de registros y LUTs. El mayor consumo de recursos se da en los módulos DSP48Es debido a la cantidad y tipo de operaciones matemáticas que demanda el algoritmo. De hecho, la cantidad de tales módulos en el modelo xc5vlx110t resulta insuficiente y es por ello que se tuvo que elegir un FPGA Virtex-5 diferente.

	xc5vlx110t		xc5vfx100t	
Bloque de datos de entrada al estimador	2(N+CP)	1056	2(N+CP)	1056
Bloque de datos en estimaciones intermedias	(N)	512	(N)	512
Frecuencia de operación	(MHz)	131.990	(MHz)	145.449
Registros (<i>Slice registers</i>)	(69120)	2755 (3%)	(64000)	2755 (4%)
LUTs (<i>Slice LUTs</i>)	(69120)	2920 (4%)	(64000)	2428 (3%)
BRAMs (<i>Block RAMs</i>)	(148)	4 (2%)	(228)	4 (1%)
Módulos DSP48Es	(64)	66 (100%)	(256)	68 (26%)

Tabla 5.7: Resultados de síntesis de la arquitectura del estimador de canal ECEOOF en dos modelos de FPGA Virtex-5.

En lo que respecta a la cantidad ciclos necesarios para estimar los coeficientes del canal en la arquitectura ECEOOF, ésta va a ser aproximadamente la suma de los ciclos que le toma a cada uno de los cuatro procesos (recepción de datos, corrección de TSS, corrección de BS y estimación verdadera de canal) ejecutar todas sus tareas, i.e.

$$eceof_ciclos_{\hat{h}} = load_MB_{ciclos} + TSS_{ciclos} + BS_{ciclos} + CIR_{ciclos}, \quad (5.21)$$

donde:

$$load_MB_{ciclos} = 2(N + P) + 2, \quad (5.22a)$$

$$TSS_{ciclos} = \underbrace{(N + 4)}_{h_{\tau}} + \underbrace{(2P - 1)}_{funcional} + P \underbrace{(P - M + 1)}_{funcional} + 6, \quad (5.22b)$$

$$BS_{ciclos} = N_P \left[\underbrace{(N + 4)}_{h_{\tau_P}} + \underbrace{(2P - 1)}_{funcional} + \underbrace{(P - M + 1)}_{funcional} + 6 \right], \quad (5.22c)$$

$$CIR_{ciclos} = \underbrace{(N + 4)}_h + \underbrace{(2P - 1)}_m + \underbrace{(M + 1)}_m. \quad (5.22d)$$

Pero, debido a que la corrección de TSS se ejecuta en forma paralela durante la recepción de datos, entonces (5.21) queda expresada como:

$$eceof_ciclos_{\hat{h}} = load_MB_{ciclos} + BS_{ciclos} + CIR_{ciclos}. \quad (5.23)$$

Debido a que no existe en el estado del arte implementación alguna —incluso a nivel experimental— de un estimador de canal para DDST similar o con las mismas características que el estimador ECEOOF. Su desempeño se evaluó comparando las implementaciones hechas para los dos modelos de FPGA Virtex-5 (tabla 5.7). Los resultados de tal comparativa se visualizan en la tabla 5.8.

Modelo de Virtex-5	Bloque de datos de entrada al estimador	Corrección TSS (ciclos)	Corrección BS (ciclos)	Estimación de canal (ciclos)	TP (MS/s)	TP/area (MS/s/slices)
xc5vlx110t	1056	713	18016	19629	148.715	7.1
xc5vfx100t					134.954	7.82

Tabla 5.8: Rendimiento del estimador ECEOOF implementado en el FPGA Virtex-5.

Basados en la tabla anterior, se puede notar que el rendimiento del estimador ECEOOF es superior a los 7 MS/s en ambos casos, cifra bastante aceptable si se toma en cuenta que es capaz de ejecutar alrededor de 25752 sumas complejas y 8704 multiplicaciones complejas en cada estimación de canal (sin incluir las sumas y multiplicaciones reales que realiza el módulo de la norma Euclidiana al cuadrado).

5.3.11. Resultados de simulación

A continuación se presenta la simulación funcional de la arquitectura del estimador ECEOOF y su comparación respecto al modelo de oro en punto flotante programado Matlab. El escenario de simulación que se contempló fue el siguiente:

- (i) La longitud de la secuencia de datos de entrada es de dos bloques de 512 + 16 símbolos 4-QAM.
- (ii) La longitud del prefijo cíclico es de $CP = 16$.
- (iii) $c(k)$ es una secuencia OCI de periodo $P = 16$ y su potencia es de $0.167\sigma_s^2$.
- (iv) El canal es Rayleigh, selectivo en frecuencia y de tipo *block-flat-fading*.
- (v) El canal se genera al azar en cada realización, donde la parte real e imaginaria de $h(k)$ son variables aleatorias independientes e idénticamente distribuidas con PDF Gaussiana de media cero.
- (vi) La longitud del canal es $M = 7$ y su energía se normaliza a uno.

- (vii) A cada realización del canal se le introduce: un ruido blanco Gaussiano con media cero y con varianza acorde al SNR en cuestión, un desplazamiento aleatorio de DC en el rango de $0 \leq n \leq 0.3$ y un desplazamiento aleatorio de sincronización entre 0 y $N + P - 1$ muestras.

En la figura 5.22 se bosqueja el esquema de verificación que se utilizó en el estimador ECEOOF. Por principios de cuenta, se corroboró que todas las funciones y procesos para las que fue diseñada la arquitectura se lleven a cabo correctamente. Por lo tanto, fue necesario obtener los diagramas de tiempo (cronogramas) de operación de la arquitectura ECEOOF con la ayuda del Simulador ISE.

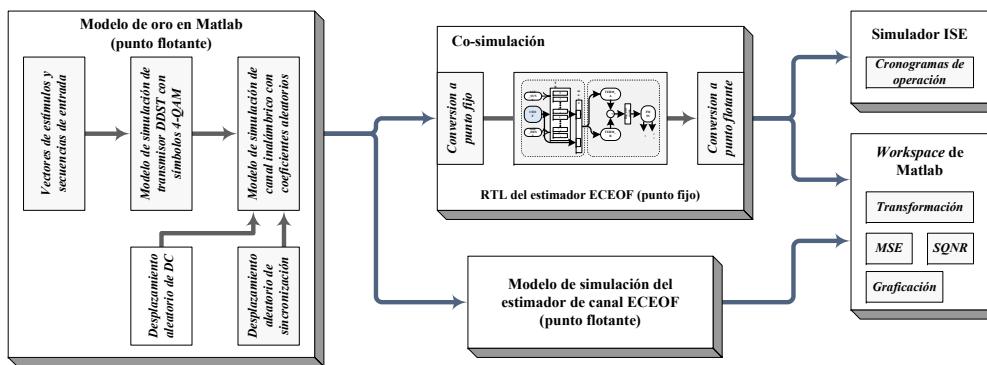


Figura 5.22: Esquema de verificación para la arquitectura del estimador de canal ECEOOF.

Las figura 5.23 muestra el diagrama de tiempos de un ciclo completo del estimador ECEOOF junto con los procesos necesarios para obtener el estimado del canal. Se puede observar que tanto la recepción de datos como la corrección de TSS son ejecutados al mismo tiempo y también como la corrección de sincronía de trama consume la mayor parte del tiempo de procesamiento del estimador. La figura 5.24 muestra la versión analógica de algunas señales del cronograma de la figura 5.23, donde ya es posible identificar con claridad las 34 estimaciones de la media cíclica (señal *yest*), el mismo número de estimaciones intermedias del CEOF (señales *CEOF hest*) así como el desplazamiento y rotación de sus últimos $P - M$ coeficientes, el comportamiento de la magnitud de los términos A y B del funcional de la norma cuadrada y las variaciones en el estimado del desplazamiento de DC. Asimismo, se puede notar el efecto que tiene el desplazamiento de DC sobre la componente real del estimado de la media cíclica (señal *yest(re)*), comparado con su contraparte imaginaria (señal *yest(im)*).

Por lo extenso del tiempo de duración de las simulaciones presentadas en las figuras 5.23 y 5.24, resulta difícil contemplar detalles particulares de cada uno de los procesos inmersos en la arquitectura ECEOOF. Por tal motivo, resulta necesario hacer ampliaciones (*zoom*) en ciertos instantes de tiempo de tales simulaciones para apreciar con plenitud cada una de las señales involucradas.

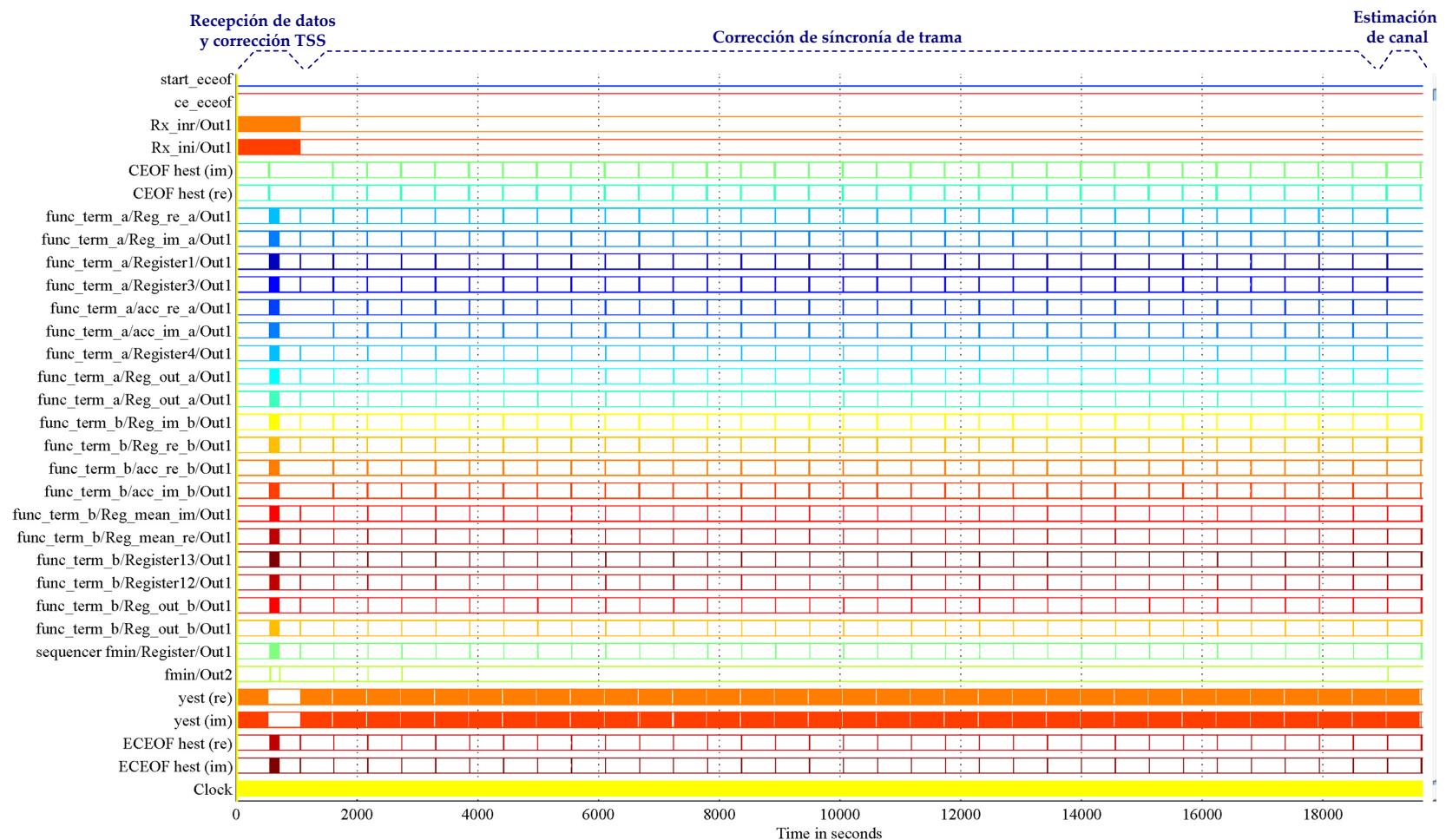


Figura 5.23: Cronograma de operación del estimador de canal ECEOFS (señales en forma digital).

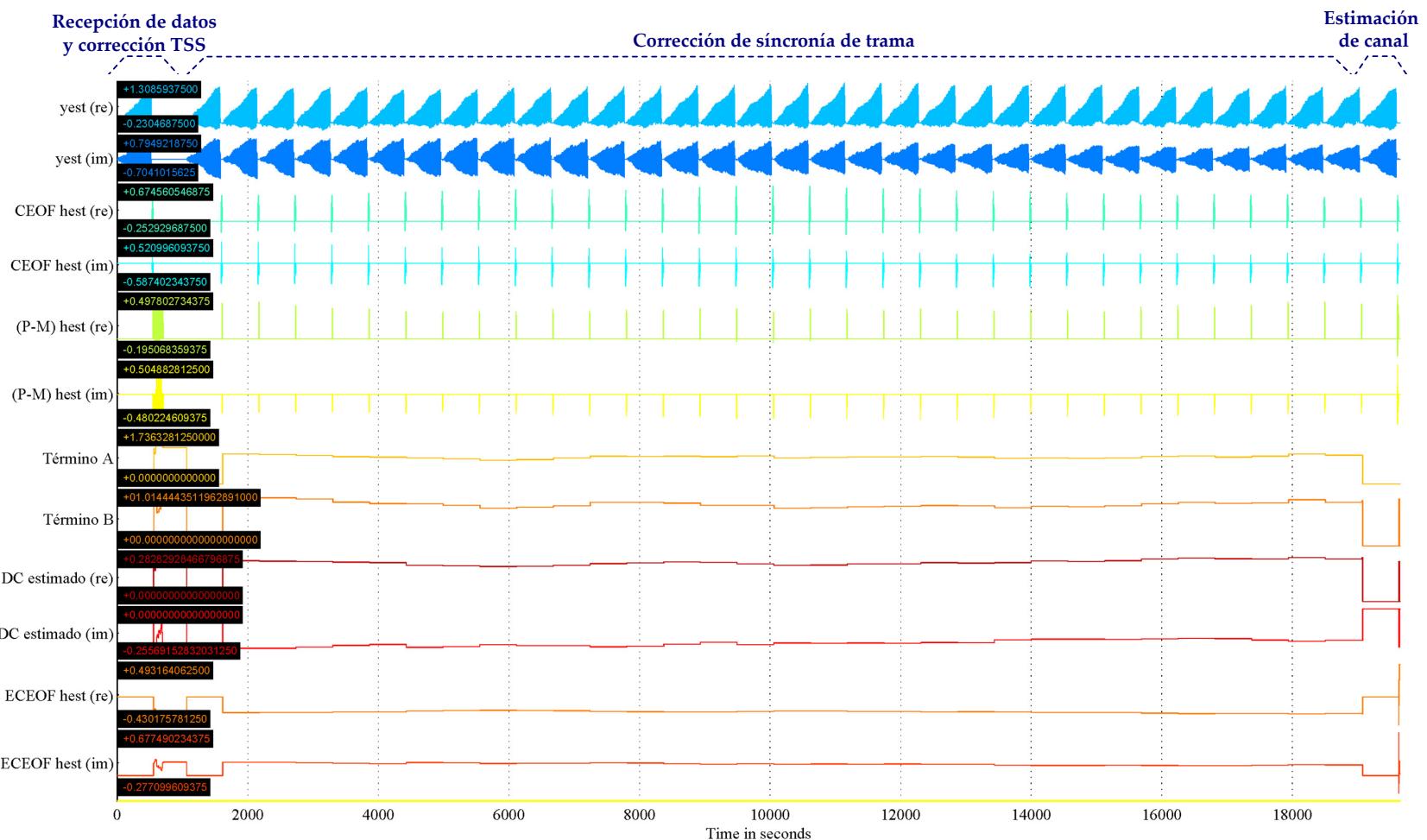


Figura 5.24: Cronograma de operación del estimador de canal ECEOFS (visualización de señales en forma analógica).

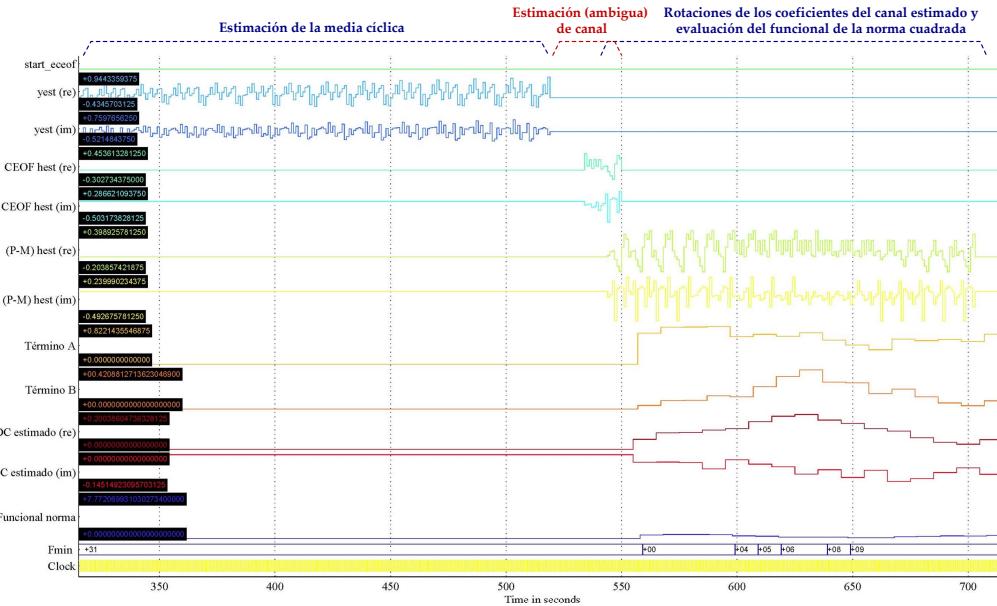


Figura 5.25: Diagrama de tiempo del proceso de corrección de TSS en el estimador ECEO.

La figura 5.25 muestra el comportamiento de las señales del estimador durante la corrección de TSS. Se puede distinguir el incremento de la magnitud de las componentes real e imaginaria del estimado de la media cíclica conforme transcurre el tiempo hasta que alcanza su valor final. También se nota como a partir del primer estimado del canal

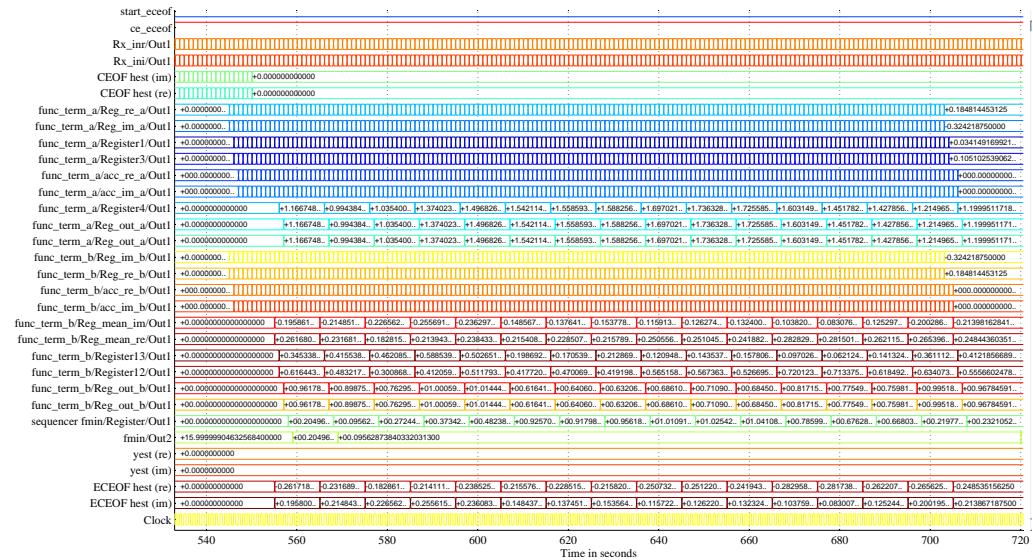


Figura 5.26: Cronograma de operación del funcional de la norma cuadrada durante el proceso de corrección de TSS.

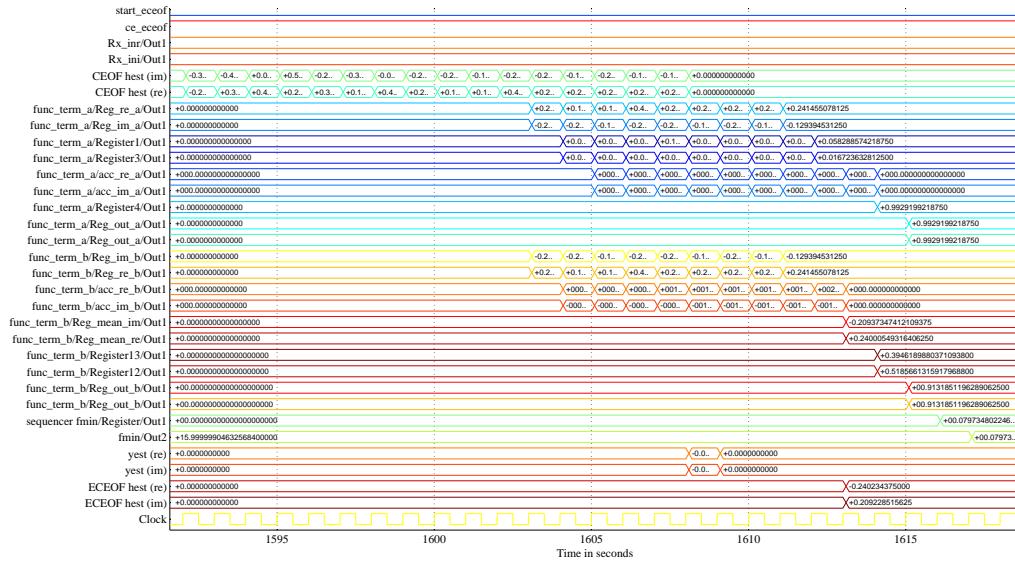


Figura 5.27: Señales del funcional de la norma cuadrada durante una de sus evaluaciones dentro del proceso de corrección de BS.

(señales *CEOOF hest*) se realizan las rotaciones de sus últimos $P - M$ coeficientes y los valores que estos generan en las señales *término A* y *término B* del funcional de la norma cuadrada. Finalmente, la señal *Fmin* muestra que el desfase de sincronía en la secuencia de entrenamiento entre el receptor y transmisor es de 9 muestras. Es importante resaltar, que durante la corrección de TSS nunca se retarda o interrumpe el flujo de datos que

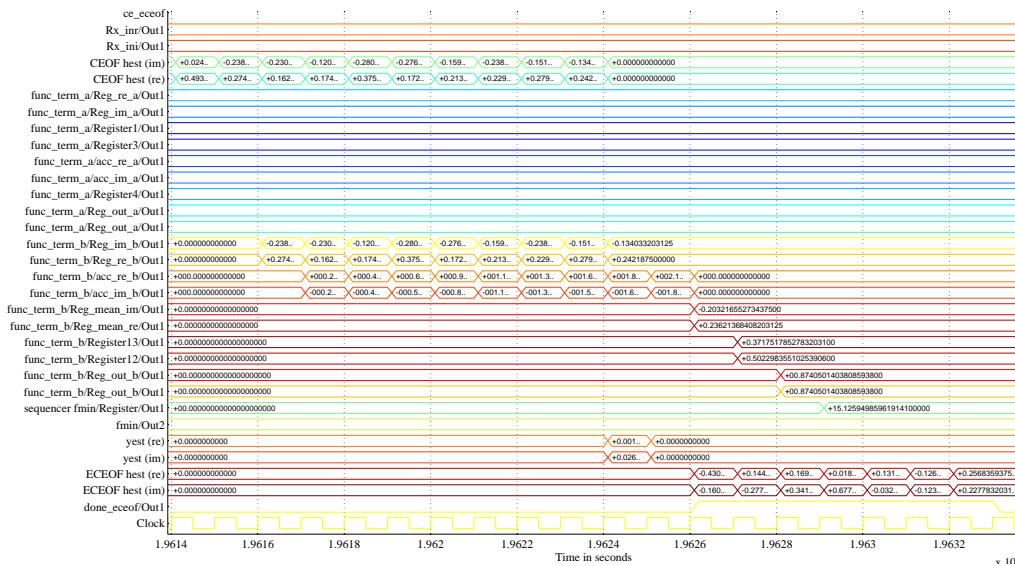


Figura 5.28: Señales del estimador ECEOOF para el proceso de estimación verdadera de canal.

procesan los módulos FUNC_TERM_A, FUNC_TERM_B del funcional de la norma y el módulo FMIN, tal como puede verse en la figura 5.26. Esto se debe en gran medida al buen diseño de las etapas de *pipeline* con que cuentan cada uno en su haber y a la correcta secuenciación de los datos que circulan a través de ella. En contraste, durante la corrección BS es necesario esperar a que se terminen de estimar la media cíclica y el canal antes de proceder a realizar una evaluación del funcional (figura 5.27). Esta gran dependencia de datos entre las operaciones mencionadas es la causante de la duración tan grande del proceso de corrección BS.

La figura 5.28 presenta como para el caso del proceso del estimación verdadera del canal, únicamente se habilita la operación del módulo FUNC_TERM_B del funcional de la norma para obtener el estimado del desplazamiento de CD. De igual manera se puede apreciar como las señales *CEOOF hest* son retardadas 18 muestras para dar oportunidad al funcional de estimar dicho parámetro. En la parte inferior de la gráfica se observa como se activa la señal *done_eceof* para indicar la validez de los valores del estimado verdadero del canal (señales *ECEOOF hest*). Finalmente, la magnitud de cada uno de los coeficientes en punto fijo de los estimados verdaderos, tanto de la media cíclica como del canal, obtenidos con la arquitectura propuesta se comparan con sus contrapartes en punto flotante que entrega el modelo de oro. Las gráficas de la figura 5.29 muestran el resultado de tal comparación, donde visualmente la diferencia entre las trazas de punto flotante respecto a la de punto fijo para ambos parámetros es imperceptible a grado tal que se superponen completamente.

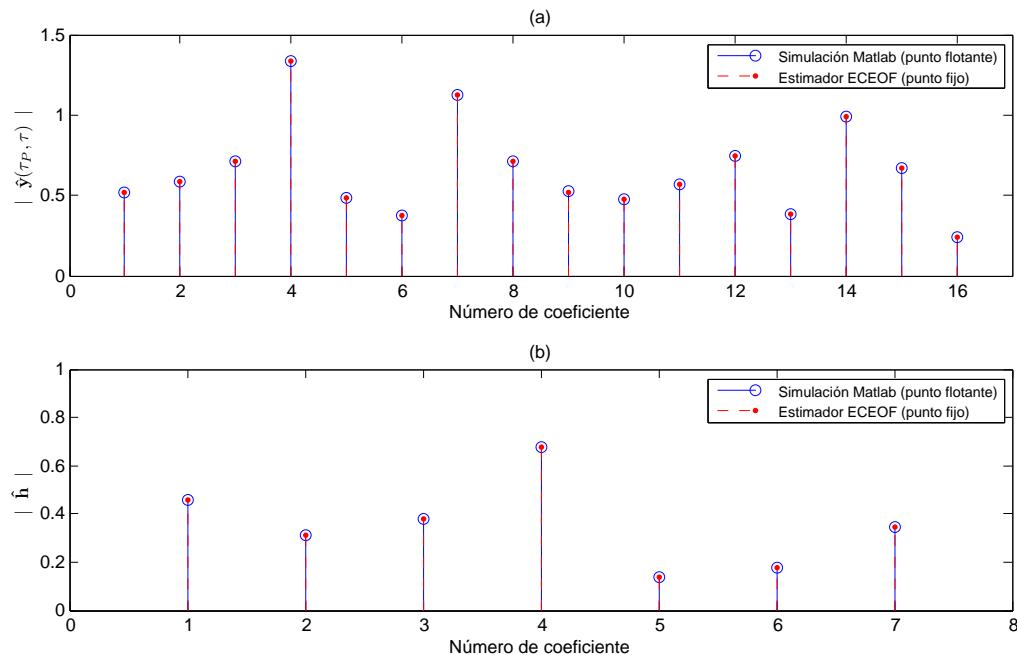


Figura 5.29: Estimados verdaderos realizados por el estimador ECEOF versus modelo de simulación en Matlab; a) Media cíclica, b) Canal.

Con respecto a las métricas que definen el desempeño funcional del estimador ECEO, un conjunto de 300 simulaciones Monte Carlo para cada valor de SNR del sistema fueron usados para evaluar el MSE del canal estimado y el SQNR promedio de las variables relevantes. En relación a la primera métrica, la figura 5.30 exhibe su comportamiento. En ella se puede apreciar que la diferencia entre las tres trazas es mínima dentro el rango de 0-20 dB de SNR. Sin embargo, en el rango de 20-25 dB, el rendimiento en el MSE de la arquitectura ECEO comienza a decrecer —y a separarse tanto de la traza teórica como la del modelo de oro— como consecuencia de la precisión finita de las variables. Esto significa que el estimador comienza a resentir los efectos del error de cuantización, el cual empieza a ser mayor que el ruido del sistema.

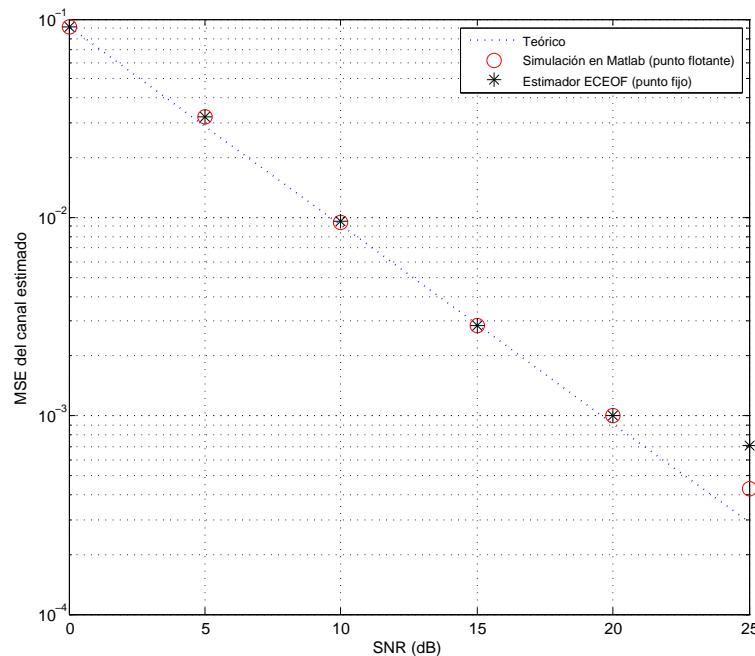


Figura 5.30: Rendimiento del MSE del canal estimado de la arquitectura ECEO *versus* el obtenido con el modelo de simulación en Matlab.

Para analizar los efectos de la aritmética de precisión fija, se cuantificó el SQNR individual en la salida del funcional de la norma cuadrada en cada una de las 300 simulaciones Monte Carlo. De igual forma se obtuvieron sus histogramas correspondientes donde se usó la regla de Scott [71] para determinar el número de clases. La figura 5.30 resume de manera gráfica los resultados de estas mediciones. Para el caso de corrección de TSS, claramente se puede observar en la figuras 5.30a y b) que el rendimiento SQNR se mantiene alrededor de los 60 dB en las primeras 1500 realizaciones (rango de 0-20 dB del SNR), pero en las últimas 300 (SNR=25 dB) el SQNR cae drásticamente a los 2-9 dB, pero a pesar de esto se tiene un SQNR promedio > 50 dB. Las figuras 5.31c y d) evidencian un rendimiento menor del SQNR del funcional para el caso de corrección BS. En las realizaciones que corresponden a un SNR= 0 dB, el SQNR se mantiene en

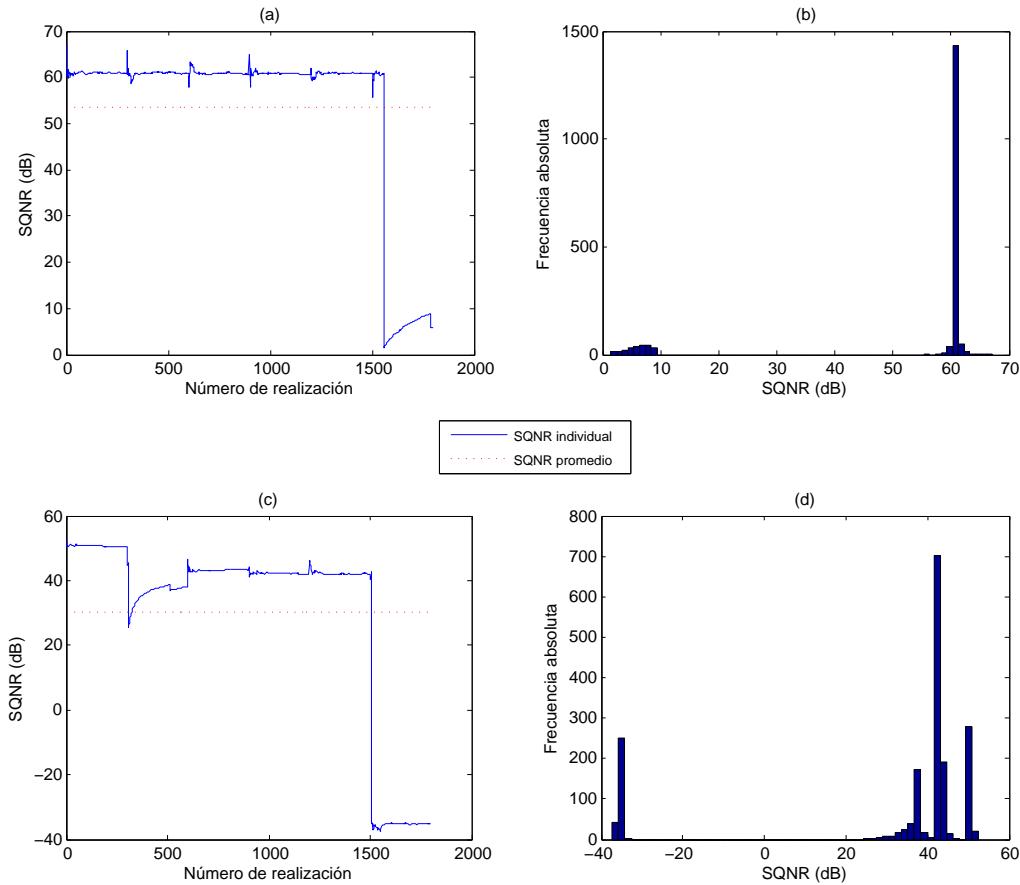


Figura 5.31: SQNR individual, promedio e histograma para 300 simulaciones Monte Carlo del funcional de la norma cuadrada del estimador ECEO; a) y b) Corrección de TSS, c) y d) Corrección de BS.

los 50 dB para luego caer por debajo del SQNR promedio (32 dB) cuando el SNR iguala los 5 dB. Este comportamiento se atribuye a que el algoritmo no fue capaz de estimar correctamente el desfase de BS. Las siguientes 900 realizaciones (SNR de 10-20 dB) muestran un SQNR casi constante de poco más de 43 dB. En las realizaciones para un SNR de 25 dB, el funcional ya no es capaz de entregar un resultado confiable.

Lo concerniente al rendimiento SNQR de los estimados verdaderos de la arquitectura ECEO se visualiza en las gráficas e histogramas de la figura 5.32. El SQNR del estimado del desplazamiento de CD se comporta de manera oscilante alrededor de su SQNR promedio de 35 dB. Se puede ver en el histograma de la figura 5.32b como en una de las realizaciones el SQNR cae hasta los -6 dB, las restantes quedan distribuidas en entre los 7-47 dB. Para el estimado verdadero de la media cíclica se tiene un SQNR promedio de 55 dB. La parte final de la gráfica de la figura 5.32b muestra las repercusiones negativas en el estimado de la media cíclica que tienen los resultados erróneos del funcional de la norma para la corrección BS. Asimismo, comparando las gráficas 5.32c

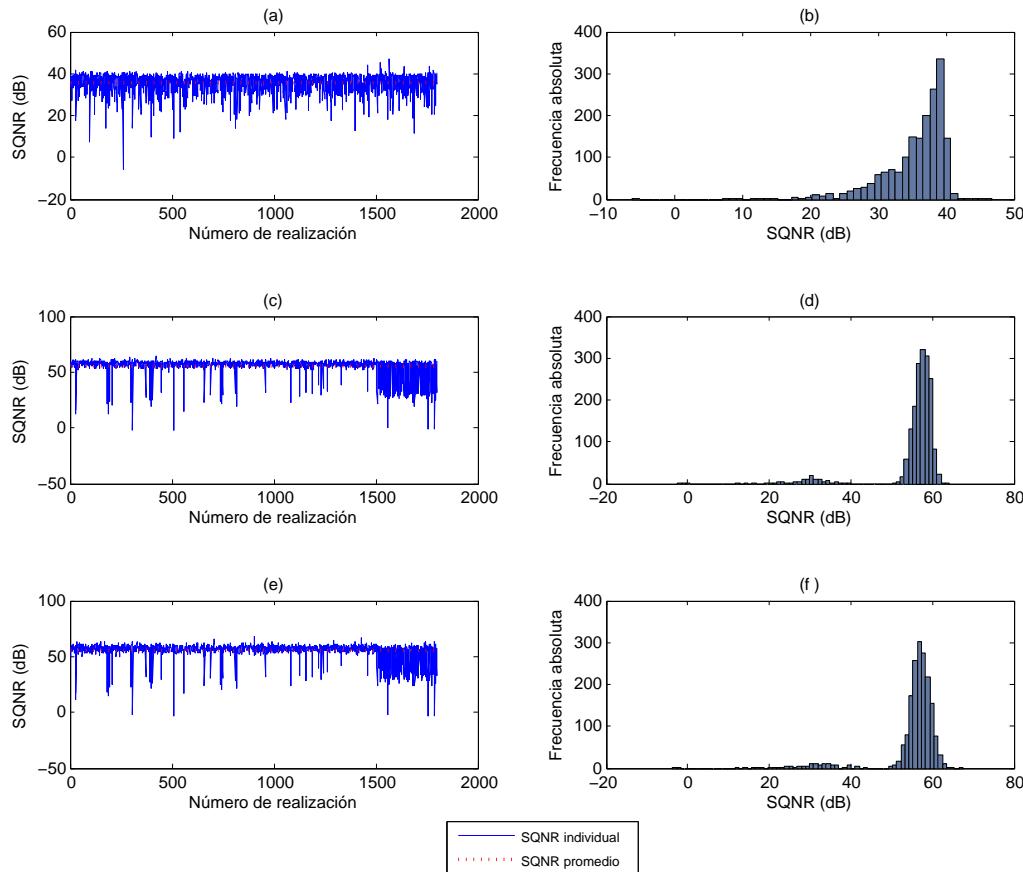


Figura 5.32: SQNR individual, promedio e histograma para 300 simulaciones Monte Carlo de las salidas del estimador ECEO; a) y b) Desplazamiento de CD, c) y d) Estimado verdadero de la media cíclica, e) y f) Estimado verdadero del canal.

y e) se puede constatar el alto grado de correlación entre ambas debido a que el estimado verdadero del canal depende de que tan bien la arquitectura haya estimado la media cíclica. En general, como lo indica la figura 5.32e, el rendimiento SQNR promedio en la estimación verdadera del canal es de 55 dB.

5.4. Arquitecturas de *hardware* para la FFT

En la sección 3.4 se expusieron varios algoritmos para la FFT que reducen la complejidad $O(N^2)$ del cálculo original de la DFT. Ahora, en este apartado se abordarán los pormenores en el desarrollo de arquitecturas para procesadores FFT, en particular aquellas basadas en memoria (*memory-based FFT*), donde a menudo se tienen uno o dos bloques de memoria centralizados de donde se acceden todos los datos. Tales diseños están enfocados a incrementar la tasa de utilización del elemento procesador (*butterfly* o *dragonfly*) y reducir la redundancia de múltiples de ellos. Asimismo, debido a que los

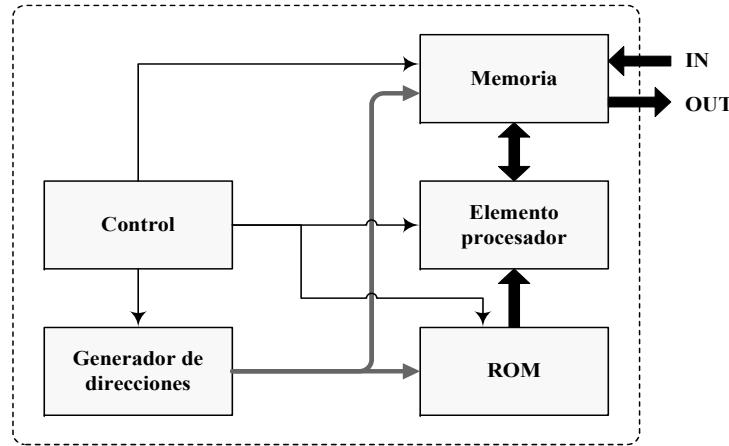


Figura 5.33: Diagrama a bloques de un procesador FFT basado en memoria.

resultados de salida son almacenados en las mismas localidades de memoria de donde fueron leídos, las arquitecturas son de tipo *in-place* [72, 73]. Esto minimizará la cantidad de memoria necesaria e impactará directamente en el consumo de *hardware*.

Por otro lado, un procesador FFT genérico basado en memoria consiste por lo regular de: un controlador, una ROM de coeficientes, un generador de direcciones, un elemento procesador y un bloque de memoria; tal como se ilustra en la figura 5.33.

A continuación expone el desarrollo de las arquitecturas para:

- ▷ Procesador FFT/IFFT radix–2 con decimado en tiempo.
- ▷ Procesador FFT/IFFT radix–2 con decimado en frecuencia.

Vale la pena mencionar, que en todos ellos su operación se partitiona en tres grandes procesos que son: Almacenamiento de los datos de entrada, cálculo de la FFT o IFFT y descarga de los resultados. Igualmente, el resultado de la FFT se entrega escalado entre N para limitar posibles *overflows* durante los cálculos.

5.4.1. Procesador FFT/IFFT radix–2 decimado en tiempo

La arquitectura digital propuesta para el cálculo de la FFT DIT–2, definida previamente con (3.71) y (3.72), se muestra en la figura 5.34. En ella se identifican los siete módulos que la conforman: un *butterfly* DIT–2, una RAM (RAM_FFT), una ROM (ROM_FFT), dos generadores de direcciones (AGU_IO y AGU_DIT2), un reordenador y un controlador. De forma resumida, el flujo que siguen los datos a través de tales módulos para el cálculo de la FFT (IFFT) es el siguiente. Tan pronto se activa la señal *start_fft*, un bloque de N datos se lee a través de la entrada *IN_FFT* y se almacena en RAM_FFT. Cada dato se guarda en la localidad correspondiente a su índice pero en binario inverso. Luego, el control desvía el flujo de datos hacia el *butterfly* y dependiendo de valor lógico

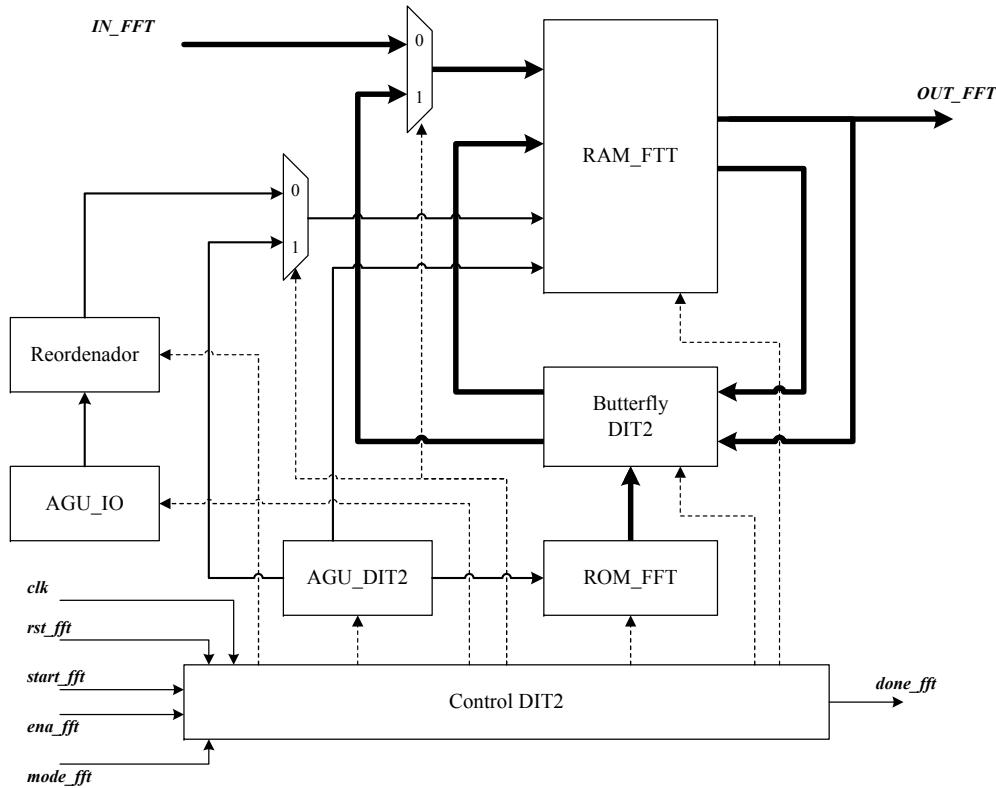


Figura 5.34: Arquitectura digital para el procesador FFT/IFFT radix-2 con decimado en tiempo.

que tenga la entrada *mode_fft* lo configurará para ejecutar ya sea la FFT o la IFFT (uno lógico). Durante $N \log_2(N)$ ciclos, el *butterfly* leerá un par de datos de la RAM_FFT, los procesará y retornará el par de resultados a dicha memoria para que los almacene, sobre escribiendo su contenido. Una vez que se han procesado todos los datos, la RAM_FFT tendrá en orden consecutivo los valores de la FFT del bloque de datos. Para terminar, se activa la señal *done_fft* y uno a uno los resultados se extraen de la RAM y se envían al bus de salida *OUT_FFT*.

5.4.1.1. Generador de direcciones de I/O

El módulo AGU_IO se encarga de generar las direcciones de la RAM_FFT para las fases de carga y descarga de datos. Esencialmente es un contador binario ascendente módulo N , por lo que la anchura de su bus de salida es de $\log_2(N)$ bits.

5.4.1.2. Módulo reordenador

Una de las pocas desventajas de las que adolecen los algoritmos para la FFT, es la necesidad de aplicar algún tipo de reordenamiento en la secuencia de datos. En la FFT

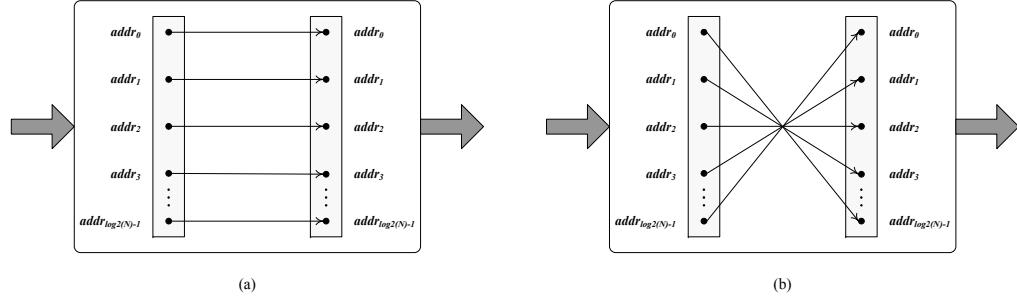


Figura 5.35: Estructura interna del módulo de reordenamiento; a) Interconexión directa, b) Interconexión inversa.

DIT-2 un reordenamiento de tipo *bit-reversal* se debe realizar a los datos de entrada antes de su procesamiento. A menudo, en la literatura la solución a nivel *hardware* se presenta en forma de un *buffer* de memoria, en donde los datos se guardan en forma consecutiva para posteriormente extraerse en orden binario invertido. Esto implica incluir una memoria adicional de N localidades en la arquitectura y consumir $2N$ ciclos extras para el reordenamiento [74]. Para evitar esto, el módulo que se desarrolló cambia el orden de los datos alterando la secuencia de su almacenamiento en la RAM_FFT. Esto a través de un *switch* alambrado (*hard-wired*) que modifica la interconexión entre las líneas del bus de salida del AGU_IO y el bus de direccionamiento de la RAM_FFT; en la forma como lo ejemplifica la figura 5.35b. Por lo tanto, el producto final es un módulo sencillo pero ingenioso que efectúa el reordenamiento de los datos de entrada “al vuelo” conforme se van almacenado en memoria, sin latencia y con el valor agregado de un consumo casi nulo de área.

5.4.1.3. Memoria RAM_FFT

Una memoria RAM de doble puerto de N localidades se usa para mantener almacenados los datos de entrada, los valores de operaciones intermedias y los resultados de la transformación final. Su característica principal permite que un par de datos ubicados en localidades distintas puedan ser accesados por el *butterfly* al mismo tiempo, proporcionando acceso paralelo. Su descripción es lo suficientemente genérica para que cualquier herramienta de desarrollo la infiera como un bloque de memoria dedicado (Block-RAM), evitando construirla en forma distribuida con los recursos del FPGA.

5.4.1.4. Memoria ROM_FFT

Los factores rotatorios definidos en (3.69) son constantes conocidas *a priori*, por ende se calculan y cuantizan *off-line* para almacenarse en una ROM de puerto sencillo a manera de LUT. Para el caso de la FFT radix-2, únicamente se necesitan $\frac{N}{2}$ factores rotatorios independientemente del tipo de decimado que se aplique.

5.4.1.5. Butterfly DIT-2

Es la unidad de PDS elemental de la FFT DIT-2, que ejecuta el cálculo de las ecuaciones (3.71) y (3.72). Está conformado de un multiplicador complejo, dos sumadores complejos y dos desplazadores aritméticos; interconectados tal y como se muestra en la figura 5.36. Los N datos son leídos secuencialmente desde la RAM_FFT en pares junto con el factor rotatorio correspondiente. En caso de que la operación que se ha elegido sea la IFFT ($mode_fft = 1$), se debe obtener además el complejo conjugado de los factores rotatorios antes de que se envíen al multiplicador. Para minimizar los efectos adversos debido al uso de la aritmética de punto fijo, las salidas del *butterfly* siempre se escalan en un factor de $\frac{1}{N}$ por medio de desplazadores aritméticos. Por lo tanto, un escalamiento total de $\frac{1}{N}$ se distribuye uniformemente a lo largo del algoritmo. Esto reduce el error de propagación e incrementa el SQNR [62, 75].

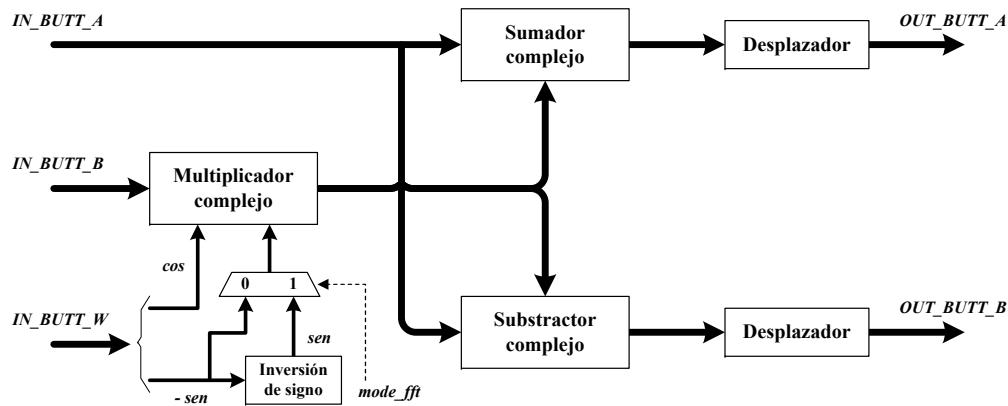


Figura 5.36: Estructura interna del *butterfly* DIT-2.

5.4.1.6. Generador de direcciones para FFT DIT-2 (AGU_DIT2)

Considerando las características del *butterfly* DIT-2, se diseño una estrategia de direccionamiento *in-place* para las memorias que fuera válida tanto para el modo de operación FFT como el de IFFT. Tomando como referencia el flujo de datos de la figura 3.4, las direcciones para cada puerto de la RAM_FFT y la ROM_FFT se pueden obtener —si se conoce la etapa actual (*etapa_act*) que se esté procesando en el algoritmo y se tiene identificado cuál de los $\frac{N}{2}$ *butties* de la etapa en cuestión es el que se está ejecutando (*butt_act*)— usando las siguientes expresiones:

$$addr_rom = butt_act \ll temp, \quad (5.24)$$

$$addr_pa = rotate(0 \& aux, temp, derecha), \quad (5.25)$$

$$addr_pb = rotate(1 \& aux, temp, derecha), \quad (5.26)$$

con

$$aux = \text{rotate}(butt_act, temp, izquierda),$$

$$temp = \log_2(N) - etapa_act,$$

donde $\text{rotate}(dat, numb, direc)$ rota circularmente $numb$ bits del dato dat hacia la dirección $direc$, “&” es el operador concatenación y “ $<<$ ” denota la operación de desplazamiento lógico a la izquierda.

De conformidad con (5.24–5.26), se diseñó la arquitectura que se observa en la figura 5.37 para el AGU_DIT2. Un contador ascendente monitorea el número de *butterflies* que se ejecutan en cada etapa. Su salida se modifica combinacionalmente a través de un conjunto de desplazadores y rotadores lógicos (*barrel shifters*) acorde a la etapa actual y al tipo de operación.

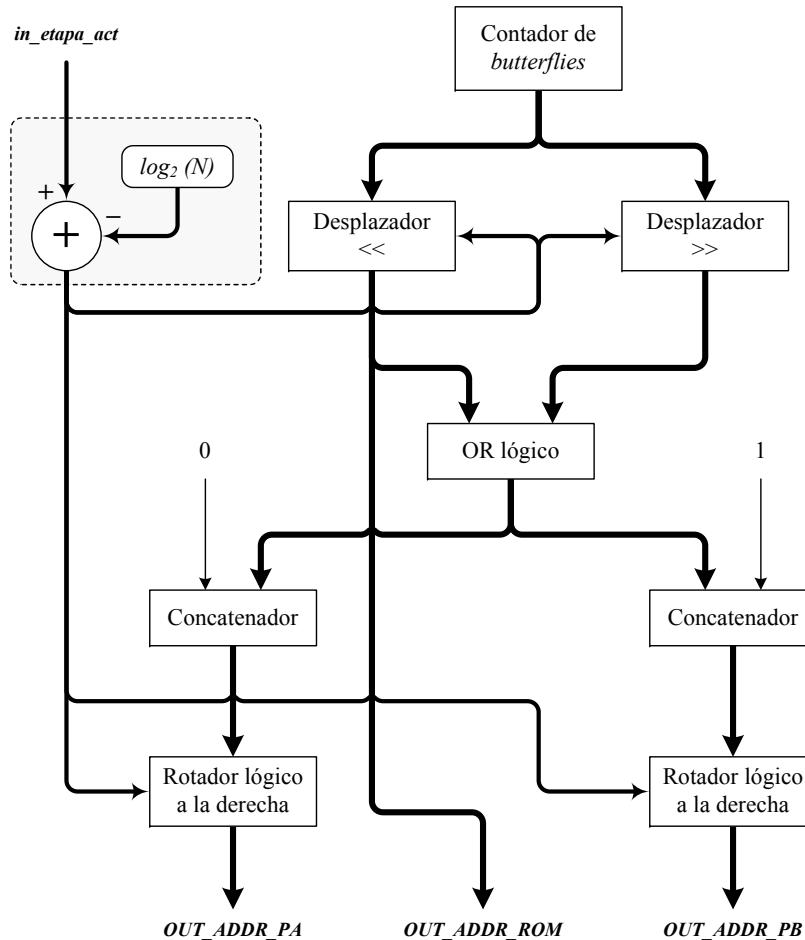


Figura 5.37: Arquitectura del módulo generador de direcciones para la FFT DIT-2.

5.4.1.7. Módulo de control DIT-2

La FSM que se presenta en la 5.38 es la encargada de secuenciar todas las tareas que se ejecutan durante las tres fases de operación del procesador FFT y de su respectiva señalización. Por principio de cuentas, detecta la activación de la señal *start_fft* para proceder a la carga y reordenamiento de los datos. Hecho esto, supervisa que se ejecuten todas las etapas del algoritmo con sus respectivos *butterflies*. Para finalizar, activa la señal de salida *done_fft* e inicia la descarga de los resultados, para posteriormente quedar en espera de una nueva activación.

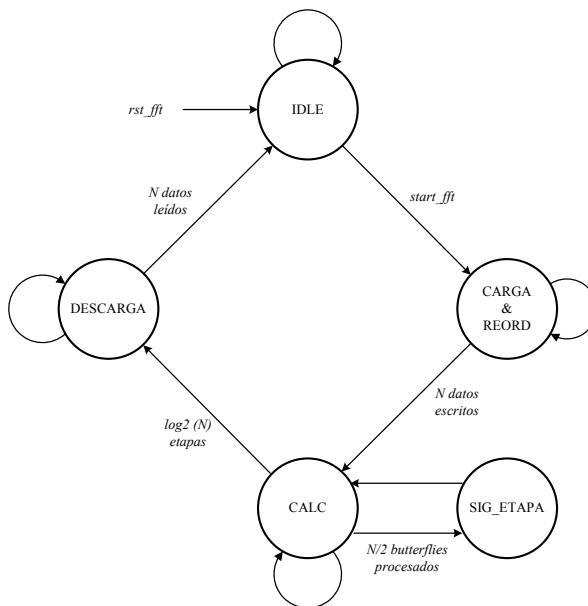


Figura 5.38: Máquina de estados finitos del control DIT2.

5.4.2. Procesador FFT/IFFT radix-2 con decimado en frecuencia

Aprovechando la ventaja que ofrece la similitud entre los algoritmos FFT DIT-2 y FFT DIF-2, en el diseño de la arquitectura de éste último fue posible reusar varios de los módulos que ya se habían desarrollado para la variante DIT-2. Salvo lo concerniente al reordenamiento de datos, todas las demás tareas se llevan a cabo en la misma secuencia. De hecho, la estructura en la arquitectura es semejante a la de su contraparte DIT-2, tal como queda en evidencia en la figura 5.39. Los únicos módulos que marcan la diferencia son: el generador de direcciones (AGU_DIF2), el elemento procesador (*butterfly DIT2*) y el controlador (control DIF2). Esto como consecuencia de que: los datos son leídos en una orden distinto en cada etapa, ahora en los cálculos se usan las ecuaciones (3.75–3.76) y el reordenamiento se hace ahora en la fase de descarga. Detalles específicos de tales módulos se exponen en los siguientes párrafos.

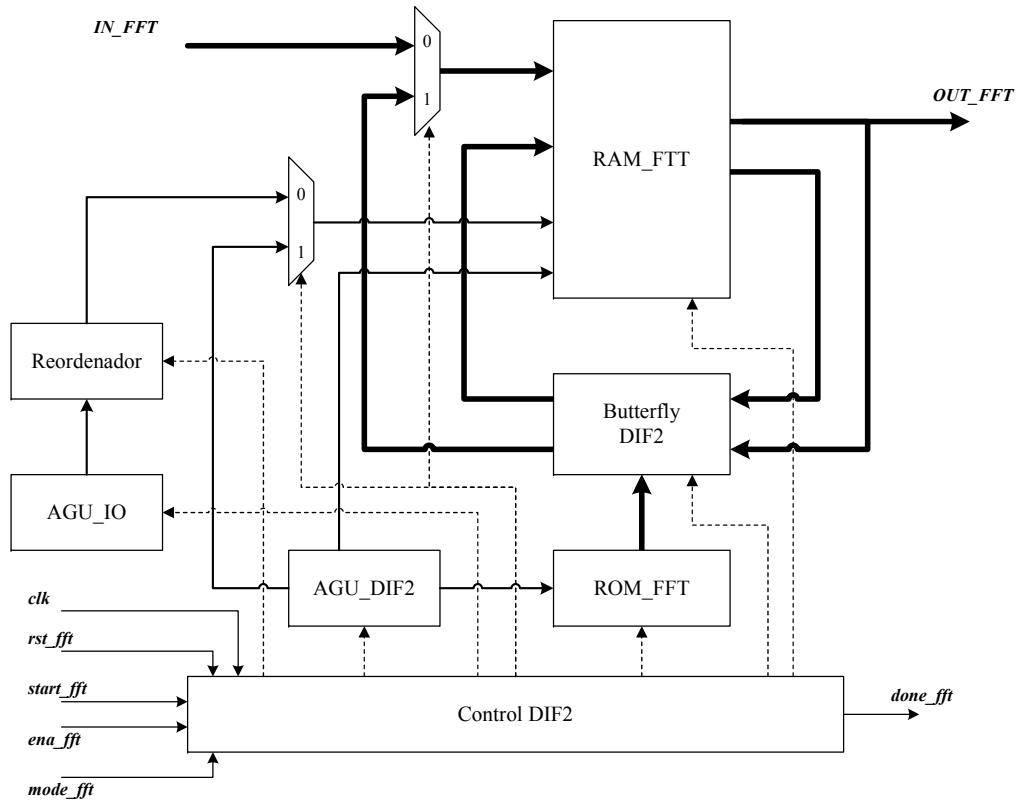


Figura 5.39: Arquitectura digital para el procesador FFT/IFFT radix-2 con decimado en frecuencia.

5.4.2.1. Generador de direcciones DIF-2 (AGU_DIF2)

La gran flexibilidad que presenta el esquema de direccionamiento presentado en el apartado 5.4.1.6, permite que su diseño pueda extrapolarse para el procesador FFT DIF-2 introduciendo modificaciones mínimas. Por lo tanto, todo lo que se mencionó para el AGU_DIT2 —incluyendo (5.24-5.26)— prevalece para el AGU_DIF2, a excepción de la ecuación para *temp*. Analizando el flujograma mostrado en la figura 3.6, se puede inferir que la expresión para calcular *temp* cambia a

$$\text{temp} = \text{etapa.act} - 1. \quad (5.27)$$

Por lo tanto, el módulo AGU_DIF2 es exactamente el mismo que se diseñó para su símil DIT-2, excepto por la parte sombreada de la arquitectura en la figura 5.38, la cual ahora se sustituye con los bloques que se muestran en figura 5.40. Estos corresponden al mapeo en hardware de (5.27).

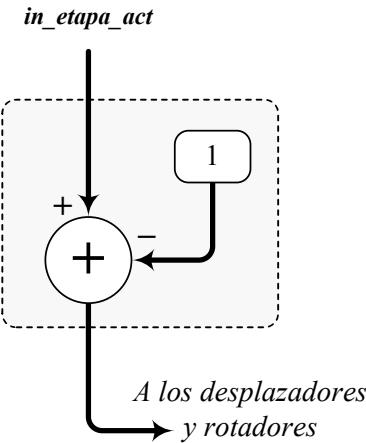


Figura 5.40: *Hardware* para el cálculo de *temp* usando (5.27) en el AGU_DIF2.

5.4.2.2. Butterfly FFT DIF-2

La razón de que tanto el algoritmo de decimado en tiempo y como el de frecuencia tengan la misma complejidad computacional, tiene su origen en el hecho de que en sus *butterflies* (figuras 3.3 y 3.5) se ejecuten el mismo tipo y número de operaciones (una multiplicación y dos sumas, todas complejas). Por lo tanto, en el diseño del *butterfly* DIF-2 se emplearon los mismos bloques aritméticos de su contraparte DIT-2. Sin embargo, acorde a lo que se establece en (3.75–3.76), la interconexión entre tales bloques es distinta, tal como puede notarse en la figura 5.41. Asimismo, todo lo que ya se discutió relativo al manejo de los factores rotatorios cuando *mode_fft* = 1 y al escalado en las salidas; sigue aplicando para este *butterfly*.

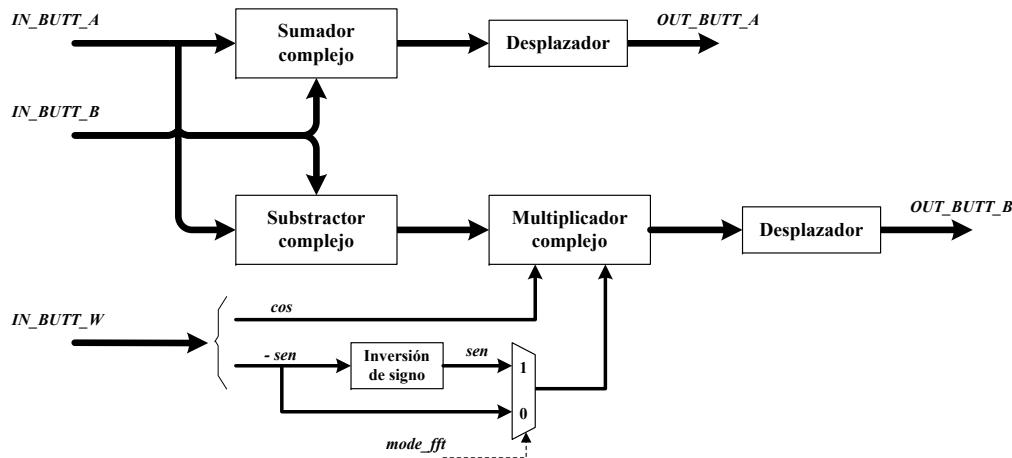


Figura 5.41: Estructura interna del *butterfly* DIF-2

5.4.2.3. Módulo de control DIF-2

Nuevamente, sólo una ligera modificación en el control DIT-2 es lo que se necesita para poder aplicarlo en la FFT DIF-2. En la fase de carga, el control DIF-2 debe configurar una interconexión directa entre el AGU_IO y el módulo reordenador (5.35a). Esto asegura que los datos de entrada se almacenen consecutivamente en la RAM_FFT conforme van arribando al bus *IN_FFT*. Las tareas en la fase de procesamiento permanecen invariantes. Por el contrario, en la última fase de operación del procesador, el control DIF2 debe configurar una interconexión inversa en el reordenador para que los datos sean extraídos de la memoria en orden binario invertido, tal como lo dicta el algoritmo. Estos cambios se ven reflejados en la FSM de la figura 5.42 correspondiente al control DIF-2.

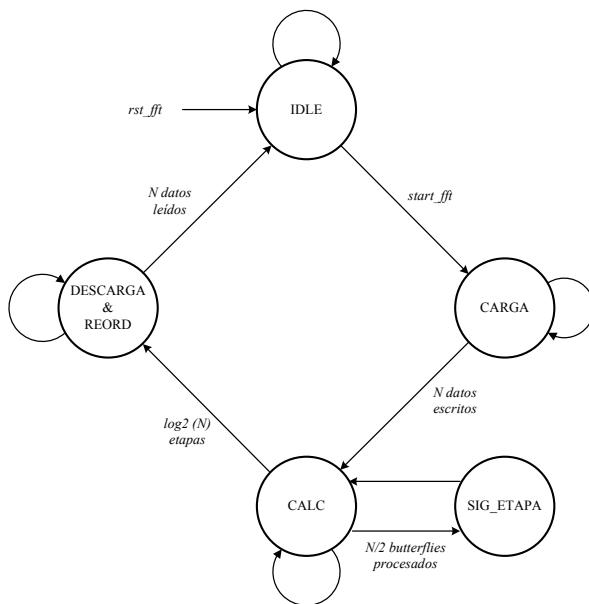


Figura 5.42: Máquina de estados finitos del control DIF2.

5.4.3. Resultados de implementación y simulación

Se describieron en verilog HDL para ambos procesadores FFT, arquitecturas parametrizables tanto en el número de muestras como en el ancho de la palabra con el fin de explorar sus rendimientos y consumos de *hardware* ante diversos valores de N y Q_W . Asimismo, en la FFT se puede establecer *a priori* que el rango dinámico que va experimentar la señal de entrada es de ± 1 . Con lo que, los resultados intermedios que se generan en cada etapa de la FFT tampoco sobrepasaran dicho rango a consecuencia del escalado que se aplica en el *butterfly*. Todas estas consideraciones permitieron que se tuviera un ancho de palabra constante con un formato de palabra invariante en toda la arquitectura.

Arquitectura	Tamaño	Frecuencia de operación	Slices	Slices FFs	LUTs de 4 entradas	BRAMs	MULT18x18b
(Q _w = 16 bits)	(N)	(MHz)	(4656)	(9312)	(9312)	(20)	(20)
FFT DIT-2	512	60.35	256	5%	45	<1%	461 4% 2 10% 4 20%
FFT-DIT-2	1024	60.32	274	5%	69	<1%	492 5% 3 15% 4 20%
FFT-DIF-2	2048	59.27	285	6%	75	<1%	516 5% 6 30% 4 20%
FFT-DIF-2	4096	58.28	302	6%	82	<1%	546 5% 12 60% 4 20%

Tabla 5.9: Resultados de síntesis para procesadores FFT DIT-2 y FFT DIF-2, para tamaños de transformada de 512, 1024, 2048 y 4096 puntos y con ancho de palabra de 16 bits.

La tabla 5.9 presenta un resumen de la síntesis en un FPGA Spartan 3E de Xilinx; de los procesadores FFT para ambos tipos de decimado, con diferentes tamaños y con un $Q_W = 16$ bits. Los valores entre paréntesis en cada *item* señalan el total disponible en el FPGA del recurso en cuestión. Analizando dicha tabla, se puede resaltar que la frecuencia de operación sólo disminuye marginalmente no obstante que el tamaño de la transformada se duplica en cada procesador. Mismo comportamiento experimentan los *items* de *slices*, *slices FFs*, LUTs de 4 entradas y multiplicadores dedicados. Esto es indicativo de que la complejidad de la arquitectura no crece en la misma proporción del tamaño de la FFT y por lo tanto, la ruta crítica de los datos tampoco sigue esa tendencia. Unicamente aumenta la cantidad de memoria en el procesador conforme lo hace el tamaño de la secuencia a transformar. Por lo tanto, se puede afirmar que las arquitecturas FFT DIT-2 y FFT DIF-2 son casi insensibles al tamaño de N , lo cual es muy conveniente para el procesamiento de secuencias largas.

Para determinar el total de ciclos necesarios para el cálculo de una transformada de N muestras, se debe tomar en cuenta que rigurosamente se necesitan 2 ciclos (uno para lectura a memoria y otro para escritura) en la ejecución de cada *butterfly*, por lo que al final esto es igual a:

$$ciclos_{FFT} = 2N + N \log_2(N), \quad (5.28)$$

donde el primer término corresponde a las fases de almacenamiento y descarga de datos y el segundo al procesamiento en sí de la FFT.

Para evaluar a nivel funcional las arquitecturas de los procesadores FFT, se dispuso de la función intrínseca de Matlab $fft(x, N)$ como referencia para comparar los resultados obtenidos con el *hardware*. Por principios de cuenta, se generan y se cuantizan con $Q_W = 16$ bits, las secuencias discretas de 64 muestras* de tres señales representativas: un impulso unitario, una constante y una cosenoidal; todas con un rango dinámico de ± 1 . Luego se calcula la FFT con los procesadores diseñados y con Matlab. Los resultados comparativos de este proceso se pueden observar en las gráficas de las figuras 5.43, 5.44 y 5.45. Visualmente, en los tres casos no se observan disparidades entre las señales de salida en punto flotante y su contraparte de punto fijo, que indiquen un mal funcionamiento de las arquitecturas para la FFT.

*Se eligió este tamaño de FFT para una mejor legibilidad en las señales de las figuras 5.43–5.45

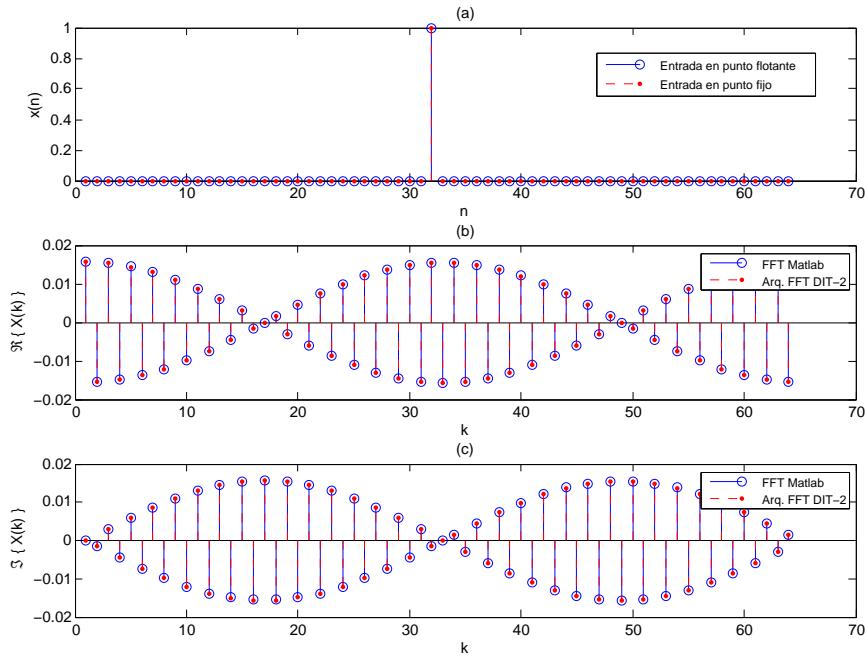


Figura 5.43: Comparación de la FFT de una señal impulso unitario calculada con el procesador FFT DIT-2 *versus* Matlab ; b) Parte real, c) Parte imaginaria.

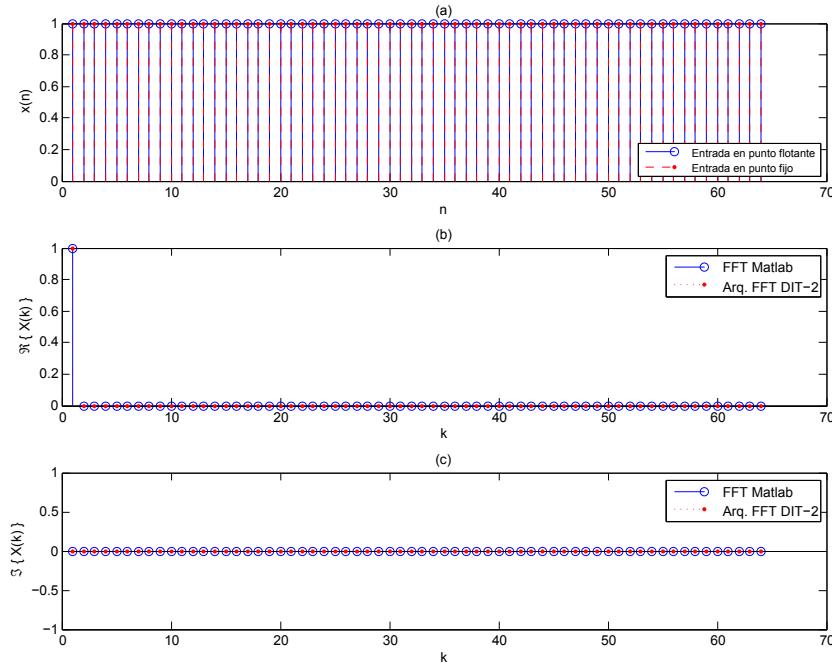


Figura 5.44: Comparación de la FFT de una señal de magnitud unitaria constante calculada con el procesador FFT DIT-2 *versus* Matlab ; b) Parte real, c) Parte imaginaria.

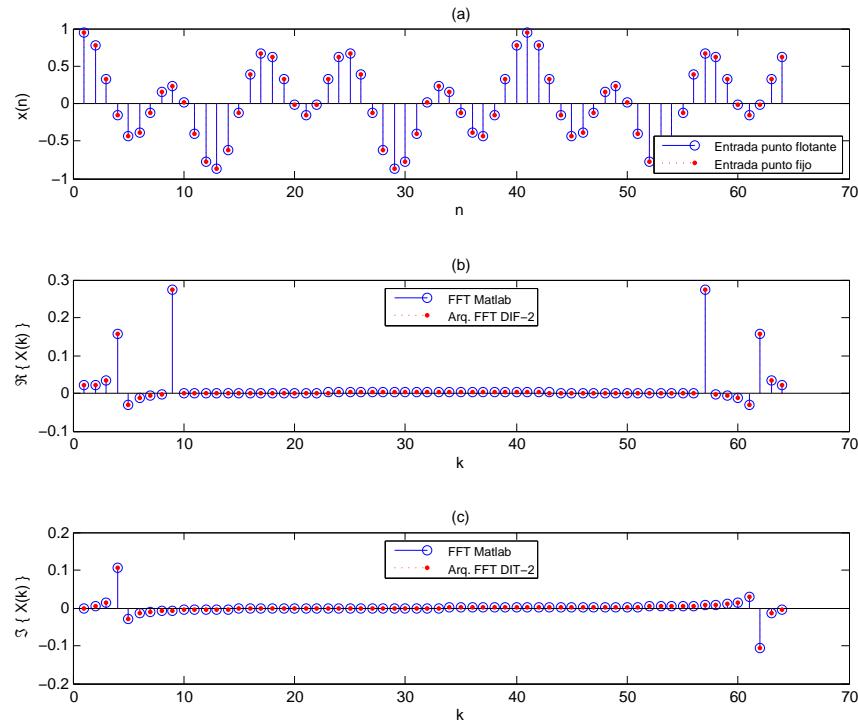


Figura 5.45: FFT de la señal $x(n) = 0.4 \cos(0.1\pi n) + 0.55 \cos(0.25\pi n)$ obtenida con el procesador FFT DIF–2 versus Matlab ; b) Parte real, c) Parte imaginaria.

Para determinar el grado de fidelidad de las salidas de los procesadores FFT, se precisa calcular el SQNR para diferentes valores de parametrización en las arquitecturas. Inicialmente, se define $N = 1024$ muestras de $Q_W = 16$ bits para la arquitectura FFT

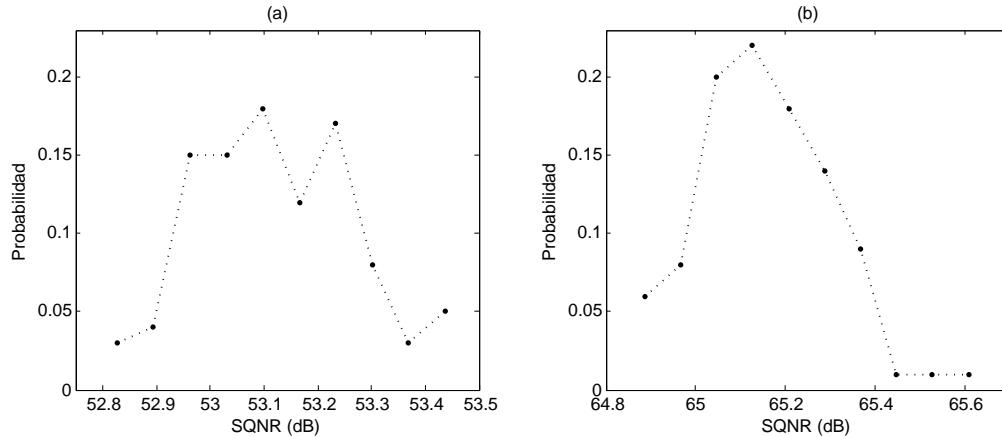


Figura 5.46: Funciones de densidad de probabilidad del SQNR en los procesadores FFT con $N = 1024$; a) Para arquitectura FFT DIF–2 con $Q_W = 16$ bits, b) Para arquitectura FFT DIT–2 con $Q_W = 18$ bits.

DIF–2 y $Q_W = 18$ bits para la FFT DIT–2. Despues, para encontrar las PDFs del SQNR en ambas arquitecturas; se ejecuta un conjunto de 100 transformadas a señales de entrada generadas de manera aleatoria en cada realización. En las gráficas de la figura 5.46 quedan de manifiesto los resultados de esta prueba. Se puede notar que el rango del SQNR en las dos arquitecturas FFT es menor a 1 dB, lo que asegura que la calidad de las señales de salidas en los procesadores FFT es consistente independientemente del tipo de señal de entrada.

En forma similar, en la figura 5.47 se explora la variación del SQNR de la arquitectura FFT DIT–2 ante diferentes anchos de palabra y manteniendo $N = 1024$ muestras. La gráfica muestra un comportamiento monotónico ascendente del SQNR a una razón de aproximadamente 6 dB por cada bit adicional en Q_W , lo cual es congruente con lo que marca la teoría de cuantización.

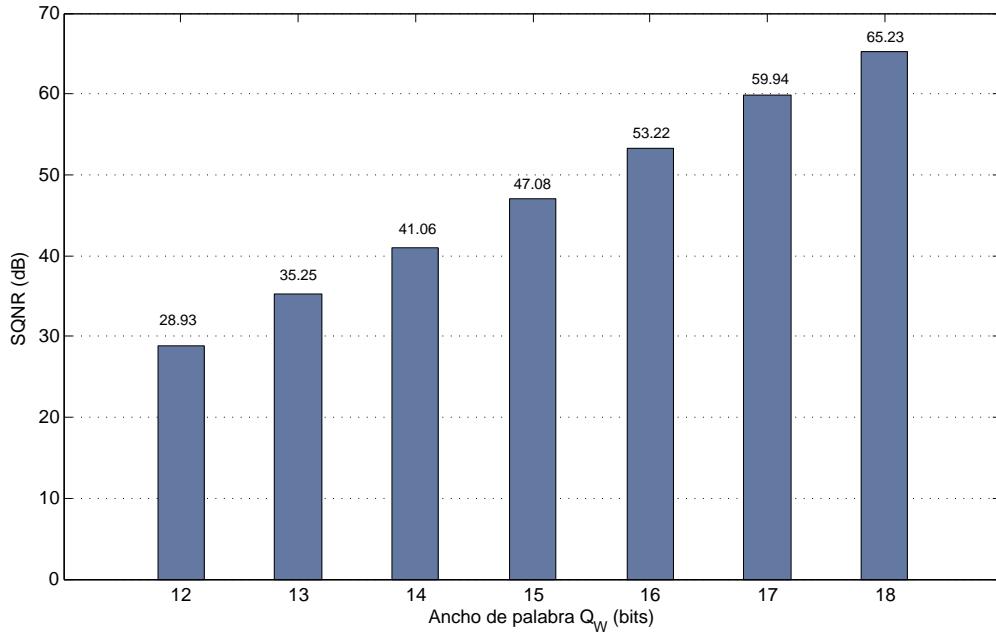


Figura 5.47: Comportamiento del SQNR en la arquitectura FFT DIT–2 de $N = 1024$ muestras ante diferentes anchos de palabra.

Para finalizar, la figura 5.48 exhibe los valores del SQNR en la arquitectura FFT DIF–2 para diversos tamaños de transformada y manteniendo el ancho de palabra fijo a 16 bits en un caso y 18 en el otro. Así, se puede notar que el SQNR disminuye alrededor de 3 dB cada vez que se duplica N . Esta variación prevalece a pesar de que exista una modificación en Q_W .

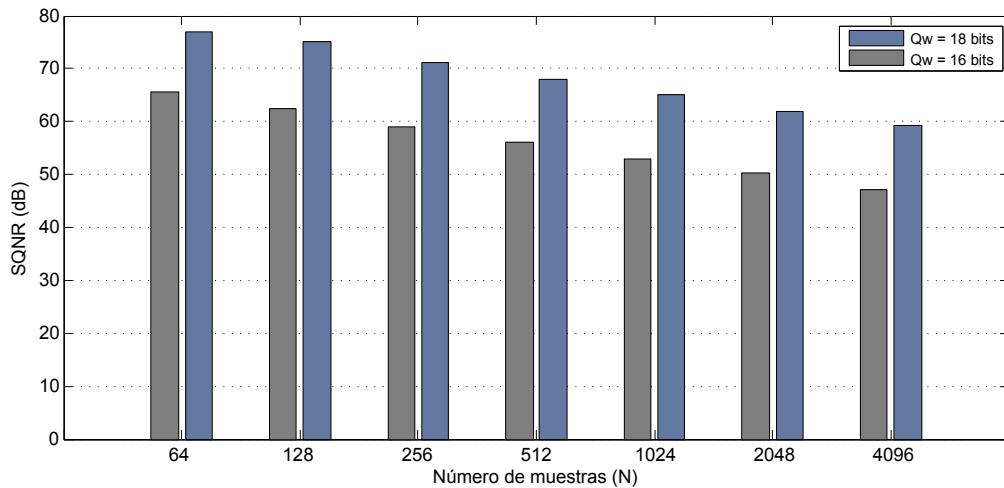


Figura 5.48: Comportamiento del SQNR en la arquitectura FFT DIF–2 para diferentes números de muestras y con anchura de palabra constante.

5.5. Correlacionador en el dominio de la frecuencia (FDCP)

Las ventajas del cálculo de la correlación en el dominio de la frecuencia eran conocidas desde tiempo atrás, pero nunca pudieron aprovecharse a causa de la inexistencia de un método eficiente para el cálculo de la DFT, lo cual evitaba una reducción en la complejidad computacional $O(N^2)$ presente en la correlación temporal. No fue sino hasta la aparición de los algoritmos de la FFT lo que hizo que la correlación en el dominio de la frecuencia fuera una herramienta atractiva.

Por otro lado, el mapeo a *hardware* de la expresión (3.83) de la correlación en dominio de la frecuencia, implica la ejecución de dos FFTs, una IFFT y un producto de Hadamard. La solución tradicional a lo anterior puede verse en la figura 5.49, donde con la ayuda de tres procesadores FFT se ejecutan las operaciones pertinentes y dos módulos de reordenamientos se encargan de restaurar la secuencia correcta en los datos, todo esto independientemente del tipo de radix que se use.

Sin embargo, los módulos de reordenamiento significan RAM y tiempo de procesamiento adicional en el diseño de una arquitectura para el correlacionador. Afortunadamente se puede evitar estas etapas extras de reordenamiento —inherentes en cada operación FFT— empleando el análisis presentado en [76] y que se expone en la siguiente subsección.

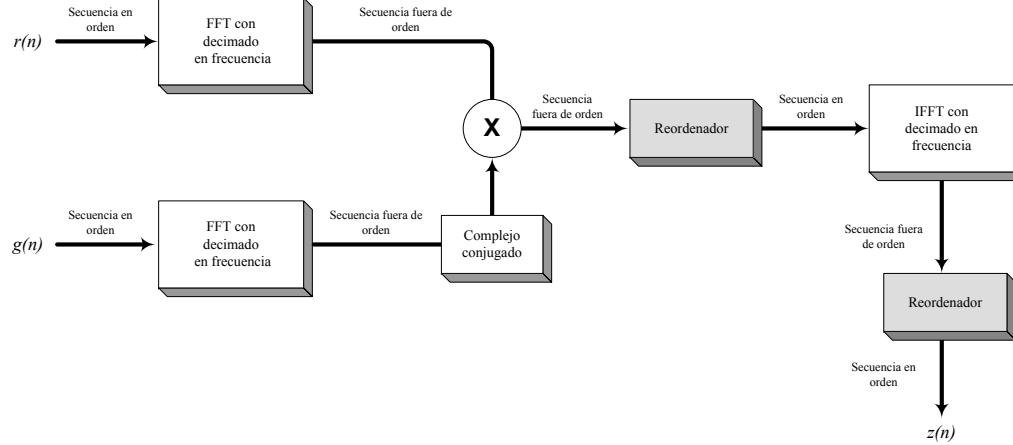


Figura 5.49: Diagrama a bloques del flujo de datos en la correlación en el dominio de la frecuencia.

5.5.1. Correlación sin permutaciones usando la FFT/IFFT radix–2

Retomando (3.83) y representándola en forma matricial se tiene

$$\mathbf{z}_{\mathcal{F}} = \mathbf{r}_{\mathcal{F}} \odot \mathbf{g}_{\mathcal{F}}^*, \quad (5.29)$$

donde \odot denota el producto de Hadamard, $\mathbf{z}_{\mathcal{F}}$, $\mathbf{r}_{\mathcal{F}}$ y $\mathbf{g}_{\mathcal{F}}$, representan las vectores con las transformadas de Fourier de $z(n)$, $r(n)$ y $g(n)$ respectivamente.

La correlación se obtiene transformando (5.29) al dominio temporal por medio de la IDFT,

$$\mathbf{z} = \mathcal{F}^{-1}\{\mathbf{z}_{\mathcal{F}}\}. \quad (5.30)$$

Sin embargo, si se sustituyen las operaciones DFTs con FFTs radix–2, se pueden hacer factorizaciones no recursivas en ellas donde se incluyan las operaciones de permutación que se llevan a cabo [77].

Así, la DFT $\mathbf{x}_{\mathcal{F}} \in \mathbb{C}$ de la secuencia $\mathbf{x} \in \mathbb{C}$ se define matricialmente con

$$\mathbf{x}_{\mathcal{F}} = \mathbf{W} \mathbf{x}, \quad (5.31)$$

donde \mathbf{W} es la matriz de transformación con los elementos dados por (3.69).

Sí $N = 2^t$, donde $t \in \mathbb{Z}$, entonces (5.31) puede calcularse con la FFT DIT–2, i.e.

$$\mathbf{x}_{\mathcal{F}} = \mathbf{F} \mathbf{x}, \quad (5.32)$$

donde $\mathbf{F} \in \mathbb{C}^{N \times N}$ corresponde a la matriz de transformación para la FFT DIT–2, la cual se representa concisamente por la siguiente factorización:

$$\mathbf{F} = \mathbf{A}_q \mathbf{A}_{q-1} \cdots \mathbf{A}_1 \mathbf{P}_N^T = \mathbf{D} \mathbf{P}_N^T, \quad q = 1, 2, \dots, t, \quad (5.33)$$

donde

$$\mathbf{A} = \mathbf{I}_s \otimes \mathbf{B}_L, \quad L = 2^q, \quad s = \frac{N}{L},$$

$$\mathbf{B}_L = \begin{bmatrix} \mathbf{I}_{L_0} & \Omega_{L_0} \\ \mathbf{I}_{L_0} & -\Omega_{L_0} \end{bmatrix}, \quad L_0 = \frac{L}{2}, \quad \Omega_{L_0} = \text{diag}(1, W_L^1, \dots, W_L^{L_0-1}),$$

con \otimes representando el producto Kronecker.

De igual modo, la permutación P_N esta dada por

$$\mathbf{P}_N = \mathbf{R}_q \mathbf{R}_{q-1} \cdots \mathbf{R}_1, \quad q = 1, 2, \dots, t, \quad (5.34)$$

donde $\mathbf{R}_q = \Pi_{2^{t-q}}$ y la expresión $\Pi_{2^{t-q}}$ se obtiene permutando las columnas de la matriz identidad de tamaño $2^q \times 2^q$ acorde al vector $\mathbf{v} \in \mathbb{Z}^N$, i.e.

$$\Pi_{2^q} = \mathbf{I}_{2^q}(0 : 2^q - 1, \mathbf{v}) \quad (5.35)$$

con $\mathbf{v} = [par(0 : 2^q - 1) \quad impar(0 : 2^q - 1)]$.

Se puede reformular la factorización en (5.33) para el caso de FFT DIT-2, tomando su transpuesta y considerando el hecho de que \mathbf{F} es simétrica.

$$\mathbf{F} = \mathbf{P}_N \mathbf{A}_q^T \mathbf{A}_{q-1}^T \cdots \mathbf{A}_1^T = \mathbf{P}_N \mathbf{D}^T. \quad (5.36)$$

Del mismo modo, se puede obtener la factorización para la IFFT DIT-2 de (5.33) reemplazando cada referencia a W_L con su complejo conjugado \bar{W}_L , con lo que

$$\mathbf{F}^{-1} = \frac{1}{N} \bar{\mathbf{A}}_q \bar{\mathbf{A}}_{q-1} \cdots \bar{\mathbf{A}}_1 \mathbf{P}_N = \frac{1}{N} \bar{\mathbf{D}} \mathbf{P}_N^T. \quad (5.37)$$

Luego entonces el producto de Hadamard de (5.29), expresado en términos de (5.36) se representa como,

$$\mathbf{z}_{\mathcal{F}} = (\mathbf{F} \mathbf{r}) \odot (\mathbf{F} \mathbf{g})^* = \mathbf{P}_N ((\mathbf{D}^T \mathbf{r}) \odot (\mathbf{D}^T \mathbf{g})^*). \quad (5.38)$$

Finalmente, la correlación en el dominio de la frecuencia definida en (5.30) se obtiene aplicando (5.37) en (5.38), i.e.

$$\begin{aligned} \mathbf{z} &= \mathbf{F}^{-1} \mathbf{z}_{\mathcal{F}} = \frac{1}{N} \bar{\mathbf{D}} \mathbf{P}_N^T \left[\mathbf{P}_N ((\mathbf{D}^T \mathbf{r}) \odot (\mathbf{D}^T \mathbf{g})^*) \right] \\ \mathbf{z} &= \frac{1}{N} \bar{\mathbf{D}} \left[(\mathbf{D}^T \mathbf{r}) \odot (\mathbf{D}^T \mathbf{g})^* \right]. \end{aligned} \quad (5.39)$$

Se puede apreciar en esta última ecuación que las operaciones referentes a las permutaciones se han cancelado a través de una juiciosa elección en el tipo de decimado que se debe aplicar en cada operación FFT (IFFT). Esto se puede visualizar claramente en el diagrama a bloques de la figura 5.50, mismo que se empleará como referencia para el diseño de la arquitectura del correlacionador

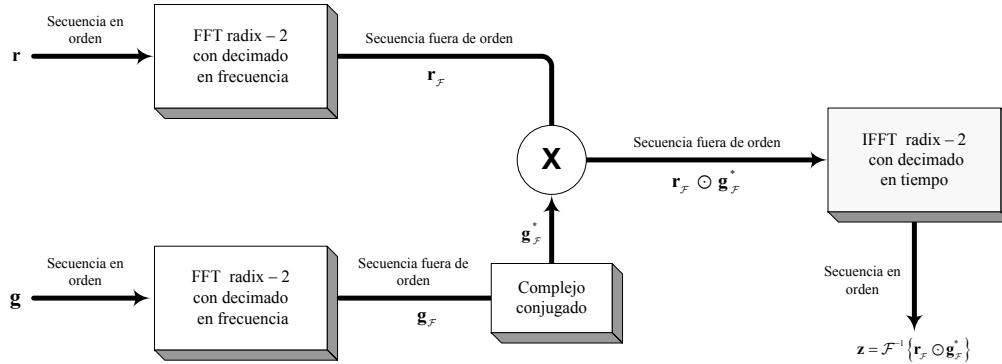


Figura 5.50: Diagrama a bloques de la correlación en el dominio de la frecuencia sin permutaciones usando FFTs/IFFT con radix-2.

5.5.2. Arquitectura de *hardware* y funcionamiento

La eficiencia en el consumo de *hardware* fue la principal directriz que prevaleció en el diseño de la arquitectura para la correlación descrita en 5.39. Como se puede apreciar en la figura 5.51, el sistema dispone de sólo un procesador FFT/IFFT reconfigurable (RIFP) que se reusa tres veces durante los cálculos. Adicionalmente, un módulo para el producto Hadamard, dos memorias RAMs, una unidad de control general (CGCU) y bloque para obtener el complejo conjugado de una señal, terminan de conformar el procesador para la correlación en el dominio de la frecuencia, referenciado en lo subsecuente como (FDCP) [76].

De manera general, su operación es como sigue. En el instante que se activa la señal *start_fdcp*, la secuencia de datos r se lee a través del bus *IN_FDCP* y se almacena en RAM1. Concluido lo anterior, la CGCU configura al RIFP para funcionar en modo FFT DIF-2 e inicia el cálculo de la FFT con los datos de la RAM1. Al mismo tiempo, la secuencia g se toma del *IN_FDCP* y se escribe pero ahora en RAM2. Toda vez que se concluye la ejecución de la FFT de r , el CGCU conmuta el flujo de datos del procesador hacia la RAM2 e inicia el cálculo de la FFT DIF-2 de su contenido. Por lo tanto, ya que se usa la estrategia de direccionamiento *in-place*, eventualmente las dos RAMs tendrán los resultados de cada transformación (r_f y g_f).

Posteriormente, se extraen los datos de la RAM2, se obtiene su complejo conjugado y se multiplican elemento a elemento con los datos de la RAM1; los resultados son escritos en RAM2. Luego, el CGCU reconfigura el modo de operación del procesador para ejecutar la IFFT DIT-2 con los datos almacenados en la RAM2. Para concluir, se activa la señal *done_fdcp* y los datos son leídos de la RAM2 y enviados al bus de salida *OUT_FDCP* y el correlacionador queda listo para otro procesamiento.

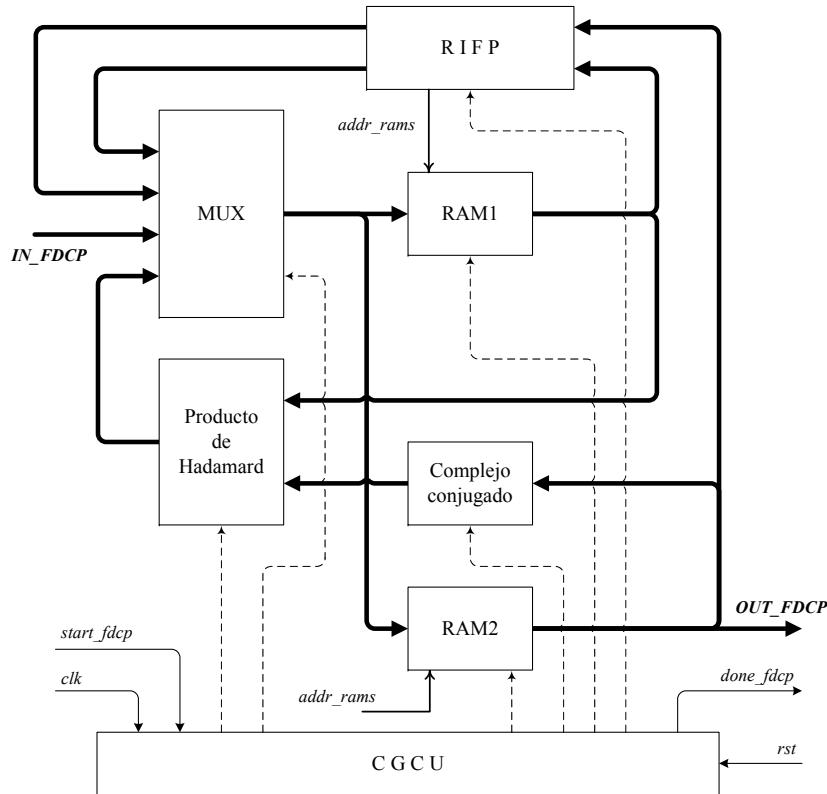


Figura 5.51: Arquitectura simplificada del correlacionador en el dominio de la frecuencia.

5.5.3. Control general del correlacionador (CGCU)

Este módulo se encarga de la temporización de los eventos y la señalización en la arquitectura del correlacionador. El CGCU se modeló como una FSM cuyos estados se muestran en la figura 5.52.

IDLE	LOAD_RAM1 (N ciclos)	CALC_FFT_RAM1 ($N \log_2 N$ ciclos)	--	--	--	LOAD_RAM1 (N ciclos)
	LOAD_RAM2 (N ciclos)	--	CALC_FFT_RAM2 ($N \log_2 N$ ciclos)	CALC_HAD (N ciclos)	CALC_IFFT_RAM2 ($N \log_2 N$ ciclos)	UNLOAD_RAM2 (N ciclos)

Figura 5.52: Estados de la CGCU y su duración.

Vale la pena resaltar, que durante el estado **CALC_FFT_RAM1**, se llena simultáneamente la RAM2 con la segunda secuencia, ahorrando N ciclos de procesamiento. De igual manera, otra correlación puede iniciar durante el estado **UNLOAD_RAM2**.

La CGCU condiciona el direccionamiento de las memorias RAMs durante la carga y descarga de las secuencias de datos, además cuando se calcula el producto de Hadamard. Pero, en todas aquellas en las que se involucra operaciones FFT/IFFT, el control cede el

direccionamiento de una o ambas RAMs al generador de direcciones del RIFP.

5.5.4. Procesador FFT/IFFT reconfigurable con decimado mixto (RIFP)

Es el módulo principal del FDCP, su diseño es basado en memoria con esquema de direccionamiento *in-place*, con lo que se logra eficiencia en el consumo de memoria aún para secuencias de datos muy grandes. El RIFP es capaz de calcular en punto fijo, la FFT DIF-2 escalada o la IFFT DIT-2 sin escalar, de un bloque de datos de longitud N . El RIFP está conformado de un *butterfly* radix-2 con decimado reconfigurable (RBE), un controlador FFT/IFFT (IFC), una ROM (igual a la descrita en 5.4.1.4) y un generador de direcciones (AGU_RIFP), todos interconectados en una sola arquitectura reconfigurable (figura 5.53) capaz de reproducir la funcionalidad de los procesadores FFT diseñados en 5.4.1 y 5.4.2, pero sin la ayuda de algún módulo de reordenamiento.

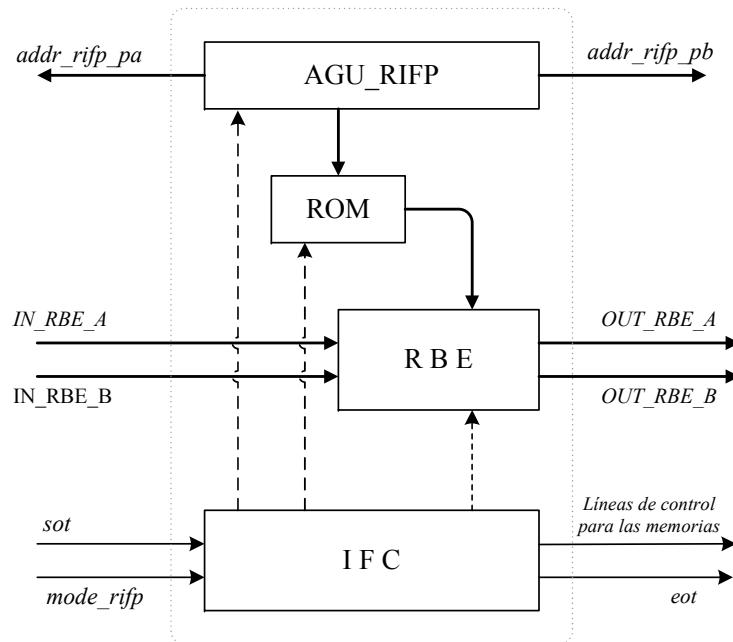


Figura 5.53: Arquitectura del RIFP.

5.5.4.1. *Butterfly* radix-2 con decimado reconfigurable (RBE)

Es la unidad de procesamiento del RIFP, constituida por un multiplicador y tres sumadores, dispuestos como se muestra en la figura 5.54. El RBE posee la capacidad de modificar el flujo de datos dentro de su arquitectura con el fin de ejecutar los *butterflies* DIF-2 o DIT-2, dependiendo del valor presente en la terminal *mode_rifp*. Asimismo, al igual que sus antecesores, procesa dos datos de entrada en paralelo y genera dos salidas escaladas en $\frac{1}{2}$ para generar un escalado uniforme total de $\frac{1}{N}$.

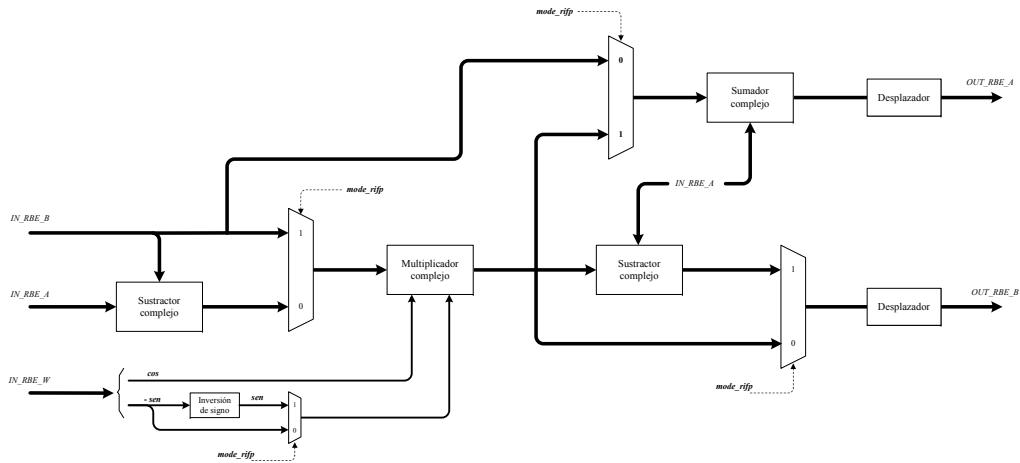


Figura 5.54: Arquitectura del *butterfly* radix–2 con decimado reconfigurable.

5.5.4.2. Controlador FFT/IFFT (IFC)

Una FSM con cuatro estados se usa para secuenciar todas las tareas del RIFP. Como se aprecia en la figura 5.55, en ninguno de los estados se define alguna tarea de carga o descarga de datos, ya que esto le corresponde al CGCU. Por lo que, el IFC asume que los datos que va a procesar el RIFP se han almacenado previamente en memoria. El IFC inicia su funcionamiento cuando la señal *sot* = 1. Luego, durante $N \log_2(N)$ ciclos habilita al procesador para leer, ejecutar *butterflies* y guardar *in-place* los resultados. Concluido esto, la bandera *eot* se activa para indicar el fin de la transformación.

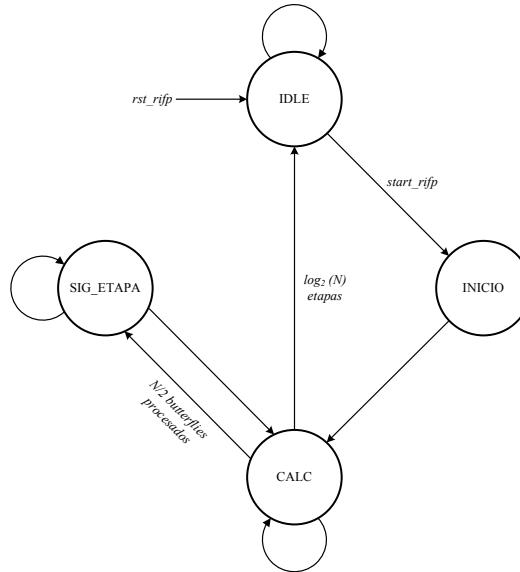


Figura 5.55: Máquina de estados finitos del IFC.

5.5.4.3. Generador de direcciones del RIFP (AGU_RIFP)

Para el diseño de este módulo de nueva cuenta hay que recalcar la flexibilidad del esquema de direccionamiento propuesto en 5.4.1.6, ya que permite integrar en una sola AGU; el direccionamiento para todas las memorias de la arquitectura del FDCP y para ambos modos de operación de RIFP (FFT DIF-2 o IFFT DIT-2). Retomando la arquitectura mostrada en 5.37, el único cambio consiste en sustituir los bloques de la parte sombreada con los necesarios para llevar a cabo las operaciones relativas al cálculo de $temp$. Si $mode_rifp = 0$ entonces $temp = etapa_act - 1$ de lo contrario su valor será $\log_2(N) - etapa_act$. La modificación final se muestra en la figura 5.56.

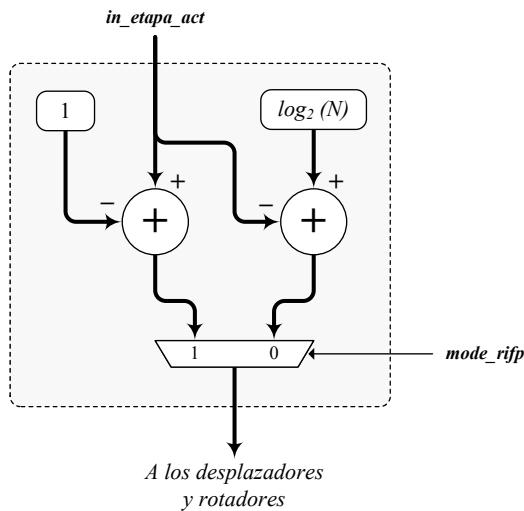


Figura 5.56: *Hardware* para el cálculo de $temp$ en el AGU_RIFP.

5.5.4.4. Memorias RAMs

Una estrategia de almacenamiento *ping-pong* con direccionamiento *in-place* se desarrolló con las memorias RAM1 y RAM2, con el objetivo de que mientras se esté procesando el contenido de una, la otra puede ser llenada simultáneamente con datos diferentes y viceversa. Luego entonces, es posible ejecutar en forma paralela algunas tareas en el FDCP, sin el riesgo de pérdida de datos. Además, ya que cada RAM de doble puerto tiene una profundidad de N localidades de memoria, el total requerido dentro de la arquitectura del FDCP es de $2N$.

5.5.5. Resultados de implementación y simulación

Se describió a nivel RTL usando Verilog HDL, una arquitectura parametrizable tanto en el número de muestras como en la anchura de palabra (N y Q_W) para el FDCP. Secuencias de entrada con longitudes de 512, 1024, 2048 y 4096 muestras y con anchos de palabra de 18, 20, 22 y 24 bits en el FDCP se sintetizaron en los FPGAs Cyclone

Longitud de la correlación (N)	512			1024		
	18	20	22	18	20	22
Frecuencia (MHz)	44.29	44.22	41.74	43.91	42.14	40.72
Elementos lógicos (LEs)	1349 (4%)	1634 (5%)	1784 (5%)	1422 (4%)	1701 (5%)	1838 (6%)
Registros	93 (<1%)	93 (<1%)	93 (<1%)	99 (<1%)	99 (<1%)	99 (<1%)
Memoria (Kbits)	46.08 (2%)	59.9 (2%)	59.9 (12%)	101.37 (21%)	115.2 (24%)	124.41 (26%)
BRAM	10 (10%)	13 (12%)	13 (12%)	22 (21%)	25 (26%)	27 (26%)
Multiplicadores (9 bits)	32 (46%)	56 (80%)	56 (80%)	32 (46%)	56 (80%)	56 (80%)

Tabla 5.10: Resultados de síntesis de FDCPs para secuencias cortas en el FPGA Cyclone II EP2C3F672C6 de Altera.

II EP2C3F672C6 y Xilinx Virtex II Pro 2VP70. En todos se usó el modo de síntesis por defecto y no se estableció alguna restricción de usuario en Quartus II v9.1 y en ISE v10.1. Para asegurar la independencia en la descripción, no se empleó ningún componente proveniente de las librerías *core-generator* o *megafunction*.

Los resultados de las síntesis se dividieron se la siguiente forma. Los correlacionadores de secuencias cortas (512 y 1024) y que se sintetizaron en el Cyclone II se muestran en la tabla 5.10. El resumen de síntesis en el Virtex II Pro de los FDCPs para secuencias largas (2048 y 4096) se presenta en la tabla 5.11.

Analizando tales tablas, resulta revelador notar que la frecuencia de operación decrece marginalmente a pesar que la longitud de la correlación se duplica. Mismo comportamiento se observa en la cantidad de *slices*/LEs, registros/*flip-flops* (FF) y multiplicadores dedicados que se necesitan. Por lo tanto, se puede establecer que la arquitectura para el FDCP es robusta en términos de la longitud de la correlación; lo cual lo hace bastante atractivo para el procesamiento de secuencias largas. Por otro lado, sí la anchura de palabra se incrementa, el retardo en la ruta de los datos (*path delay*) también crecerá, por lo que la frecuencia de operación se reducirá.

En lo que respecta al número total de ciclos para ejecutar una correlación de longitud N , es claro notar en la figura 5.52 que es aproximadamente de:

$$fdcp_ciclos = 4N + 3N \log_2(N). \quad (5.40)$$

Longitud de la correlación (N)	2048			4096		
	18	20	22	18	20	22
Frecuencia (MHz)	56.14	49.47	48.66	56.14	49.47	48.66
Slices	835 (2%)	891 (2%)	993 (3%)	860 (2%)	915 (2%)	1017 (3%)
Slices Flip Flops	101 (<1%)	100 (<1%)	100 (<1%)	107 (<1%)	106 (<1%)	106 (<1%)
LUTs	1285 (2%)	1697 (2%)	1891 (2%)	1631 (2%)	1743 (2%)	1937 (2%)
BRAM	10 (3%)	13 (3%)	13 (3%)	22 (6%)	25 (7%)	27 (8%)
Multiplicadores (18 bits)	8 (2%)	32 (9%)	32 (9%)	8 (2%)	32 (9%)	32 (9%)

Tabla 5.11: Resultados de síntesis de FDCPs para secuencias largas en el FPGA Virtex II Pro de Xilinx.

El ancho de palabra no está presente en (5.40) y por ende no afecta en el número de ciclos, pero de manera indirecta incrementa el tiempo necesario para el cálculo de la correlación (CT), tal como se visualiza en la tabla 5.12. En ella se resume el CT para varias implementaciones del FDCP en el FPGA Virtex II Pro. De tal forma, se observa que para secuencias cortas se tiene que $CT < 1$ ms. Para el caso de 2048 y 4096 muestras, el CT no supera los 2 ms y 3.4 ms, respectivamente.

Para comparar la eficiencia velocidad–consumo de *hardware* en los FDCPs, de nueva cuenta se obtiene su rendimiento por área (TP/área) y se muestra en la última columna de la tabla 5.12. Así por ejemplo, se puede notar que la arquitectura para un FDCP con $N = 2048/Q_W = 22$ bits tiene una eficiencia similar a un FDCP con $N = 4096/Q_W = 20$ bits. Cabe señalar que para esta comparativa no se toma en cuenta a las BRAMs.

Longitud de la correlación (N)	Ancho de palabra (Q_W bits)	Ciclos / correlación	CT (ms)	TP (MS/s)	TP/area (MS/s/slices)
512	18	15877	0.285	1.790	2.25
	20	<i>idem</i>	0.322	1.588	1.86
	22	<i>idem</i>	0.327	1.562	1.64
1024	18	34821	0.620	1.651	2
	20	<i>idem</i>	0.703	1.455	1.65
	22	<i>idem</i>	0.715	1.431	1.45
2048	18	75781	1.349	1.517	1.81
	20	<i>idem</i>	1.531	1.337	1.5
	22	<i>idem</i>	1.557	1.315	1.32
4096	18	163845	2.918	1.403	1.63
	20	<i>idem</i>	3.311	1.236	1.35
	22	<i>idem</i>	3.367	1.216	1.19

Tabla 5.12: Rendimiento de la arquitectura del FDCP sintetizada en el FPGA Virtex II Pro de Xilinx para diversas longitudes de correlación y anchos de palabras.

Concerniente al SQNR presente en el correlacionador, la figura 5.57 muestra el comportamiento de esta métrica con respecto N y Q_W . De tal forma, se observa que ha medida que se aumenta la longitud de la correlación entonces el SQNR se va degradando como consecuencia de que el incremento en la cantidad de etapas en las FFTs desemboca en un mayor acumulación de error. Entre tanto, como lo dicta la teoría de cuantización, se logra una ganancia de 6 dB en el SQNR por cada bit adicional en el ancho de la palabra. También se puede establecer la relación que existe entre el SQNR y la frecuencia de operación, p.ej., acorde a lo expuesto en la tabla 5.12, con dos bits adicionales en la profundidad de palabra de un FDCP de 2048 muestras; se logrará una ganancia de 12 dB en el SQNR a costa de un decremento de alrededor del 11 % en su frecuencia de operación.

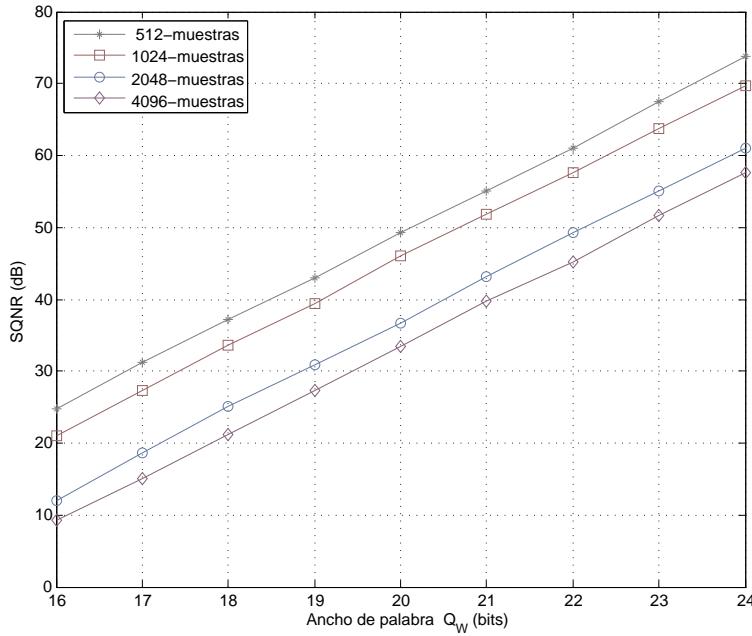


Figura 5.57: Gráfica de SQNR *versus* ancho de palabra en la arquitectura del FCDP para distintas longitudes de correlación.

Conclusiones del capítulo

En la parte inicial de este capítulo se presentaron tres arquitecturas para estimadores de canal para DDST que operan bajo condiciones de sincronía perfecta y ausencia de desplazamiento de CD. Hasta donde se tiene conocimiento, las tres representan las primeras implementaciones reportadas en la literatura bajo el paradigma de diseño *full-hardware*. Estos diseños presentan excelentes rendimientos y consumo reducido de recursos del FPGA. Además, la mayoría de sus módulos internos pueden modificarse sin mucha dificultad para procesar canales de diferentes tamaños.

Es preciso señalar que, a pesar de que las tres arquitecturas cumplen cabalmente con la tareas de estimación, su uso final dentro del receptor DDST es diferente. Por consiguiente, podemos señalar que el estimador CEOF está pensado para usarse en aquellos bloques que reciben y procesan directamente los datos de la secuencia de entrada. Entre tanto, la arquitectura CEDAM tiene su razón de ser en los bloques intermedios, donde la secuencia ya ha sufrido un procesamiento y los resultados se han almacenado en algunos de los *buffers* de memoria. La arquitectura sistólica, a pesar de ser la que menor rendimiento experimentó, tiene su utilidad en los algoritmos donde la MVM sea un recurrente en ellos.

Igualmente, se presenta por primera vez en la literatura el desarrollo de un estimador de canal para DDST de tipo *full-hardware* que opera en condiciones de falta de sincronía entre transmisor-receptor y en presencia de un desplazamiento de CD desconocido. Di-

cho estimador, denominado ECEO, se diseña a partir de la arquitectura CEOF y por lo tanto hereda sus características de alto desempeño. De la misma forma, se desarrolla una arquitectura novedosa de alto desempeño para calcular el cuadrado de la norma Euclídea de un vector complejo. También se demuestra que con un diseño adecuado de la arquitectura, a pesar de estar usando aritmética de punto fijo, es posible prescindir de la precisión completa la mayoría operaciones matemáticas en aras de un menor consumo de *hardware* y aun así lograr un rendimiento MSE y SNQR objetivo en el estimado del canal.

Por otro lado, también se presentaron dos arquitecturas basadas en memoria para la FFT radix-2, una para cada tipo de decimado. A lo largo de su desarrollo, se aplicaron mejoras en su diseño con el fin de reducir en la medida posible la complejidad en cada uno de sus módulos internos sin tener que sacrificar su velocidad. Las arquitecturas finales cumplen con el objetivo de mantener un buen balance entre velocidad y consumo de *hardware*. Ya que como quedó constancia, tanto la frecuencia de operación como el área de la arquitectura solo se ven afectadas marginalmente cuando se duplica el tamaño de la transformada.

Además, únicamente aplicando cambios mínimos en el diseño de los módulos internos de las arquitecturas FFT DIT-2 y FFT DIF-2, la mayoría de ellos se reutilizó para desarrollar un correlacionador en el dominio de la frecuencia, el cual hereda las ventajas de rendimiento y eficiencia de *hardware* no obstante la longitud que tengan las secuencias a procesar.

Para finalizar, derivado de los resultados obtenidos en los desarrollos de las arquitecturas presentadas en este capítulo, se publicaron los siguientes artículos:

- Romero-Aguirre, E.; Parra-Michel, R.; Longoria-Gandara, O.; Aguirre-Hernández, M., “A Hardware-Efficient Frequency Domain Correlator Architecture for Acquisition Stage in GPS,” Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on , vol. 2010, pp. 412-417, Dec. 2010.
- Romero-Aguirre, E.; Parra-Michel R.; Carrasco-Alvarez, R.; Orozco-Lugo, A.G., “Full-Hardware Architectures for Data-Dependent Superimposed Training Channel Estimation,” Signal Processing Systems (SiPS), 2011 IEEE Workshop on, vol. 2011, pp. 49-54, Oct. 2011.
- Romero-Aguirre, E.; Parra-Michel R.; Carrasco-Alvarez, R.; Orozco-Lugo, A.G., “Architecture Based on Array Processors for Data-Dependent Superimposed Training Channel Estimation,” Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on, vol. 2011, pp. 303-308, Dec. 2011.
- Romero-Aguirre, E.; Carrasco-Alvarez, R.; Parra-Michel, R.; Orozco-Lugo, A.; Mondragón-Torres, A., “Full-hardware architectures for data-dependent superimposed training channel estimation (extended version),” Journal of Signal Processing Systems, pp. 1-19, 10.1007/s11265-012-0706-2, Springer.

Capítulo 6

Conclusiones

6.1. Conclusiones

El concepto de IT fue primero propuesto por Farhang en 1995 y han transcurridos casi 8 años desde que Orozco lo formalizó en [5]. Desde entonces ha sido motivo de investigaciones intensas a nivel mundial y por parte de esta tesis.

Con respecto a las técnicas IT se concluye que:

- Ante la exigencia continua de sistemas de comunicaciones inalámbricos con mayores prestaciones, las técnicas de ST/DDST ofrecen una alternativa simple y eficiente en ancho de banda que hace posible tasas de transferencias de datos mas elevadas y con la ventaja adicional de poder enviar entrenamiento todo el tiempo.
- No obstante la existencia de trabajos recientes que ya consideran ST/DDST en sistemas OFDM, MIMO y OFDM/MIMO, un aspecto clave para su consolidación total consiste en fortalecer las propuestas encaminados en llevar las investigaciones más allá de las propuestas teóricas.
- Se espera que conforme vayan surgiendo más prototipos experimentales que permitan superar la barrera antes mencionada, más pronto estarán disponibles comercialmente los productos ligados a este concepto.

En lo que toca al diseño de arquitecturas de *hardware*, se exponen las siguientes conclusiones.

- No resulta suficiente analizar de manera individual un algoritmo específico, siempre hay que diseñar su arquitectura teniendo en mente su interacción con los bloques restantes que conforman el sistema.

- La estrategia obvia de usar precisión completa en las operaciones matemáticas siempre debe ser usada a discreción. De lo contrario, siempre queda el riesgo latente de que la arquitectura diseñada quede sobredimensionada en términos de rendimiento y que esto desembogue en consumos de área innecesarios
- Siempre que sea posible, en el diseño de una arquitectura no se debe interrumpir o retardar en exceso el flujo natural de los datos. Asimismo es recomendable emplear estrategias de almacenamiento *in-place*, introducir etapas de *pipeline* si la ruta crítica es muy larga y describir las memorias FIFO o LIFO a partir de BRAMs de doble puerto.
- Debido a que el desarrollo de una arquitectura digital es un problema multifactorial, es imposible dilucidar una función de costo cerrada que permita establecer si es óptima o no.
- La habilidad y criterio del arquitecto de *hardware* siempre juega un rol importante al diseñar el sistema en cuestión. Hasta el día de hoy, no existe herramienta de diseño alguna que tenga la suficiente “inteligencia” para superar el ingenio e intuición que solo la experiencia le da al arquitecto al momento de estructurar la estrategia de diseño que va a aplicar.

Referente a la investigación llevada a cabo y sus aportaciones, se puede concluir que:

- Las arquitecturas para procesadores FFT radix-2 que se diseñaron son totalmente parametrizables en la longitud de la secuencias de entrada y la longitud en bits para representar cada muestra. Su principal fortaleza radica en el hecho de que el consumo de *hardware* y su frecuencia de operación se mantienen casi constantes independientemente de la longitud de la secuencia a transformar. Esto las hace particularmente atractivas en el caso de procesar secuencias de entradas de gran longitud.
- Se planteó como a partir de un procesador FFT radix-2 con decimado mixto se desarrolló un correlacionador en el dominio de la frecuencia que no emplea etapas de reordenamiento alguna para procesar y generar los resultados en el orden correcto.
- En virtud de que el correlacionador hace uso de las arquitecturas FFT anteriores, también hereda la cualidad de procesar secuencias de gran longitud sin afectaciones mayores a su rendimiento y al área utilizada.
- Las tres arquitecturas (CEDAM, CEOF y ECEO) que se propusieron para estimación de canal bajo condiciones de sincronía perfecta y ausencia de desplazamiento de CD tienen en común la capacidad de desarrollar rendimientos respetables y consumos reducidos de recursos FPGA. De igual manera, la mayoría de sus

módulos internos pueden modificarse sin mayores complicaciones para procesar canales de diferente duración.

- Como quedó de manifiesto, los estimadores CEDAM, CEOF y SYSDCE cumplen cabalmente con la tareas de estimación de la media cíclica y el canal. Sin embargo, su uso final dentro del receptor es diferente. Así, el estimador CEOF está pensado para emplearse en aquellos módulos que reciben/procesan las secuencias de datos en las cercanías de las entradas del receptor y donde es prohibitivo interrumpir su flujo. La arquitectura CEDAM es conveniente cuando el bloque de datos a procesar ha sido previamente almacenado en alguna zona de memoria intermedia. De forma similar a la anterior, la arquitectura SYSDCE tiene su utilidad cuando la secuencia a procesar ha sido almacenada con antelación y además se necesiten operaciones MVM adicionales en el algoritmo.
- Los resultados comparativos de las implementaciones en tecnología ASIC de 90 nm de las arquitecturas CEDAM y CEOF, nos permiten asentar que este último estimador es el que presenta la mejor relación entre frecuencia de operación y consumo de área. Por tal motivo, fue la que se eligió como módulo atómico para el diseño de un estimador más sofisticado
- El desarrollo del estimador ECEO, el cual realiza la estimación del canal en condiciones de operación más realistas (falta de sincronía entre transmisor-receptor y desplazamiento desconocido de CD), es un claro ejemplo de como las técnicas de paralelismo, sistolización, encauzamiento y reusabilidad, pueden coexistir dentro de una arquitectura digital.
- Se demostró la validez de la propuesta de adaptación (inédita) del algoritmo de estimación de la media cíclica para su ejecución en un arreglo sistólico que realiza la operación de MVM. De la misma forma, se pudo corroborar la efectividad de las reformulaciones planteadas para el cálculo de la norma Euclíadiana.
- El rango de SNR en el cual se pretende operar el sistema de comunicación es la cota inferior de rendimiento SQNR promedio que las arquitecturas de estimadores de canal DDST deben cumplir. Esto abre la posibilidad de introducir precisión limitada en las operaciones y truncamientos en los resultados con la finalidad de reducir el consumo de área. Esto implica llevar el error de cuantización de las arquitecturas mencionadas a fronteras donde sea una fuente de ruido considerablemente con respecto al SNR.
- En todas las arquitecturas que se presentaron en este trabajo, más allá de las adaptaciones mayores propuestas para los algoritmos, existen varias pequeñas pero novedosas mejoras inmersas en su diseño, las cuales ya vistas en conjunto logran que las implementaciones tengan un buen balance entre rendimiento y consumo de área.

- Las arquitecturas *full-hardware* del transmisor y las de todos los estimadores de canal desarrolladas en este trabajo doctoral son inéditas en el estado del arte y las primeras implementaciones prácticas en presentar resultados de desempeño. Por lo que se espera que pueden utilizarse como referencia en prototipos futuros.
- Las altas frecuencias de operación (> 100 MHz), tasas de rendimiento elevadas (7.1 – 157.73 MS/s) y consumos reducidos de recursos de FPGA (silicio en los implementados en ASIC) que experimentan tanto el transmisor como los estimadores de canal, permiten establecer que el concepto de ST/DDST es una opción real que puede implantarse en los estándares de comunicación actuales y futuros.

6.2. Líneas de investigación futuras

Como se observó, el trabajo realizado durante esta tesis a pesar de ser extenso, no alcanza cubrir a la exploración y diseño de arquitecturas adicionales para algoritmos que atacan otras problemáticas existentes en un sistema de comunicación basado en ST/DDST. Por tal motivo y continuando con la directriz de implementación eficiente de algoritmos para estos sistemas, se propone lo siguiente:

- Diseño e implementación de arquitecturas eficientes para efectuar la sincronía de frecuencia en receptores con entrenamiento implícito.
- Desarrollar arquitecturas de alto rendimiento para los algoritmos dedicados a la estimación de canal variante en ST/DDST.
- Extrapolar lo expuesto en esta tesis con la finalidad de desarrollar una arquitectura para un sistema de comunicación MIMO con ST/DDST.
- Evaluar la pertinencia de un sistema de comunicación con ST/DDST desarrollado bajo el enfoque de NoC.

La primera línea está enfocada en evaluar la factibilidad práctica de los algoritmos propuestos en la literatura para estimar/compensar las variaciones de frecuencia en sistemas con IT. El segundo trabajo que se sugiere está estrechamente relacionado con un conjunto algoritmos novedosos, reportados en [33, 34], que solucionan la problemática de estimar un canal variante en el tiempo en un sistema ST/DDST. Hasta el momento, los resultados a nivel de simulación de tales algoritmos permiten asentar su superioridad al compararlos con otras propuestas del estado del arte. Por lo que, el siguiente paso es establecer su factibilidad práctica mediante su diseño eficiente en *hardware*. La justificación atrás de la tercera línea sugerida, tiene que ver con el protagonismo impuesto por los sistemas MIMO, que recientemente los han catapultado como uno de los adelantos de mayor relevancia en las comunicaciones modernas y con el potencial de resolver el cuello de botella en la capacidad de tráfico de las existentes y futuras redes inalámbricas. La cuarta propuesta permitirá contrastar las ventajas al diseñar la arquitectura y las

inherentes pérdidas en el rendimiento, que implica el usar redes de procesadores *on-chip* (comutada por circuitos o por paquetes) en un caso de estudio complejo como lo es un sistema de comunicación con ST/DDST.

Bibliografía

- [1] S. Haykin. *Handbook of Array Processing and Sensor Networks*.
- [2] B. Farhang-Boroujeny. Pilot-based channel identification: proposal for semi-blind identification of communication channels. *Electronics Letters*, 31(13):1044–1046, 22 June 1995.
- [3] F. Mazzenga. Channel estimation and equalization for m-qam transmission with a hidden pilot sequence. *IEEE Trans. Broadcast.*, 46(2):170–176, 2000.
- [4] G. Tong Zhou, M. Viberg, and T. McKelvey. Superimposed periodic pilots for blind channel estimation. In *Conference Record of the Thirty-Fifth Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 653–657 vol.1, 2001.
- [5] A.G. Orozco-Lugo, M.M. Lara, and D.C. McLernon. Channel estimation using implicit training. *IEEE Trans. Signal Process.*, 52(1):240–254, Jan 2004.
- [6] J.K. Tugnait and Weilin Luo. On channel estimation using superimposed training and first-order statistics. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '03)*, volume 4, pages IV–624–7 vol.4, 2003.
- [7] M. Ghogho. Channel and dc-offset estimation using data-dependent superimposed training. *Electronics Letters*, 41(22):1250–1252, 27 Oct. 2005.
- [8] E. Alameda-Hernandez, D. McLernon, A.G. Orozco-Lugo, and M. Ghogho. Synchronisation and dc-offset estimation for channel estimation using data-dependent superimposed training. In *European Signal Processing Conference (EUSIPCO 2005)*, 2005.
- [9] E. Alameda-Hernandez, D. C. McLernon, M. Ghogho, A. G. Orozco-Lugo, and M. Lara. Synchronisation for superimposed training based channel estimation. *Electronics Letters*, 41(9):565–567, 2005.
- [10] E. Alameda-Hernandez, D.C. McLernon, A.G. Orozco-Lugo, M.M. Lara, and M. Ghogho. Frame/training sequence synchronization and dc-offset removal for (data-dependent) superimposed training based channel estimation. *IEEE Trans. Signal Process.*, 55(6):2557–2569, June 2007.
- [11] S. M. A. Moosvi, D. C. McLernon, E. Alameda-Hernandez, and M. Ghogho. Block synchronisation for joint channel and dc-offset estimation using data-dependent superimposed training. In *Proc. 9th International Symposium on Signal Processing and Its Applications ISSPA 2007*, pages 1–4, 12–15 Feb. 2007.

- [12] D.H. Pham and J.H. Manton. Orthogonal superimposed training on linear precoding: a new affine precoder design. In *Proc. IEEE 6th Workshop on Signal Processing Advances in Wireless Communications*, pages 445–449, 2005.
- [13] Xiaohong Meng and J.K. Tugnait. Semi-blind channel estimation and detection using superimposed training. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '04)*, volume 4, pages iv–417–iv–420, 17–21 May 2004.
- [14] J.K. Tugnait and Xiaohong Meng. On superimposed training for channel estimation: performance analysis, training power allocation, and frame synchronization. *IEEE Trans. Signal Process.*, 54(2):752–765, Feb. 2006.
- [15] Sung-Yoon Jung and Dong-Jo Park. Linear mmse receiver using hidden training sequence in ds/cdma. *Electronics Letters*, 39(9):742–744, 2003.
- [16] Ning Chen and G.T. Zhou. Superimposed training for ofdm: a peak-to-average power ratio analysis. *IEEE Trans. Signal Process.*, 54(6):2277–2287, June 2006.
- [17] Daofeng Xu and Luxi Yang. Channel estimation for ofdm systems using superimposed training. In *Internet, 2005. The First IEEE and IFIP International Conference in Central Asia on*, page 5 pp., sept. 2005.
- [18] J.P. Nair and R.V. Raja Kumar. Channel estimation and equalization based on implicit training in ofdm systems. In *Wireless and Optical Communications Networks, 2006 IFIP International Conference on*, pages 5 pp. –5, 0-0 2006.
- [19] R. Dinis, N. Souto, J. Silva, A. Kumar, and A. Correia. Joint detection and channel estimation for ofdm signals with implicit pilots. In *Mobile and Wireless Communications Summit, 2007. 16th IST*, pages 1 –5, july 2007.
- [20] R. Dinis, N. Souto, J. Silva, A. Kumar, and A. Correia. On the use of implicit pilots for channel estimation with ofdm modulations. In *Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th*, pages 1077 –1081, 30 2007-oct. 3 2007.
- [21] Xiliang Luo and G.B. Giannakis. Low-complexity blind synchronization and demodulation for (ultra-)wideband multi-user ad hoc access. *IEEE Trans. Wireless Commun.*, 5(7):1930–1941, July 2006.
- [22] M. Ghogho, D. McLernon, E. Alameda-Hemandez, and A. Swami. Siso and mimo channel estimation and symbol detection using data-dependent superimposed training. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '05)*, volume 3, pages iii/461–iii/464, 18–23 March 2005.
- [23] Haidong Zhu, B. Farhang-Boroujeny, and C. Schlegel. Pilot embedding for joint channel estimation and data detection in mimo communication systems. *IEEE Commun. Lett.*, 7(1):30–32, Jan 2003.
- [24] R. Carrasco-Alvarez, R. Parra-Michel, A. G. Orozco-Lugo, and J. K. Tugnait. Enhanced channel estimation using superimposed training based on universal basis expansion. *IEEE Trans. Signal Process.*, 57(3):1217–1222, 2009.

- [25] A.G. Orozco-Lugo, G.M. Galvin-Tejada, M.M. Lara, and D.C. McLernon. A new approach to achieve multiple packet reception for ad hoc networks. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '04)*, volume 4, pages iv–429–iv–432, 17–21 May 2004.
- [26] S.M.A. Moosvi, D.C. McLernon, A.G. Orozco-Lugo, M.M. Lara, and E. Alameda-Hernandez. Improved carrier frequency offset estimation using data-dependent superimposed training. In *Proc. 4th International Conference on Electrical and Electronics Engineering ICEEE 2007*, pages 126–129, 5–7 Sept. 2007.
- [27] A.G. Orozco-Lugo, M.M. Lara, E. Alameda-Hernandez, S. Moosvi, and D.C. McLernon. Frequency offset estimation and compensation using superimposed training. In *Proc. 4th International Conference on Electrical and Electronics Engineering ICEEE 2007*, pages 118–121, 5–7 Sept. 2007.
- [28] Shuangchi He and J.K. Tugnait. On doubly selective channel estimation using superimposed training and discrete prolate spheroidal sequences. *Signal Processing, IEEE Transactions on*, 56(7):3214 –3228, july 2008.
- [29] Junruo Zhang and Y. Zakharov. Iterative b-spline estimator using superimposed training in doubly-selective fading channels. In *Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on*, pages 1795 –1799, nov. 2007.
- [30] J.K. Tugnait and Shuangchi He. Doubly-selective channel estimation using data-dependent superimposed training and exponential basis models. *Wireless Communications, IEEE Transactions on*, 6(11):3877 –3883, november 2007.
- [31] Yuanjie Li and Luxi Yang. Channel estimation and tracking using implicit training. In *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, volume 1, pages 72 – 75 Vol. 1, sept. 2004.
- [32] Mengxing Li, Weiming Zuo, and Zemin Liu. Time-selective mimo channel estimation based on implicit training. In *Intelligent Signal Processing and Communication Systems, 2007. ISPACS 2007. International Symposium on*, pages 384 –387, 28 2007-dec. 1 2007.
- [33] R. Carrasco-Alvarez, R. Parra-Michel, and A.G. Orozco-Lugo. Enhanced time-varying channel estimation based on two dimensional basis projection and self-interference suppression. In *Signal Processing Advances in Wireless Communications (SPAWC), 2010 IEEE Eleventh International Workshop on*, pages 1 –5, june 2010.
- [34] R. Carrasco-Alvarez, R. Parra-Michel, A.G. Orozco-Lugo, and J.K. Tugnait. Time-varying channel estimation using two-dimensional channel orthogonalization and superimposed training. *Signal Processing, IEEE Transactions on*, 60(8):4439 –4443, aug. 2012.
- [35] O. Longoria-Gandara, R. Parra-Michel, M. Bazdresch, and A.G. Orozco-Lugo. Iterative mean removal superimposed training for frequency selective channel estimation. In *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, pages 1 –5, oct. 2008.
- [36] O. Longoria-Gandara, R. Parra-Michel, M. Bazdresch, and A.G. Orozco-Lugo. Iterative mean removal superimposed training for siso and mimo channel estimation. *International Journal of Digital Multimedia Broadcasting*, vol. 2008, 2008.

- [37] M. Qaisrani and S. Lambotharan. Estimation of doubly selective mimo channels using superimposed training and turbo equalization. In *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, pages 1316 –1319, may 2008.
- [38] Weina Yuan and Pingzhi Fan. Implicit mimo channel estimation without dc-offset based on zcz training sequences. *Signal Processing Letters, IEEE*, 13(9):521 –524, sept. 2006.
- [39] Han Zhang, XianHua Dai, and DaRu Pan. Linearly time-varying channel estimation and training power allocation for ofdm/mimo systems using superimposed training. *SCIENCE CHINA Information Sciences*, 54:1456–1470, 2011. 10.1007/s11432-011-4241-8.
- [40] Toni Levanen, Jukka Talvitie, and Markku Renfors. Performance evaluation of time-multiplexed and data-dependent superimposed training based transmission with practical power amplifier model. *EURASIP Journal on Wireless Communications and Networking*, 2012:1–19, 2012. 10.1186/1687-1499-2012-49.
- [41] A. Goljahani, N. Benvenuto, S. Tomasin, and L. Vangelista. Superimposed sequence versus pilot aided channel estimations for next generation dvb-t systems. *Broadcasting, IEEE Transactions on*, 55(1):140 –144, march 2009.
- [42] V. Najera-Bello. Diseño y construcción de un sistema de comunicaciones digital basado en entrenamiento implícito. Master's thesis, CINVESTAV-IPN, 2011.
- [43] Fernando Martín del Campo, René Cumplido, Roberto Perez-Andrade, and A. G. Orozco-Lugo. A system on a programmable chip architecture for data-dependent superimposed training channel estimation. *International Journal of Reconfigurable Computing*, 2009, 2009.
- [44] Lang Tong, B.M. Sadler, and Min Dong. Pilot-assisted wireless transmissions: general model, design criteria, and signal processing. *IEEE Signal Process. Mag.*, 21(6):12–25, Nov. 2004.
- [45] J.-P. Javaudin, J.-P. Javaudin, D. Lacroix, and A. Rouxel. Pilot-aided channel estimation for ofdm/oqam. In D. Lacroix, editor, *Proc. VTC 2003-Spring Vehicular Technology Conference The 57th IEEE Semannual*, volume 3, pages 1581–1585 vol.3, 2003.
- [46] Ye Li. Pilot-symbol-aided channel estimation for ofdm in wireless systems. *IEEE Trans. Veh. Technol.*, 49(4):1207–1215, July 2000.
- [47] Xiaoli Ma, Xiaoli Ma, G.B. Giannakis, and S. Ohno. Optimal training for block transmissions over doubly selective wireless fading channels. *IEEE Trans. Signal Process.*, 51(5):1351–1366, 2003.
- [48] M. Ghogho, D. McLernon, E. Alameda-Hernandez, and A. Swami. Channel estimation and symbol detection for block transmission using data-dependent superimposed training. *IEEE Signal Process. Lett.*, 12(3):226–229, 2005.
- [49] D.C. McLernon, E. Alameda-Hernandez, A.G. Orozco-Lugo, and M.M. Lara. Performance of data-dependent superimposed training without cyclic prefix. *Electronics Letters*, 42(10):604 – 606, may 2006.
- [50] Weina Yuan, Ping Wang, and Pingzhi Fan. On the performance of data-dependent superimposed training without cyclic prefix for siso/mimo systems. *Journal of Electronics (China)*, 27:37–42, 2010.

- [51] T. Whitworth, M. Ghogho, and D. C. McLernon. Data identifiability for data-dependent superimposed training. In *Proc. IEEE Int. Conf. Communications ICC '07*, pages 2545–2550, 2007.
- [52] O. Longoria-Gandara, R. Parra-Michel, M. Bazdresch, and A.G. Orozco-Lugo. Mimo channel estimation via iterative hybrid superimposed training and pilot assisted transmission. *International Conference on Telecommunication Systems Modeling and Analysis (IC-TSM)*, vol. 2012, 2012.
- [53] B. Razavi. Design considerations for direct-conversion receivers. *IEEE Trans. Circuits Syst. II*, 44(6):428–435, June 1997.
- [54] P.J. Davis. *Circulant Matrices*. Chelsea Publishing Series. Chelsea, 1994.
- [55] E. Aboutanios. A modified dichotomous search frequency estimator. *IEEE Signal Process. Lett.*, 11(2):186–188, Feb. 2004.
- [56] Richard Bellman. *Introduction to Matrix Analysis (Classics in Applied Mathematics)*. Society for Industrial Mathematics, 2 edition, 1 1987.
- [57] D. Falconer, S.L. Ariyavisitakul, A. Benyamin-Seeyar, and B. Eidson. Frequency domain equalization for single-carrier broadband wireless systems. *IEEE Commun. Mag.*, 40(4):58–66, April 2002.
- [58] Harry L. Van Trees. *Detection, Estimation, and Modulation Theory, Part I*. Wiley-Interscience, 1 edition, 9 2001.
- [59] R. Parra-Michel, V.Y. Kontorovitch, and A.G. Orozco-Lugo. Simulation of wideband channels with non-separable scattering functions. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '02)*, volume 3, pages III–2829–III–2832, 13–17 May 2002.
- [60] A.G. Orozco-Lugo, R. Parra-Michel, D.C. McLernon, and V.Y. Kontorovitch. Enhancing the performance of the cr blind channel estimation algorithm using the karhunen-loeve expansion. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '02)*, volume 3, pages III–2653–III–2656, 13–17 May 2002.
- [61] D.J. DeFatta. *Digital Signal Processing: A System Design Approach*. World Publishing Corporation, 1988.
- [62] J.G. Proakis and D.G. Manolakis. *Digital signal processing*. Pearson Prentice Hall, 2007.
- [63] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. of Comp.*, 19(90):297–301, April 1965.
- [64] W. M. Gentleman and G. Sande. Fast fourier transforms for fun and profit. *AFIPS Proc., 1966 Fall Joint Computer Conf.*, 29:563–678, Washington, D. C.; Spartan 1966.
- [65] J.L. Pizano-Escalante. Diseño e implementación digital del filtro igualador de canal para el estándar 802.15.3c. Master's thesis, CINVESTAV-IPN, 2010.
- [66] R Carrasco-Alvarez. *Estimación de Canales Variantes en Tiempo de Banda Amplia*. PhD thesis, CINVESTAV-GDL, 2010.

- [67] S. Kung. *VLSI Array processors*. Prentice Hall, 1985.
- [68] N. Petkov. *Systolic Parallel Processing*. Elsevier Science Inc., New York, NY, USA, 1992.
- [69] I. Kuon and J. Rose. Measuring the gap between fpgas and asics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):203 –215, feb. 2007.
- [70] H. A. Sturges. The choice of a class interval. *Journal of the American Statistical Association*, 66(153):65–66, 1926.
- [71] David W. Scott. On optimal and data-based histograms. *Biometrika*, 66(3):605–610, 1979.
- [72] L.G. Johnson. Conflict free memory addressing for dedicated fft hardware. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 39(5):312 –316, may 1992.
- [73] J.H. Takala, T.S. Jarvinen, and H.T. Sorokin. Conflict-free parallel memory access scheme for fft processors. In *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, volume 4, pages IV–524 – IV–527 vol.4, may 2003.
- [74] C. Gonzalez-Concejero, V. Rodellar, A. Alvarez-Marquina, E. Icaya, and P. Gomez-Vilda. An fft/ifft design versus altera and xilinx cores. In *Reconfigurable Computing and FPGAs, 2008. ReConFig '08. International Conference on*, pages 337 –342, dec. 2008.
- [75] Yingning Peng Bin Zhou and David Hwang. Pipeline fft architectures optimized for fpgas. *International Journal of Reconfigurable*, 2009, 2009.
- [76] E. Romero-Aguirre, R. Parra-Michel, O. Longoria-Gandara, and M. Aguirre-Hernández. A hardware-efficient frequency domain correlator architecture for acquisition stage in gps. In *Reconfigurable Computing and FPGAs (ReConfig), 2010 International Conference on*, pages 412 –417, dec. 2010.
- [77] C.F.V. Loan, Society for Industrial, and Applied Mathematics. *Computational frameworks for the fast fourier transform*. Frontiers in applied mathematics. SIAM, 1992.
- [78] Eduardo Romero-Aguirre, Roberto Carrasco-Alvarez, Ramón Parra-Michel, Aldo Orozco-Lugo, and Antonio Mondragón-Torres. Full-hardware architectures for data-dependent superimposed training channel estimation. *Journal of Signal Processing Systems*, pages 1–19. 10.1007/s11265-012-0706-2.

Apéndice A

Artículos publicados en congresos internacionales

A Hardware-Efficient Frequency Domain Correlator Architecture for Acquisition Stage in GPS

E. Romero-Aguirre,
 R. Parra-Michel
 Dept. of Electrical Engineering,
 CINVESTAV-IPN, Mexico.
 eromero@gdl.cinvestav.mx
 rparra@gdl.cinvestav.mx

O. Longoria-Gandara
 Electronics, Systems and Computer
 Sc. Department, ITESO, Mexico.
 olongoria@iteso.mx

M. Aguirre-Hernández
 Comms. Research Center-Mexico
 Intel Corp., Jalisco, México
 mariano.aguirre@intel.com

Abstract— A frequency domain correlator (FDC) with low-complexity architecture is proposed. This signal processing module is part of FFT-based acquisition stage of a global position system (GPS). The FDC is based on a single reconfigurable core capable of computing both decimation in frequency radix-2 (DIF-2) fast Fourier transform (FFT) and decimation in time radix-2 (DIT-2) inverse fast Fourier transform (IFFT). The proposed FDC reuses the reconfigurable core together with appropriate decimation choice, providing an improvement in resource management while avoiding any additional reorder data hardware. Additionally, the FDC architecture shows a great flexibility due to three levels of parameterization which allows to fix the number of complex input samples, data width and word format. The architecture has been implemented in Altera Cyclone II and Xilinx Virtex II Pro for N -points correlation with different word lengths. The simulation and synthesis results shows that the FDC is suitable for performing acquisition stage in GPS, enabling its utilization in software define radios.

Keywords: Global positioning system, discrete Fourier transform, FPGA, address generator, correlation, FFT.

I. INTRODUCTION

Global Position System (GPS) is a satellite-based navigation. Its operation is based on the computation of a known signal transmitted by different satellites in order to obtain an estimated of user's position. There are two kinds of GPS receiver: software-based and hardware-based. The first option presents a high level of flexibility but requires more calculation time for processing the signals. On the other hand, hardware-based has the advantage of computing the mathematical operations faster than software-based, because it is designed specifically for carrying out these operations, but with the penalty of low degree of flexibility. A trade-off between both design paradigms can be reached via FPGA-based implementation [1]. Consider the GPS receiver block diagram presented in Fig. 1. The RF to IF conversion and the analog to digital conversion (ADC) block constitute the RF front-end receiver. Following it, the acquisition and tracking stage is placed. The main tasks of acquisition part consist in identifying the visible satellites and provides initial code acquisition and Doppler frequency estimation.

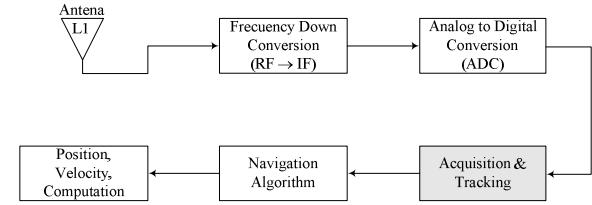


Fig. 1. Block diagram of GPS receiver [1]

These two tasks are accomplished by correlating the incoming signal with a locally replicated satellite code. There are different strategies to perform the acquisition, two of them are: serial search in the time-domain (low-complexity but slow) or the parallel search in the frequency domain (FFT-method) [2, 3]. In the second method, the fundamental processing operation is the frequency-domain correlation, which is depicted in Fig. 2 with shadow blocks inside of acquisition stage. After getting the correlation, next blocks must calculate the module square, search the maximum and compare with a threshold. Finally, the outputs of acquisition are used in tracking stage as the initial code acquisition and Doppler frequency estimate.

This paper is focused on FDC design architecture with the following features: portable (EDA tools independence) and fully parameterizable (N -points, datawidth and word format). The proposed architecture allows that one single processor can be reused within the FDC and avoids additional data reordering hardware, which is typical in FFT algorithms.

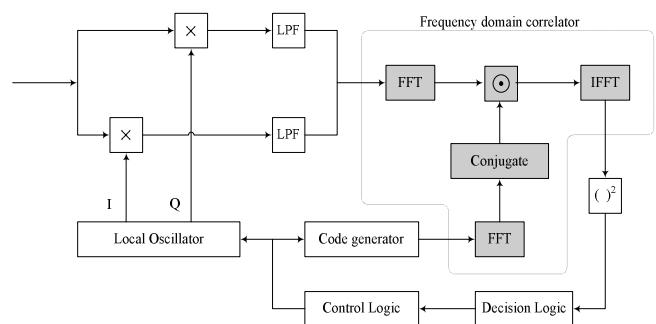


Fig. 2. FDC in FFT-based acquisition stage

The FDC design uses “in-place” method in all operations [4], for that reason it is efficient from memory utilization point of view because only needs $2N$ memory locations, with N denoting the length of the signal to be transformed.

The rest of the paper is organized as follows. In section II, a short description of the radix-2 FFT and permutationless correlation algorithms are given. Section III describes the overall proposed correlator architecture, including the organization of the main components and the interconnection between them. Section IV explains various correlator implementations and the results achieved at hardware consumptions and performance level. Finally some brief conclusions are presented in Section V.

II. FFT AND CORRELATION

2.1 Radix-2 FFT

The Discrete Fourier Transform (DFT) is one of the most used operations in digital signal processing, because it facilitates the efficient transformation between the time-domain and the frequency-domain for a discrete signal.

The N -point DFT of time-domain discrete signal $x(n)$ is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}, \quad \text{for } k = 0, 1, \dots, N-1 \quad (1)$$

$$W_N^{nk} = e^{-j\frac{2\pi nk}{N}} = \cos\left(\frac{2\pi nk}{N}\right) - j \sin\left(\frac{2\pi nk}{N}\right) \quad (2)$$

where W_N^{nk} are the twiddle factors, n and k represents the discrete time-domain and normalized frequency-domain indexes, respectively.

The processing time and implementation cost of DFT using (1) is too large even for short-length sequences. For that reason, it is usually preferable to use the FFT, which is a fast algorithm for computing the DFT. There are two basic types of FFT algorithms: decimation in time (DIT) [5] and decimation in frequency (DIF) [6]. If radix-2 is used then the number of samples must be a power of two ($N = 2^t$, where $t \in \mathbb{Z}$).

Recently, many applications need to compute long-length DFT with high performance which conveyed to the development of several ad-hoc FFT processors [7, 8]. In the next paragraphs, the FFT algorithms will be applied for correlation computing in a fast and efficient way.

2.2 Permutationless correlation using radix-2 FFT

The coarse acquisition algorithm uses correlation between the carrier-stripped input signal $r(n)$ and the local code replica $g(n)$. To compute correlation, the following expression is considered:

$$z(n) = \sum_{m=0}^{N-1} r(m)g(m+n) \quad (3)$$

where N is the correlation length (CL).

Transforming (3) to frequency-domain using the DFT, it follows that,

$$z_f(k) = r_f(k)g_f^*(k) \quad (4)$$

where $(\cdot)^*$ denotes the complex conjugate.

The matrix form of (4) is given by

$$\mathbf{z}_f = \mathbf{r}_f \odot \mathbf{g}_f^* \quad (5)$$

where \odot is the Hadamard (pointwise) product.

The correlation is obtained back to time-domain by taking IDFT in (5),

$$\mathbf{z} = \mathcal{F}^{-1}\{\mathbf{z}_f\} \quad (6)$$

However, if the FFT is used instead of DFT in the correlation computing then it is necessary to avoid any extra reordering steps inherent in each FFT operation. The following analysis shows how this can be performed by using a non-recursive factorization for radix-2 FFT [9].

The DFT $\mathbf{x}_f \in \mathbb{C}^N$ of the input sequence $\mathbf{x} \in \mathbb{C}^N$ is defined in matrix form by

$$\mathbf{x}_f = \mathbf{W}\mathbf{x} \quad (7)$$

where $\mathbf{W} \in \mathbb{C}^{N \times N}$ is the transformation matrix with elements given by (2).

If $N = 2^t$ then (7) can be computed by DIT-2 FFT algorithm. In this case, the input sequence must be reordered before the computations are performed. Therefore, (7) can be expressed as follows:

$$\mathbf{x}_f = \mathbf{F}\mathbf{x}. \quad (8)$$

In (8), $\mathbf{F} \in \mathbb{C}^{N \times N}$ corresponds to the DIT-2 FFT transformation matrix, which is concisely represented by next factorization:

$$\mathbf{F} = \mathbf{A}_q \mathbf{A}_{q-1} \cdots \mathbf{A}_1 \mathbf{P}_N^T = \mathbf{D} \mathbf{P}_N^T \quad 1 \leq q \leq t \quad (9)$$

where

$$\mathbf{A}_q = \mathbf{I}_s \otimes \mathbf{B}_L, \quad L = 2^q, \quad s = N/L$$

$$\mathbf{B}_L = \begin{bmatrix} \mathbf{I}_{L_0} & \mathbf{\Omega}_{L_0} \\ \mathbf{I}_{L_0} & -\mathbf{\Omega}_{L_0} \end{bmatrix}, \quad L_0 = L/2, \quad \mathbf{\Omega}_{L_0} = \text{diag}(1, W_L^1, \dots, W_L^{L_0-1})$$

with \otimes denotes the Kronecker product.

The permutation operation \mathbf{P}_N is given by

$$\mathbf{P}_N = \mathbf{R}_q \mathbf{R}_{q-1} \cdots \mathbf{R}_1, \quad 1 \leq q \leq t \quad (10)$$

where $\mathbf{R}_q = \mathbf{I}_{2^{t-q}} \otimes \mathbf{\Pi}_{2^q}$ and the expression $\mathbf{\Pi}_{2^q}$ is obtained by permuting the columns of the $2^q \times 2^q$ identity matrix according to vector $\mathbf{v} \in \mathbb{Z}^N$, i.e.

$$\mathbf{\Pi}_{2^q} = \mathbf{I}_{2^q}(:, \mathbf{v}^T) \quad (11)$$

$$\text{with } \mathbf{v} = \begin{bmatrix} \text{even}(0 \text{ to } 2^q-1) & \text{odd}(0 \text{ to } 2^q-1) \end{bmatrix}^T.$$

The factorization in (9) can be reformulated for DIF-2 FFT by taking its transpose and using the fact that \mathbf{F} is symmetric.

$$\mathbf{F} = \mathbf{P}_N \mathbf{A}_q^T \mathbf{A}_{q-1}^T \cdots \mathbf{A}_1^T = \mathbf{P}_N \mathbf{D}^T \quad (12)$$

Similar to (12), the DIT-2 IFFT factorization can be obtained from (9) by replacing each reference to W_L with its complex conjugate \bar{W}_L . Thus,

$$\mathbf{F}^{-1} = \frac{1}{N} \bar{\mathbf{A}}_q \bar{\mathbf{A}}_{q-1} \cdots \bar{\mathbf{A}}_1 \bar{\mathbf{P}}_N^T = \frac{1}{N} \bar{\mathbf{D}} \bar{\mathbf{P}}_N^T. \quad (13)$$

Thereby, the Hadamard product of (5), expressed in terms of (12), is represented as,

$$\mathbf{z}_F = (\mathbf{F} \mathbf{r}) \odot (\mathbf{F} \mathbf{g})^* = \mathbf{P}_N ((\mathbf{D}^T \mathbf{r}) \odot (\mathbf{D}^T \mathbf{g})^*). \quad (14)$$

Finally, the frequency-domain correlation described in (6) can be obtained by applying (13) in (14), i.e.

$$\begin{aligned} \mathbf{z} &= \mathbf{F}^{-1} \mathbf{z}_F \\ \mathbf{z} &= \frac{1}{N} \bar{\mathbf{D}} \bar{\mathbf{P}}_N^T \left[\mathbf{P}_N ((\mathbf{D}^T \mathbf{r}) \odot (\mathbf{D}^T \mathbf{g})^*) \right] \\ \mathbf{z} &= \frac{1}{N} \bar{\mathbf{D}} \left[(\mathbf{D}^T \mathbf{r}) \odot (\mathbf{D}^T \mathbf{g})^* \right]. \end{aligned} \quad (15)$$

It can be note that in (15), the permutation operations were cancelled through judicious choice of decimation in every FFT operation. Also, the FFT and IFFT expressed in (12) y (13) were used for permutation removal analysis purposes.

III. CORRELATOR ARCHITECTURE

The architecture proposed for previously described correlation in (15) is shown in Fig. 3, where six components can be identified: a parameterized reconfigurable IFFT/FFT processor (RIFP), two RAMs, a Hadamard product, a complex conjugate and general control unit (GCU). The entire process to calculate the correlation through each of the components of architecture is described in the subsequent paragraphs.

As soon as the START signal is asserted, the data \mathbf{r} provided by input bus IN is stored in RAM1. When this process is complete, a pipeline stage is performed in the following way: the GCU configures the processor in DIF-2 FFT mode and it begins the FFT computing with RAM1 data; in parallel form, the sequence \mathbf{g} is taken from IN bus and stored in RAM2. Once the FFT in RAM1 has finished, GCU switch the processor to RAM2 and computing another DIF-2 FFT.

It is important to note that after each transformation is accomplished, the FFT values are stored in the corresponding RAM. This strategy is well-known as "in-place" computing (data are read from and written-back to the same memory location).

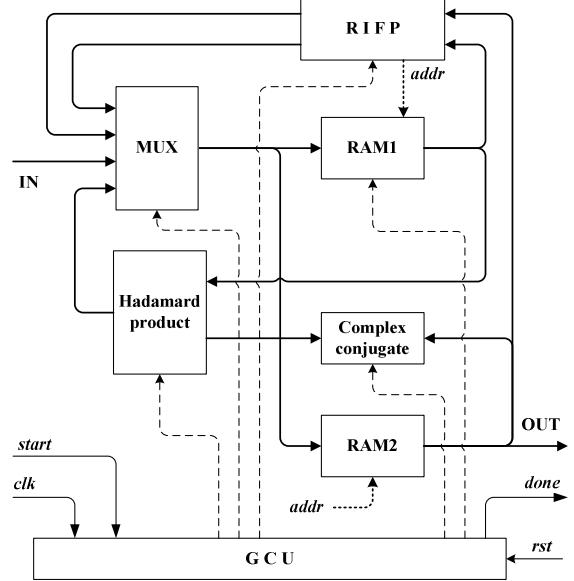


Fig. 3. Architecture of frequency domain correlator

Subsequently, complex conjugate of data RAM2 are calculated and the Hadamard product is performed, and the results are written-back in RAM2. In the next stage, the GCU reconfigures the mode operation of the processor for computing the DIT-2 IFFT with RAM2 data. Finally, the DONE signal is asserted, and the data results are read from RAM2 and sent to OUT bus. With reference to Fig. 2, the correlation computed is fed to the module square block and the FDC is prepared for another signal processing.

Now, functional details of each component of the FDC architecture are presented in next subsections.

3.1 General control unit (GCU)

The GCU supervises all operations of the hardware and provides control signals to other components. It has modeled as a finite state machine (FSM) having eight states. The states and the numbers of clocks cycles are shown in Fig. 4.

It is worth to mention that during FFT_RAM1 state, the RAM2 is filled simultaneously with second data sequence, saving N cycles in the correlation computing. Also, another correlation can begin during UNLOAD_RAM2 state.

The GCU using up-counters, is responsible for addressing the RAMs in loading and unloading data sequences, as well as when the Hadamard product is calculated. But, in all those states involving IFFT/FFT processor tasks, the GCU temporarily grants the addressing process either one or both RAMs to the address generation unit (AGU) of the RIFP.

IDLE_CORR	LOAD_RAM1 (N cycles)	FFT_RAM1 (N log2N cycles)	-	-	-	LOAD_RAM1 (N cycles)
	LOAD_RAM2 (N cycles)	-	FFT_RAM2 (N log2N cycles)	PW_PROD (N cycles)	IFFT_RAM2 (N log2N cycles)	UNLOAD_RAM2 (N cycles)

Fig. 4. States of the GCU

3.2 Reconfigurable IFFT/FFT processor (RIFP)

The main unit of the correlator architecture is the reconfigurable radix-2 IFFT/FFT processor. Its design is memory-based with "in-place" addressing scheme, which is hardware efficient and suitable for long-length data sequences. The RIFP is capable to compute the fixed-point, unscaled DIT-2 IFFT or scaled DIF-2 FFT, of a data block of length N . The processor architecture is depicted in Fig. 5, and consists of a reconfigurable butterfly element (RBE), IFFT/FFT controller (IFC), single port ROM and AGU. The following paragraphs describe the functionality of these units.

Reconfigurable butterfly element: The RBE is the atomic digital signal processing unit. It consists of one complex multiplier and three complex adders that implement a radix-2 mixed decimation processing element, as shown below in Fig. 6. The RBE can be dynamically reconfigured to computes DIF-2 or DIT-2 butterfly upon IFFT/FFT signal value. The N data are read sequentially from RAMs in pairs, together with the corresponding twiddle factor. To minimize the effects due to finite word length, the butterfly outputs always are scaled by factor of two. Therefore, a $1/N$ scaling is uniformly distributed throughout the FFT/IFFT algorithm. This reduces error propagation and increase the SQNR [10].

IFFT/FFT controller: A four states FSM is used to perform overall tasks of the RIFP. Mainly, it keeps track of which RBE is working in a particular stage. The IFC begins the operation when the "starts of transformation" (SOT) signal is asserted. During $N \log_2 N$ cycles, the processor reads, computes butterflies and writes-back data in RAM. When all butterflies in overall stages have been calculated, EOT (end of transformation) flag is turned on and the results are available.

ROM: The twiddle factors defined in (2) are constants that can be pre-computed once off-line and stored in ROM like a look-up table form. In the case of radix 2, only $N/2$ twiddle factors are needed, regardless the type of decimation. Each ROM locations store both real and imaginary parts of the twiddle factors, in that order.

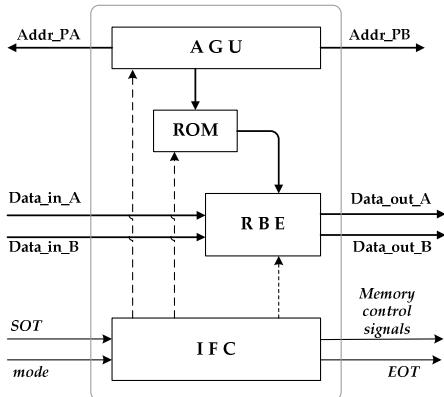


Fig. 5. Block diagram of the RIFP

Address generator unit: Considering the RBE features, a novel approach of address generation with "in-place" memory addressing strategy for mixed decimation is presented. For both RIFP operation modes (DIF-2 FFT or DIT-2 IFFT), the AGU provides the addresses to the dual port RAMs and the ROM. The unit is based on an up-counter that keeps tracking the count of butterfly operations in each stage. The address for every port RAM and ROM are generated as,

$$addr_ROM = counter \ll s \quad (16)$$

$$addr_PA = \text{rotate}(0 \& aux, s, rigth) \quad (17)$$

$$addr_PB = \text{rotate}(1 \& aux, s, rigth) \quad (18)$$

and

$$aux = \text{rotate}(counter, s, left)$$

$$s = \begin{cases} stage - 1, & \text{for DIF-2 FFT} \\ total_stages - stage, & \text{for DIT-2 IFFT} \end{cases}$$

where $\text{rotate}(data, nb, td)$ circularly rotates $data$ in td direction by nb bits, "&" is the concatenate operator and " \ll " is the logical left shift operator.

3.3 RAMs

True dual ports RAM are used to hold the input sequences, intermediate operations and results because each butterfly needs two operands and then produces two results. This allows that the processor can read data from memory or write data into memory at the same time, which provides parallel access. A "ping-pong" scheme is implemented with two dual ports RAMs, so that while RIFP is transforming the contents of a one RAM, another RAM can be filled simultaneously with a different sequence.

The considered RAMs models are generic enough such that the EDA tool infers a dedicated memory of type "block RAM" (BRAM), savings FPGA resources.

3.4 Hadamard product unit

This unit performs the Hadamard product between RAM1 data and complex conjugate of RAM2 data. The result is written-back in RAM2.

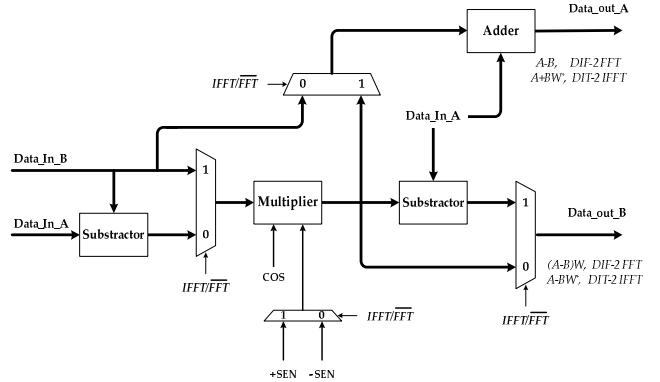


Fig. 6. Reconfigurable butterfly structure

IV. IMPLEMENTATION AND SIMULATION RESULTS

The FDC has been implemented in RTL level using Verilog hardware description language. Correlator sizes of 512, 1024, 2048 and 4096 points with 18, 20, 22, 24 bits word lengths were synthesized and targeted in Altera Cyclone II EP2C35F672C6 and Xilinx Virtex II Pro 2VP70 FPGAs. In all implementations the following considerations apply:

- a) Default settings and no user constraints were selected in Quartus II v9.1 and ISE v10.1.
- b) No any pre-designed component available from the core generator or megafunction libraries were used.
- c) The word length is the same for input signals, twiddle factors and output signals.
- d) All signals are represented in signed fixed-point two's complement.

Table I shows the results for short-length (512 and 1024 points) correlators in Cyclone II. Table II resumes the correlator implementations for long-length (2048 and 4096 points) in Virtex II Pro technology. The values in parenthesis below each feature indicate the total of the corresponding available resources in the selected FPGA.

Analyzing Tables I and II, the results reveal that the frequency operation decreased only marginally despite the correlation size was doubled. Similar behavior is observed for the number of slices/logic elements, registers/FFs and embedded 9-bit multipliers/MULT18X18 needed. Hence, the FDC architecture is robust in terms of correlation size; this feature is suitable for processing long-sequences. If the word length is increased the path delay increases, then the frequency operation is reduced.

The memory locations are $2N$ because “in-place” method was used. Since all required memory is implemented in dedicated FPGA BRAM then the consumption of FPGA LEs/slices/registers is not incremented.

Concerning to the correlation performance, it is clear from Fig. 4 that total number of cycles estimated for calculating an N -point correlation is:

TABLE I. IMPLEMENTATION RESULTS FOR THE CYCLONE II (LOW-COST FPGA)

Correlation length (N)	512			1024		
	18	20	22	18	20	22
Frequency (MHz)	44.29	44.22	41.74	43.91	42.14	40.72
Logic Elements (33216)	1349	1634	1784	1422	1701	1838
Registers (33216)	93	93	93	99	99	99
Memory bits (483.84 Kb)	46.08	51.2	56.32	101.37	115.2	124.41
Block RAMs (105)	10	13	13	22	25	27
Embedded 9bit Multipliers (70)	32	56	56	32	56	56

$$cycles_{corr} = 4N + 3N \log_2 N \quad (19)$$

The word length is not present in (19) therefore it does not affect the number of cycles, but indirectly increases the CT.

The throughput (TP) results are shown in Table III for various implementations in the FPGA Virtex II Pro. For short-sequences the CT is less than 1 ms. In the cases of 2048 and 4096 point, CT is less than 2 ms and 3.4 ms respectively. These time values are adequate for code acquisition searching in GPS.

On other hand, TP is obviously degraded (around 33%) when the number of samples or word lengths is increased. In the last column of Table III, the TP per area gives another useful metric for comparison. A higher value of this ratio indicates that the system implementation is better. Note that all comparisons do not take in account BRAMs.

Fig. 7 shows the SQNR results versus golden values. If CL is increased then the SQNR is reduced due to the chain computing is longer. As expected from quantizing theory, a SQNR gain about 6 dB can be achieved with each additional bit increment in word length. It can be note that a trade-off between SQNR gain and frequency operation is established. For example, according to Table II, an increment of two bits in word length for 2048-points FDC, the frequency operation is decreased around 11.8%, however the SQNR gain obtained is 12 dB.

It is difficult to compare directly the proposed correlator with others similar works, because the lack of available information, differences in technology, tool versions and testing conditions. Besides that, most GPS acquisitions stages are software-based. For that reason, comparison in hardware consumptions level and frequency operation is done between FDC proposed and the correlator inside in coarse acquisition processor reported in [11]. This system was targeted in Virtex II Pro FPGA and it is divided into four domains: 1) Data capture, 2) 4096 FFT, 3) 2048 IFFT, and 4) 10 point DFT/sorting. The domains 2) and 3) that correspond to frequency domain correlator were implemented using IP cores generated by the Xilinx LogiCore Software.

TABLE II. IMPLEMENTATION RESULTS FOR THE VIRTTEX II PRO (OLD HIGH RANGE FPGA)

Correlation length (N)	2048			4096		
	18	20	22	18	20	22
Word length (bits)	56.14	49.47	48.66	56.14	49.47	48.66
Frequency (MHz)	835	891	993	860	915	1017
Slices (33088)	101	101	101	107	107	107
Slice FFs (66176)	1585	1697	1891	1631	1743	1937
LUTs (66176)	10	13	13	22	25	27
Block RAMs (328)	MULT18X18 (328)	8	32	32	8	32

TABLE III. CORRELATOR THROUGHPUTS FOR THE VIRTTEX II PRO

Corr. length (N)	Word length (bits)	Cycles/Correlation	CT (ms)	TP (MS/s)	TP/area (MS/s/slices)
512	18	15875	0.285	1.790	2.25
	20	Idem	0.322	1.588	1.86
	22	Idem	0.327	1.562	1.64
1024	18	34815	0.620	1.651	2
	20	Idem	0.703	1.455	1.65
	22	Idem	0.715	1.431	1.45
2048	18	75776	1.349	1.517	1.81
	20	Idem	1.531	1.337	1.5
	22	Idem	1.557	1.315	1.32
4096	18	163840	2.918	1.403	1.63
	20	Idem	3.311	1.236	1.35
	22	Idem	3.367	1.216	1.19

Table IV shows hardware consumptions between 4096 point FDC and the 4096 FFT/2048 IFFT domains of [11]. In the word length item; the numbers separated by slash indicates the bits uses for input and output buses.

Frequency operation of 140.174 MHz is achieved in [11] with the penalty of larger hardware consumptions. The 4096-point FDC achieves frequency operation of 50.14 MHz with FPGA resource gains of: 62% in slices, 98% in slice FFs, 75% in LUTs, 42% in BRAMs and 71% in 18-bit multipliers. These gains can be obtained because the proposed correlator uses only a one reconfigurable processor instead two proprietary cores used in [11]. Moreover, the frequency of the FDC can be increased by adding more pipeline stages and sacrificing the gain in the slice FFs utilization.

TABLE IV. HARDWARE CONSUMPTIONS COMPARISON

	2048 IFFT	4096 FFT	4096 FDC
Word length (bits)	15/28	15/15	18/18
Frequency (MHz)	140.174	140.174	56.14
Slices	4447	5227	835
Slice FFs	7736	7703	101
LUTs	6286	6977	1585
Block RAMs	17	39	10
MULT18X18	40	27	8

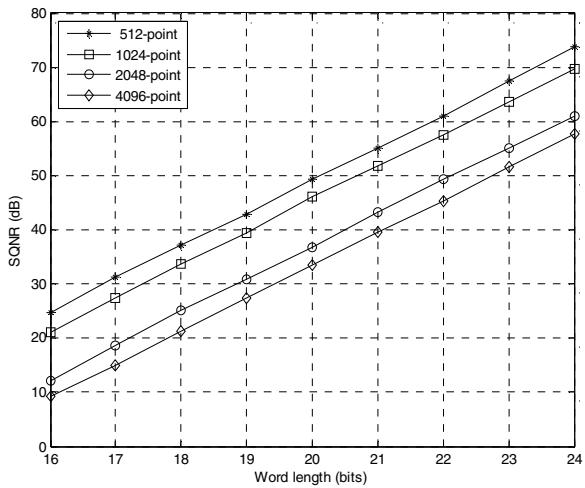


Fig. 7. SQNR versus word length with different correlation sizes

V. CONCLUSIONS

A portable FDC processor suitable to process long-length sequences has been presented. The design approach does not experience drastic frequency operation degradation when the correlation size is doubled. It keeps low-complexity and low-area cost because the IFFT/FFT processor is reused three times. The "in-place" method used in the architecture optimizes the amount of memory locations in the design. The controllers for RIFP and FDC processor are simple and easy to implement even for larger number of points. Also, the design was simplified because no any extra hardware stages are needed for reordering sequences. All these little design improvements of FDC architecture together lead to a good trade-off between performance and area utilization for long-sequences, which is suitable for searching the code acquisition in GPS.

ACKNOWLEDGMENT

This work was supported by PROMEP, CONACYT and INTEL-MCORE Research grants.

REFERENCES

- [1] Manandhar, D., Y. Suh, R. Shibasaki, "Software-based GPS receiver – A research and simulation tool for global navigation satellite system," www.gisdevelopment.net/aars/acrs/2004/gps_ns/acrs2004_f004pf.htm, consulted in November 2009.
- [2] Nee D. J. R. V., Coenen A. J. R. M., "New fast GPS code acquisition technique using FFT," Electronics Letters, 1991, 27(2): 158-160.
- [3] Manandhar, D., Y. Suh, R. Shibasaki, "GPS signal acquisition and tracking - An Approach towards development of Software-based GPS Receiver," Technical Report of IEICE, ITS2004-16, July, 2004.
- [4] J. H. Baek, B. S. Son, B. G. Jo, S. K. Oh, and M. H. Sunwoo, "A continuous flow mixed-radix FFT architecture with an in-place algorithm," in Proc. IEEE Int. Symp. Circuits And Systems, vol. 2, May 2003, pp. 133-136.
- [5] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. of Comp., Vol. 19, No. 90, pp. 297-301, April 1965.
- [6] Gentleman, W. M., and G. Sande, "Fast Fourier transforms for fun and profit," AFIPS Proc., 1966 Fall Joint Computer Conf., Vol. 29, pp. 563-678, Washington, D. C.; Spartan, 1966.
- [7] Y. Jung et al., "New efficient FFT algorithm and pipeline implementation results for OFDM/DM applications, IEEE trans. On Consumer Electron., 2003, vol. 49(1), pp. 1099 – 1103.
- [8] Yu-Wei Lin, Hsuan-Yu Liu, and Chen-Yi Lee, "A Dynamic scaling FFT processor for DVB-T applications, IEEE Journal of Solid-State Circuits, 2004, vol. 39(10), pp. 2005 – 2013.
- [9] Charles Van Loan, Computational Frameworks for the Fast Fourier Transform, SIAM, 1992.
- [10] D. J. Proakis and D. G. Manolakis, Digital Signal Processing: Principles, Algorithms and Applications, 3rd ed. New York: Macmillan, 1996.
- [11] Sajabi, C.; Chen, C.-I. H.; Lin D. M. and Tsui, J. B. Y., "FPGA frequency domain Based GPS coarse acquisition processor using FFT" Proceedings of IEEE International Instrumentation and Measurement Technology Conference, pp. 2353-2358, Sorrento, Italy, April 2006.

FULL-HARDWARE ARCHITECTURES FOR DATA-DEPENDENT SUPERIMPOSED TRAINING CHANNEL ESTIMATION

E. Romero-Aguirre, R. Parra, A. G. Orozco

CINVESTAV-GDL
Dept. of Electrical Engineering
{eromero,rparra}@gdl.cinvestav.mx
aorozco@cinvestav.mx

Roberto Carrasco-Alvarez

Universidad de Guadalajara-CUCEI
Dept. of Electronic Engineering
roberto.carrasco@red.cucei.udg.mx

ABSTRACT

Channel estimation based on superimposed training (ST) has been an active research topic around the world in recent years, because it offers similar performance when compared to methods based on pilot assisted transmissions (PAT), with the advantage of a better bandwidth utilization. However, physical implementations of such estimators are still under research, and only few approaches have been reported to date. This is due to the computational burden and complexity involved in the algorithms in conjunction with their relative novelty. In order to determine its suitability for commercial applications, counting with the performance and complexity analysis of the ST approaches is mandatory. This work proposes, at a first time, a full-hardware channel estimator architectures for an ST receiver and for one of its variants, known as data-dependent superimposed training (DDST). The architectures were described using Verilog HDL and implemented in Xilinx Virtex-5 XC5VLX110T FPGA. A fixed-point analysis has been carried out, allowing the design to produce practically equal performance to those achieved with the floating-point models. The synthesis results showed a reduced slices consumption (2%) and frequencies operation of 149 MHz, which allows to conclude that ST/DDST receivers can be utilized in practical developments.

Index Terms— Channel estimation, , data-dependent superimposed training, FPGA, superimposed training.

1. INTRODUCTION

Nowadays, the users of modern communication systems require more efficient systems in order to transmit and receive information at higher rates. For accomplishing this need, it is necessary to propose new algorithms that makes efficient use of bandwidth resources. Due to the fact that mobility is considered in all modern communications standards, it is required to waste part of the data bandwidth in sending pilots for allowing the receiver to estimate the channel. This process can be classified into two big branches:

- Pilot assisted transmissions (PAT).
- Implicit training (IT)

PAT is the technique suggested in actual communication standards, such as GSM, CDMA2000, DVB-T, etc. Under this approach, the data and pilot sequences are transmitted in orthogonal spaces, allowing a simple separation of the received pilots for channel estimation and usable data payload, at the expenses of large bandwidth waste. In the other hand, IT is a methodology recently proposed, where the paradigm consists in transmitting the training sequence hidden in the data signal [1, Ch. 6], that is, data and training are transmitted in non-orthogonal spaces. This approach presents advantages over PAT techniques, because it is not necessary to waste valuable bandwidth, therefore, it has been recognized as a true alternative for future communication standards [2]. However, the channel estimation process becomes more complicated, requiring analysis of complexity before being able to asses its suitability for practical applications.

The simplest way to carry out IT is adding the training signal into the data signal . Such approach is known as superimposed training (ST) which was first proposed in [3] and enhanced in [4, 5, 6, 7]. A refinement of ST, known as data dependent superimposed training (DDST) was presented in [8, 9, 10]. This refinement permits to cancel the interference produced by the transmitted data during the channel estimation process. The latter is achieved by adding to the training sequence utilized in ST, a second training sequence which depends on the transmitted data.

Although the topic of ST is now mature enough from the mathematical modeling point of view, it is remarkable that there is still rather few works devoted to identify its suitability for practical applications. In fact, these works are still in floating point and SW based approaches. Particularly, a DDST implementation is reported in [11], where the algorithm is programmed in a digital signal processor; while in [12] the proposed implementation is based on an embedded processor with hardware accelerators inside of an FPGA. This paper introduces a full-hardware implementation of the chan-

nel estimation section of an ST/DDST receiver. This stage is the main block of any receiver, and paves the way for a practical implementation of the ST/DDST concept.

2. SYSTEM MODEL

In this Section, the DDST algorithm before mentioned is mathematically introduced. Suppose a single carrier, baseband communication system based on DDST, as the presented in Fig. 1. The transmitted signal $s(k)$ is formed by the superposition of the data sequence $b(k)$, the training sequence $c(k)$ and the data dependent training sequence $e(k)$. It is assumed that $c(k)$ is a periodic sequence with period P and power equal to σ_c^2 [4]. The sequence $e(k)$ is constructed as mentioned in [9]. Also, it is considered that $E(b(k)) = 0$, $E(|b(k)|^2) = \sigma_b^2$, $E(|e(k)|^2) = \sigma_e^2$ and $E(\cdot)$ as the expectation operator. The index k helps us to enumerate the samples of the aforementioned signals which are transmitted at a rate equal $1/T$. The transmitted signal is propagated through the time invariant communication channel $h(k)$ which is the abstraction of the propagation medium and the system filters (pulse shaping and matched filters). This channel can be modeled as finite impulse response (FIR) filter which is assumed to have at most L coefficients. Finally, the received signal $x(k)$ is the sum of the signal distorted by the channel and a white Gaussian noise $n(k)$ which posses variance equal to σ_n^2 . Additionally, it is considered that the transmission is performed in blocks of N symbols length, preceded by a cyclic prefix of length $CP \geq L$, as explained in [8]. For the implementation of this algorithm, it is assumed perfect block synchronization which permit us to fix the length of the cyclic prefix to $CP = P$. For simplicity, we have constrained N to be a multiple of P and P to be a power of two.

The received signal after removing the cyclic prefix can be expressed by the following matrix equation:

$$\mathbf{x} = \mathbf{H}(\mathbf{b} + \mathbf{c} + \mathbf{e}) + \mathbf{n}, \quad (1)$$

where \mathbf{x} is a column vector of length N , whose its k -th ele-

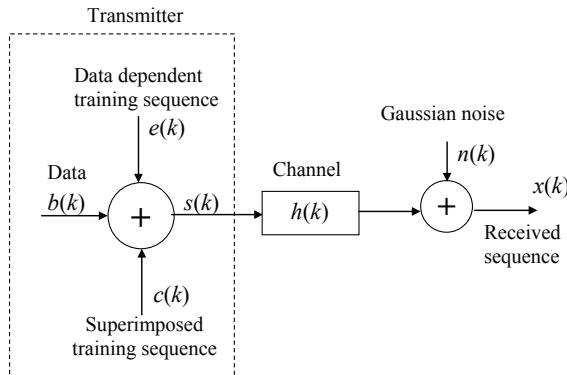


Fig. 1. Digital communication system model considered.

ment is $[\mathbf{x}]_k = x(k)$. In similar fashion is defined \mathbf{b} , \mathbf{c} and \mathbf{e} , respectively. Matrix \mathbf{H} is a circulant matrix which first row is given by $[\mathbf{h}^T, \mathbf{0}_{N-L}^T]$, where \mathbf{h} are the coefficients of the channel impulse response (CIR) ordered in vector form and $\mathbf{0}_{N-L}$ is an all zeros column vector of length $N - L$.

2.1. Channel estimation using DDST

It is possible to observe that due to the periodicity of $c(k)$, $s(k)$ will have embedded a periodic signal with period equal to P . Taking advantage of this characteristic, the channel estimation algorithms based on DDST uses an estimate of the cyclic mean of the received signal. An estimation of such cyclic mean is given by:

$$\mathbf{y} = \mathbf{J}\mathbf{x}, \quad (2)$$

where \mathbf{y} is a column vector of length P which elements are the estimate of the cyclic mean of \mathbf{x} ; $\mathbf{J} = (1/N_P) \times (\mathbf{1}_{N_P}^T \otimes \mathbf{I}_P)$, $N_P = N/P$, $\mathbf{1}_{N_P}$ is an all-ones column vector of length N_P , \mathbf{I}_P is the identity matrix of size $P \times P$ and \otimes is the Kronecker product of two matrices. According to [8], the estimation of the CIR \mathbf{h} is given by:

$$\hat{\mathbf{h}} = \boldsymbol{\Gamma}\mathbf{y}, \quad (3)$$

where $\hat{\mathbf{h}}$ is a vector containing the estimated CIR coefficients, $\boldsymbol{\Gamma}$ is a matrix formed by the first L rows of \mathbf{C}^{-1} and \mathbf{C} is a circulant matrix of size $P \times P$ formed by vector $[c(0), c(1), \dots, c(P-1)]^T$.

3. CHANNEL ESTIMATOR ARCHITECTURE FOR A DDST RECEIVER

This Section introduces two architectures for the DDST-based channel estimation process. Both architectures are based on three main functional units: a cyclic mean estimator, a systolic array matrix-vector multiplier and a memory buffer. In the first architecture, referred in this paper as “channel estimator memory data fed” (CEMDF), the hardware performs the algorithm using the data previously stored in memory. In the second one, which we have called “channel estimator on fly data fed” (CEOOFDF), the channel estimation is carried out at the same time as the input stream is being received and stored in the buffer. It is worth mentioning that in both architectures, it is considered that P must be a power of two, N must be a multiple of P and $P = L$. Also, all the signals are complex valued.

3.1. CEMDF architecture

The CEMDF architecture is presented in Fig. 2. It consists of a main buffer (MB), a parallel cyclic mean estimator (PCME) and a systolic matrix-vector multiplier (SYSMVM). The process to calculate the channel estimated through each of the components of the architecture is described below.

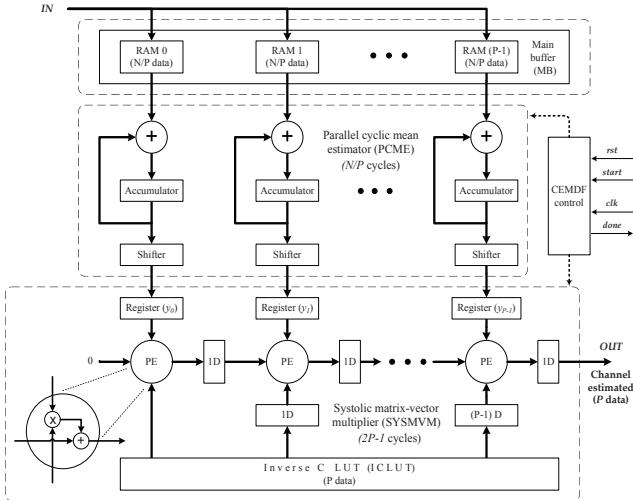


Fig. 2. CEMDF architecture for DDST receiver based on full-HW

As soon as the *start* signal is asserted, the PCME unit, during N_P cycles, process P parallel data at a time of the sequence $x(k)$, taking advantage that it has already been stored in the MB. Once the cyclic mean \mathbf{y} is obtained from the received sequence $x(k)$, the SYSMVM unit will perform the product expressed in (3). After $P + 1$ cycles, *done* flag is asserted and one by one the channel estimated $\hat{\mathbf{h}}$ values are sent to the *OUT* bus. The functional details of each component of the CEMDF architecture are presented in the next subsections.

3.1.1. Main buffer (MB)

It is composed by an array of P memories, each with a depth of N_P , organized as a memory bank as shown in Fig. 3. At the DDST receiver, the data sequence $x(k)$ must be stored in MB as it is being received. This backup is necessary to avoid data lost in case that the other blocks have to modify the received data during the algorithm.

Considering the memory bank features, a “hard-wired” addressing approach is designed for the MB. As depicted in Fig. 4, the $\log_2(N)$ bits corresponding to MB address bus are divided into two parts. The first P least significant bits (LSB) are used to select a particular memory within the bank and the remainder bits are used to address individually each of the locations in the selected memory. Therefore, the N stored data can be viewed as a $N_P \times P$ matrix, where each individual memory in the bank stores a one column of the matrix, as depicted in Fig. 3. The PCME design is massive-parallel based, because it consist of P modules computing P arithmetic means in parallel fashion. Each mean module is conformed by one complex adder, one accumulator register and one shifter, as shown in Fig. 2.

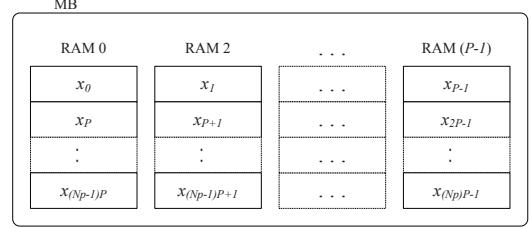


Fig. 3. Data organization in the main buffer (MB)

3.1.2. Parallel cyclic mean estimator (PCME)

Once the N data of the $x(k)$ sequence has been rearranged and stored as matrix in the MB, the PCME hardware reads N_P data from each memories to obtain a sum of their. From the point of view of the algorithm, this is equivalent to add the elements in the matrix’s columns. Next, the results of each sum are divided by N_P in the shifter block. Finally, the P results are fed to the SYSMVM unit in parallel form.

3.1.3. Systolic matrix-vector multiplier (SYSMVM)

The fundamental operation to be performed by CEMDF is the matrix operation represented by (3). From the point of view of hardware consumption, this part of the design is the most critical in the entire architecture. The obvious strategy to accelerate its computing consists in executing as many operations in parallel as possible, but this strategy consume a lot of FPGA resources. Therefore, we decided to design a systolic array processors, similar to the one presented in [13, Ch. 3], in order to obtain good performance and avoiding too much resources consumption.

The processor element (PE), shown at the bottom left in the Fig. 2, is the atomic digital signal processing module in the SYSMVM. It consists of one complex multiplier and one complex adder, that process three flows: the data flow from the PCME, the inverse \mathbf{C} look-up table (LUT) and the data produced by the previous adjacent PE.

A systolic design of (3) was considered. The sizes of matrix $\mathbf{\Gamma}$ and vector \mathbf{y} are $P \times P$ and $P \times 1$, respectively, so the number of PEs needed is P . The projection vector $\mathbf{d} = [1 \ 0]^T$ (see details in [13]) with a vector schedule $\mathbf{s} = [1 \ 1]^T$. The pipelining for this design is one and the computing time for the matrix-vector multiplication is $2P - 1$ clock periods; however, the first resulted datum is available in $P + 1$ clock cycles.

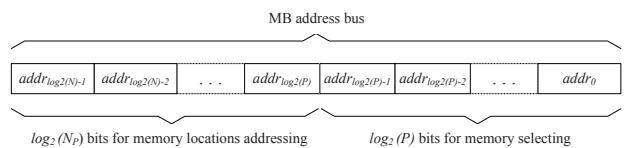


Fig. 4. Hard-wired addressing for MB.

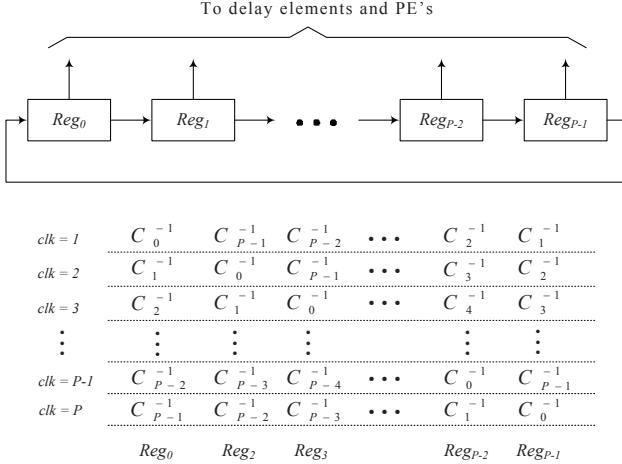


Fig. 5. Simplified architecture of *inverse C look-up table* and its corresponding outputs values.

The values of the circulant matrix \mathbf{C}^{-1} are constants that can be precomputed once off-line and stored in a LUT ROM. Only the first column data are necessary because the remainders columns are shifted versions of the first one. Consequently, the ROM location's number required for the LUT is just P . If traditional design is used then the LUT would be designed with a multi-port ROM of P locations, but it will be synthesized by the employed compiler tool as an array of P single port ROMs. Therefore, the number of memory locations is increased to P^2 . The design shown in Fig. 5 for the \mathbf{C}^{-1} LUT is built with an array of P registers operating as a circular buffer. The first row values of \mathbf{C}^{-1} are stored in the registers. Next, one rotation is applied in each tick of the clock for changing the register's outputs, as indicated in Fig. 5. This solution, called inverse C LUT (ICLUT), saves $P(P - 1)$ locations.

3.2. CEOFDF architecture

The architecture for this estimator is presented in Fig. 6, where the PCME module is substituted by a serial cyclic mean estimator (SCME) due to the fact that a different paradigm is used. The SCME module in Fig. 6 no longer needs to wait until $N + P$ data has been received and stored in MB. During $N + P$ cycles, the cyclic mean is computed “on-the-fly” as each data is being receiving. At the same time, the input sequence is stored and available for other modules. The SCME should be able to identify the first P data corresponding to the CP and do not take them into account when computing the cyclic mean. In the SCME, the RAM locations are used for storing the P sums and when the computing has finished, all of them must be set to zero together with the registers of the *lb_delay* module. The MB is not used and SYSMVM module remains unchanged in the architecture. Additionally, the CEOFDF offers two useful functional features:

1. It can operate in continuous flow when the additional buffer ($2P$ depth) is connected in its *IN* bus.
2. It can be configured to compute the channel estimation values or cyclic mean only, depending upon the *mode* control signal value. In the last case, the results are sent to *cyclic mean out* bus in serial form and *cmflag* is asserted.

4. IMPLEMENTATION AND SIMULATIONS RESULTS

This Section presents a comparative analysis of the HW-level implementation of CEMDF and CEOFDF architectures. Both channel estimators have been implemented in RTL using Verilog hardware description language. These estimators process input sequences of length $N = 512$ with $CP = P$ and $P = 8$. They were synthesized and targeted in Xilinx Virtex-5 XC5VLX110T. In the two implementations the following considerations apply:

1. Default settings and ”no user constraints” were selected in Xilinx ISE v11.
2. No one pre-designed component available from the core generator library was used.

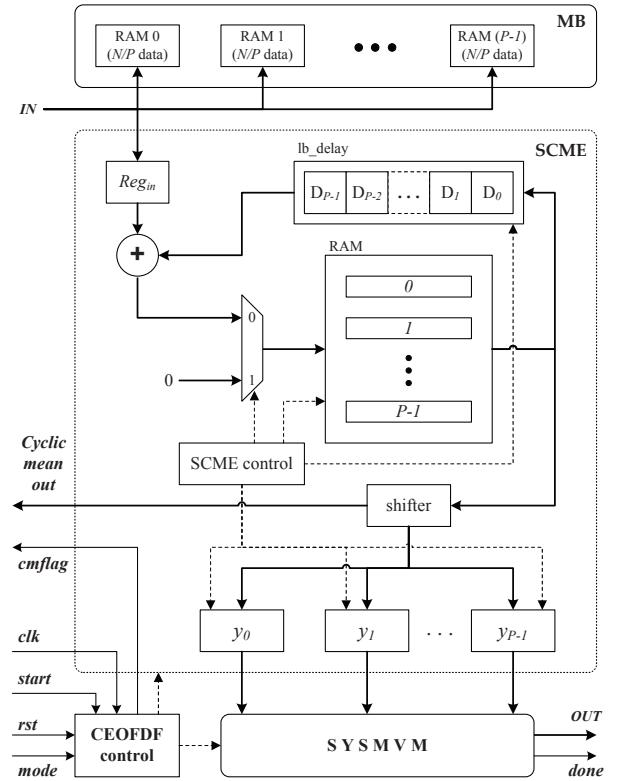


Fig. 6. CEOFDF hardware architecture.

3. All signals are represented in signed fixed-point two's complement.
4. Both real and imaginary parts of the complex values are stored together, in that order, in the same memory location or register.

It is worthwhile to mention that it is difficult to compare directly the proposed channel estimators with the others previous works [11, 12], because the lack of available information, the differences in technology, tool versions and testing conditions. The comparison against the implementation reported in [11] is not possible because this system is full-software programmed in TMS320C6713 DSP with a 300MHz external clock. In [12] it is describes a hybrid software-hardware FPGA implementation of the DDST receiver, but regarding the module corresponding to the channel estimation, only the arithmetic mean was accelerated by dedicated hardware. The matrix-vector operation described in (3) was implemented in software. Moreover, in none the aforementioned works present results about estimation error and performance graphics.

Table I resumes the synthesis results for both channel estimators. The values in the parenthesis in each feature indicate the total of corresponding available resources in the FPGA. The results in Table I reveal that the frequency operation decreased only marginally in CEOFDF architecture despite the cyclic mean unit is not massive-parallel. Similar behavior is observed for the number of slice registers, slice LUTs, block RAMs and DSP48E blocks. One might think that CEMDF system would consume more resources in the FPGA. But the hardware consumption is almost the same; this is because many of the FPGA internal resources are shared among the other modules inside the architecture. For example, a slice register can “donate” its remainder registers to another entity, likewise a slice LUT multiplexer can share its resources with other modules. Another reason is that the consumptions of both architectures are minimal with respect to the total resources of the FPGA. Note that the Table I does not include comparison with previously implementations described in [11, 12] because there were no data concerning the items listed in such Table.

Now, it is important to determine an approximate of the numbers of cycles in both architectures for computing the es-

TABLE I. IMPLEMENTATION RESULTS OF THE CHANNEL ESTIMATORS

Channel estimator		CEMDF	CEOEDF
Input length (without CP)	(N)	512	512
Frequency	(MHz)	149.949	149.439
Slice registers	(69120)	1652 (2%)	1441 (2%)
Slice LUTs	(69120)	1421 (2%)	1213 (1%)
Block RAMs	(148)	1 (<1%)	1 (<1%)
DSP48Es	(64)	32 (50%)	32 (50%)

TABLE II. CHANNEL ESTIMATOR THROUGHPUTS COMPARISON

Channel estimator	Input length	Cycles/estimation	CET (us)	TP (MS/s)	TP/area (MS/s/slices)
CEMDF	512	601	4.008	127.74	41.568e3
CEOEDF	512	545	3.647	142.584	53.723e3
Arithmetic mean coprocessor in [12]	NA	2238	20	NA	NA

timated CIR coefficients. In the CEMDF case is:

$$cycles_{\hat{h}} = (N + P) + N_P + 2P + 1 \quad (4)$$

The first term in (4) is included because it corresponds to the previous data storage in MB and the CEMDF only operates under that condition. On the other hand, the number of cycles in CEOEDF is given by

$$cycles_{\hat{h}} = (N + P) + 2P + 1 \quad (5)$$

Analyzing the set of metrics listed in Table II, it is possible to compare the performance in both designs. The channel estimation time (CET) is the time elapsed from the beginning of the channel estimation process until its computing has finished. The throughput (TP) per area is another useful metric for comparison, a higher value of this ratio indicates that the system implementation is better. Analyzing the Table II shows that although the CEOEDF system has the best performance in all compared metrics, there are no drastic differences with respect to the CEMDF architecture. As expected, both architectures provide far better performance against the arithmetic mean coprocessor used in [12].

The validity of the provided architectures is granted by comparing their results with the floating-point simulation golden model programmed in Matlab, in terms of the mean squared error (MSE) and signal to noise ratio (SNR). Thereby, the following scenario (similar to used in [10] was considered. The length of data input is $N = 512$ symbols 4-QAM. The channel is randomly generated at each Monte Carlo run and is assumed to be Rayleigh with length $L = 8$. The power of training sequence is set to 0.2 of σ_s^2 and its period P , as mentioned previously, is set to L . Also, perfect block synchronization is assumed.

The MSE performance in the CEOEDF architecture for one hundred Monte Carlo trials is shown in Fig. 7. Note that the MSE of the hardware estimator is close to the theoretical line and almost indistinguishable with respect to the golden model. The signal to quantization noise ratio (SQNR) in both estimators is about 77dB. The introduced designs consider non-rounding scheme and different formats at each stage of the algorithm, which could not be presented due to the lack of space. However, it can be commented that the fixed-point word format required in average 18 fractional bits and five integer bits.

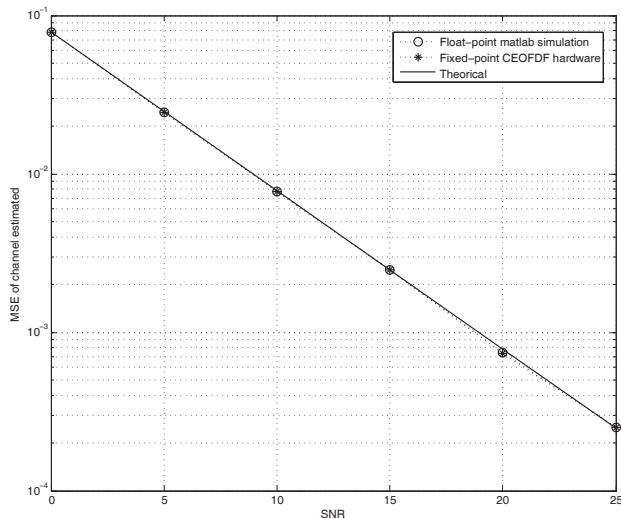


Fig. 7. MSE performance of the CEOFDF.

5. CONCLUSIONS

In this paper it has been shown two architectures for DDST channel estimation, which represents the first channel estimators implementations under the full-HW philosophy for a DDST receiver. Both designs present high throughputs and reduced FPGA resources consumption, achieving a good trade-off between performance and area utilization. Its internal modules can be easily modified to process channels of different size. The validity and performance of the proposed architectures has been verified by Monte Carlo simulations, where an SQNR of 77dB are achieved while no significant differences with respect to both theoretical MSE curve and the floating-point golden model. The provided results show that ST/DDST concepts can be effectively utilized in future communications standards.

6. ACKNOWLEDGMENT

This work was supported by PROMEP, CONACYT 84559-Y and Jal-158899 research grants.

7. REFERENCES

- [1] S. Haykin and K.J. Ray Liu, *Handbook on Array Processing and Sensor Networks*, Wiley, 2009.
- [2] A. Goljahi, N. Benvenuto, S. Tomasin, and L. Vangelista, “Superimposed sequence versus pilot aided channel estimations for next generation dvb-t systems,” *Broadcasting, IEEE Transactions on*, vol. 55, no. 1, pp. 140 –144, march 2009.
- [3] B. Farhang-Boroujeny, “Pilot-based channel identification: proposal for semi-blind identification of communication channels,” *Electronics Letters*, vol. 31, no. 13, pp. 1044–1046, 22 June 1995.
- [4] A.G. Orozco-Lugo, M.M. Lara, and D.C. McLernon, “Channel estimation using implicit training,” *IEEE Trans. Signal Process.*, vol. 52, no. 1, pp. 240–254, Jan 2004.
- [5] J.K. Tugnait and Weilin Luo, “On channel estimation using superimposed training and first-order statistics,” in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '03)*, 2003, vol. 4, pp. IV-624–7 vol.4.
- [6] J.K. Tugnait and Xiaohong Meng, “On superimposed training for channel estimation: performance analysis, training power allocation, and frame synchronization,” *IEEE Trans. Signal Process.*, vol. 54, no. 2, pp. 752–765, Feb. 2006.
- [7] R. Carrasco-Alvarez, R. Parra-Michel, A. G. Orozco-Lugo, and J. K. Tugnait, “Enhanced channel estimation using superimposed training based on universal basis expansion,” *IEEE Trans. Signal Process.*, vol. 57, no. 3, pp. 1217–1222, 2009.
- [8] E. Alameda-Hernandez, D.C. McLernon, A.G. Orozco-Lugo, M.M. Lara, and M. Ghogho, “Frame/training sequence synchronization and dc-offset removal for (data-dependent) superimposed training based channel estimation,” *IEEE Trans. Signal Process.*, vol. 55, no. 6, pp. 2557–2569, June 2007.
- [9] M. Ghogho and Ananthram Swami, “Improved channel estimation using superimposed training,” in *Proc. IEEE 5th Workshop on Signal Processing Advances in Wireless Communications*, 11–14 July 2004, pp. 110–114.
- [10] M. Ghogho, D. McLernon, E. Alameda-Hernandez, and A. Swami, “Channel estimation and symbol detection for block transmission using data-dependent superimposed training,” *IEEE Signal Process. Lett.*, vol. 12, no. 3, pp. 226–229, 2005.
- [11] V. Najera-Bello, “Design and construction of a digital communications system based on implicit training,” M.S. thesis, CINVESTAV-IPN, 2008.
- [12] Fernando Martín del Campo, René Cumplido, Roberto Pérez-Andrade, and A. G. Orozco-Lugo, “A system on a programmable chip architecture for data-dependent superimposed training channel estimation,” *International Journal of Reconfigurable Computing*, vol. 2009, 2009.
- [13] S. Kung, *VLSI Array processors*, Prentice Hall, 1985.

ARCHITECTURE BASED ON ARRAY PROCESSORS FOR DATA-DEPENDENT SUPERIMPOSED TRAINING CHANNEL ESTIMATION

E. Romero-Aguirre, R. Parra-Michel
*Dept. of Electrical Engineering
 CINVESTAV-GDL
 Zapopan, Jalisco
 {eromero, rparra}@gdl.cinvestav.mx*

Roberto Carrasco-Alvarez
*Dept. of Electronic Engineering
 UDG-CUCEI
 Guadalajara, Jalisco
 roberto.carrasco@red.cucei.udg.mx*

A.G. Orozco-Lugo
*Dept. of Electrical Engineering
 CINVESTAV-DF
 México, D. F.
 aorozco@cinvestav.mx*

Abstract—Channel estimation is a challenging problem in wireless communication systems because of users mobility and limited bandwidth. A plethora of methods based on pilot assisted transmissions (PAT) have been proposed in most practical systems to overcome this problem, but with the penalty of extra bandwidth consumption for training. Channel estimation based on superimposed training (ST) has emerged as an alternative in recent years because it saves valuable bandwidth by adding a training periodic sequence to the data signal instead of multiplexing them. However, although ST and one of its variants, known as data dependent ST (DDST), have been an active research topic, only few physical implementations of such estimators have been reported to date. In this work a full-hardware architecture based on array processors (AP) for DDST channel estimation is presented and it is compared with previous approaches. The design was described using Verilog HDL and targeted in Xilinx Virtex-5 XC5VLX110T. The synthesis results showed a slices consumption of 3% and a frequency operation of the 115 MHz. A Monte Carlo simulation demonstrates that the mean square error (MSE) of the channel estimator implemented in hardware is practically the same than the one obtained with the floating-point golden model. The high performance and reduced hardware of the proposed channel estimator allows us to conclude that it can be utilized in practical DDST receivers developments.

Keywords—Channel estimation; data-dependent superimposed training; FPGA; implicit training; systolic arrays.

I. INTRODUCTION

The process of estimate the communication channel is a vital part of any modern communication system. Its importance relies on the fact that a correct channel estimation leads to a reduction of the bit error rate (BER). This estimator must deal with multiple phenomena such as multipath propagation and frequency Doppler (due to the mobility of the users). In order to deal with these problems, current communication standards specifies the transmission of pilot signals which are known in the receiver, permitting an ease estimation of the communication channel. The way of transmit such pilot signals can be classified in two big branches: pilot assisted transmission (PAT) —where pilot and data signals are multiplexed in time, frequency, code, space or in a combination of the mentioned domains— and implicit training (IT), a technique proposed recently where the pilot

signal is hidden in the data transmitted. PAT is the technique implemented in actual standards, such as, WiMax, WiFi, Bluetooth, etc. It presents the advantage that pilots and data relies on orthogonal subspaces allowing a simple separation of them in the receiver; however, it is necessary to decrease the available bandwidth for data in order to transmit the pilot signal. On the other hand, IT overcomes this problem because all the time, data and pilot signal are transmitted; nevertheless, it leads to a transmission of such signals into non-orthogonal subspaces. Despite the aforementioned, IT has been recognized as a feasible alternative for future communication standards [1].

The simplest form to carry out IT is to add (superimpose) the pilot signal to the data. Such approach is known as superimposed training (ST), first proposed in [2] and enhanced by diverse authors which results are summarized in [3, Ch. 6]. In [4], [5], [6], [7], [8] was presented a refinement of ST known as data-dependent superimposed training (DDST), it permits to null the interference of data during the estimation process via the addition of a new training sequence—which depends on the transmitted data—together with the data and the ST sequence.

Because the benefits that offer ST/DDST, it is necessary to develop efficient implementations of these algorithms. Although these techniques have been widely studied, to this point, there exist few reported practical implementations in the literature. In fact, almost all of them are approximations based on floating-point and software. In [9], the algorithms are programmed in a digital signal processor (DSP) for a low rate communication system, while in [10] the proposed implementation is developed into a embedded microprocessor with hardware accelerators inside of a FPGA. For that reason, this paper focuses on presenting a full-hardware architecture for implementing the channel estimation section of an ST/DDST receiver. The novelty of such architecture relies on the fact that a systolic array is utilized for performing the entire estimation process—instead of two separated signal processing modules—which leads to a system with high throughput, low hardware consumption and high degree of re-usability. The performance results of the proposed

approach confirm the feasibility of the DDST concept for its practical implementation in a communication standards.

II. SYSTEM MODEL

This Section is devoted to introduce the DDST algorithm before mentioned. Suppose a single carrier, baseband communication system based on DDST as the presented in Fig. 1. The transmitted signal $x(k)$ is conformed by the sum of the data sequence $b(k)$, the training sequence $c(k)$ and the data dependent training sequence $e(k)$. The index k permits to enumerate the samples of such signals which are transmitted at a rate equal to $1/T$. $c(k)$ is a periodic sequence with period equal to P and power equal to σ_c [11]. It is assumed that the data sequence is a zero-mean, stationary stochastic process with power equal to σ_b where the symbols of such process come from a equiprobable alphabet. The sequence $e(k)$ is constructed as mentioned in [5]. $s(k)$ is propagated through the communication channel $h(k)$ whose time impulse response is conformed by the convolution of the system filters and the propagation medium impulse responses (all of them assumed to be time-invariant). Such channel can be modeled as a finite impulse response (FIR) filter with L time-invariant coefficients as much. Finally, the distorted signal by the channel is contaminated with the noise $n(k)$ for conforming the received signal $x(k)$. $n(k)$ is a zero-mean white Gaussian noise which posses variance equal to σ_n and it is obtained by filtering white Gaussian noise with the matched filter of the receiver. It is assumed the transmission of blocks of N symbols which are preceded by a cyclic prefix of length $CP > L$. It is also assumed perfect block synchronization which permits to fix $CP = P > L$. For ease of implementation, it is assumed that N is a multiple of P and P is a power of two.

Thus, the received signal after removing the cyclic prefix can be expressed in a mathematical form as:

$$\mathbf{x} = \mathbf{H}(\mathbf{b} + \mathbf{c} + \mathbf{e}) + \mathbf{n}, \quad (1)$$

where \mathbf{H} is a circulant matrix whose first row is given by $[\mathbf{h}^T, \mathbf{0}_{N-L}^T]$ where \mathbf{h} is a vector containing the coefficients of the channel impulse response (CIR) and $\mathbf{0}_{N-L}$ is an all zeros

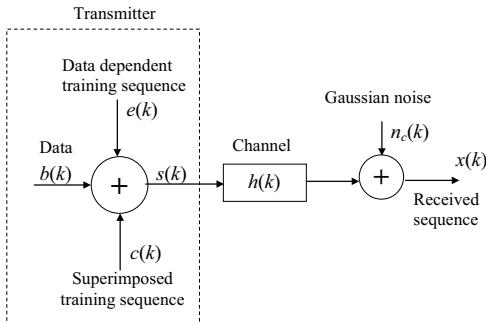


Figure 1. Digital communication system model considered.

column vector of length $N - L$. Similarly, \mathbf{x} , \mathbf{b} , \mathbf{c} , \mathbf{e} and \mathbf{n} are column vectors of length N , whose k -th element are equal to $[x]_k = x(k)$, $[b]_k = b(k)$, $[c]_k = c(k)$, $[e]_k = e(k)$ and $[n]_k = n(k)$ respectively.

A. Channel estimation using DDST

It is possible to observe that due to the periodicity of $c(k)$, $s(k)$ will have embedded a periodic signal with period equal to P . Taking advantage of this characteristic, an estimate of the cyclic mean of the received signal is utilized for performing the estimate of the channel. Such cyclic mean estimator can be defined as:

$$\mathbf{y} = \mathbf{J}\mathbf{x}, \quad (2)$$

where \mathbf{y} is a column vector of length P which elements are the estimated of the cyclic mean of \mathbf{x} and \mathbf{J} is given by:

$$\mathbf{J} = \frac{1}{N_P} (\mathbf{1}_{N_P}^T \otimes \mathbf{I}_P), \quad (3)$$

where $\mathbf{1}_{N_P}$ is an all-ones column vector of length N_P , \mathbf{I}_P is the identity matrix of size $P \times P$ and \otimes is the Kronecker product of two matrices. According to [4], the estimation of the CIR is given by:

$$\hat{\mathbf{h}} = \boldsymbol{\Gamma}\mathbf{y}, \quad (4)$$

where $\hat{\mathbf{h}}$ is a vector containing the estimated CIR coefficients, $\boldsymbol{\Gamma}$ is a matrix formed by the first L rows of \mathbf{C}^{-1} and \mathbf{C} is a circulant matrix of size $P \times P$ formed by vector $[c(0), c(1), \dots, c(P-1)]^T$.

III. CYCLIC MEAN ALGORITHM USING ARRAY PROCESSORS AND PARTITIONING

The next analysis describes how the cyclic mean is obtained using a systolic array that computes a matrix-vector multiplication (MVM). Consider (2), where it is not possible to perform directly the MVM operation due to the Kronecker product involved. To avoid this cumbersome operator, the same equation can be reformulated as follows:

$$\mathbf{y} = \frac{1}{N_P} (\mathbf{N}\mathbf{1}_{N_P}), \quad (5)$$

where \mathbf{N} is a matrix of size $P \times N_P$ which is defined as follow:

$$\mathbf{N} = \begin{bmatrix} \mathbf{x}_0 & \mathbf{x}_P & \mathbf{x}_{2P} & \cdots & \mathbf{x}_{(N_P-1)P} \\ \mathbf{x}_1 & \mathbf{x}_{P+1} & \mathbf{x}_{2P+1} & \cdots & \mathbf{x}_{(N_P-1)P+1} \\ \mathbf{x}_2 & \mathbf{x}_{P+2} & \mathbf{x}_{2P+2} & \cdots & \mathbf{x}_{(N_P-1)P+2} \\ \vdots & \vdots & \vdots & & \vdots \\ \mathbf{x}_{P-1} & \mathbf{x}_{2P-1} & \mathbf{x}_{3P-1} & \cdots & \mathbf{x}_{N_PP-1} \end{bmatrix}. \quad (6)$$

An architecture based on array processors (AP) for computing (5) would be impractical from point of view of hardware consumption because it will need N_P processor elements (PEs). This problem is known as *problem-size dependent array* where the algorithm requires a systolic

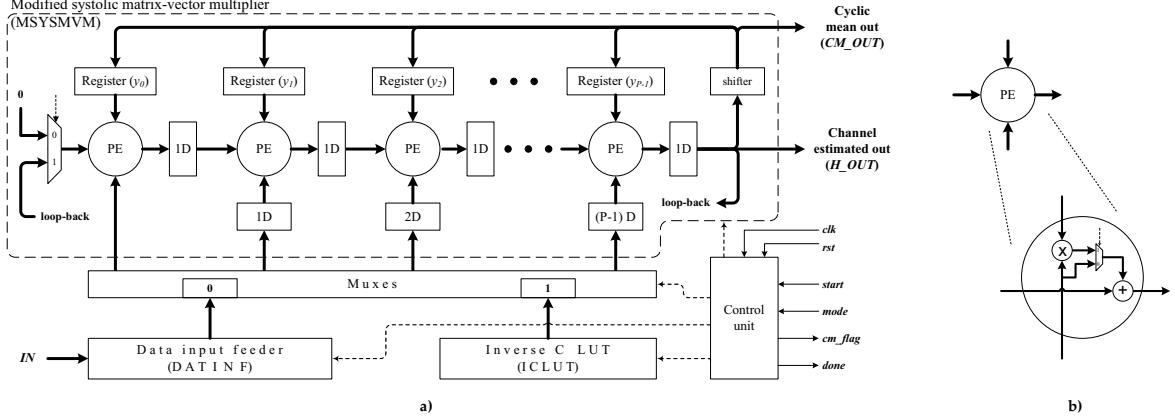


Figure 2. Systolic channel estimator for DDST receiver: a) simplified architecture, b) the PE module.

AP whose size depends on the complexity of the problem to be solved. However, it is possible to map the cyclic mean algorithm to a systolic AP of a smaller size using the *partitioning* method [12, Ch. 12]. Consider \mathbf{N} to be partitioned in blocks of size chosen to match a systolic array size P then (6) becomes

$$\mathbf{x} = \left[\mathbf{B}_0 | \mathbf{B}_1 | \cdots | \mathbf{B}_{\frac{N_P}{P}-1} \right], \quad (7)$$

where

$$\mathbf{B}_i = \begin{bmatrix} \mathbf{x}_{iP^2} & \mathbf{x}_{iP^2+2P} & \cdots & \mathbf{x}_{iP^2+(P-1)P} \\ \mathbf{x}_{iP^2+1} & \mathbf{x}_{iP^2+2P+1} & \cdots & \mathbf{x}_{iP^2+(P-1)P+1} \\ \vdots & \vdots & & \vdots \\ \mathbf{x}_{iP^2+P-1} & \mathbf{x}_{iP^2+2P-1} & \cdots & \mathbf{x}_{iP^2+P^2-1} \end{bmatrix},$$

for $i = 0, \dots, \frac{N_P}{P} - 1$.

In similar fashion, $\mathbf{1}_{N_P}$ is partitioned in $\frac{N_P}{P}$ unitary vectors of length $\mathbf{1}_P$. Substituting (7) in (5), the cyclic mean with *partitioning* is concisely expressed as:

$$\mathbf{y} = \frac{1}{N_P} (\mathbf{B}_0 \mathbf{1}_P + \mathbf{B}_1 \mathbf{1}_P + \cdots + \mathbf{B}_{\frac{N_P}{P}-1} \mathbf{1}_P). \quad (8)$$

Therefore, the array of PEs will process a one pair of \mathbf{B} and $\mathbf{1}_P$ blocks after another in a sequential manner together with partial results.

IV. CHANNEL ESTIMATOR ARCHITECTURE

This Section introduces an architecture for the DDST-based channel estimation process. Its design is based on MVM operation, which is carried out in systolic way into AP. The main idea in the system design is to reuse the same systolic array for computing the cyclic mean of the received data. The proposed architecture, called in this paper as “systolic DDST channel estimator” (SYSDCE) is depicted in Fig. 2a. It can be identified four functional units: a modified

systolic matrix-vector multiplier (MSYSMVM), a data input feeder (DATINF), an inverse C lock-up table (ICLUT) and a control unit (CU). Broadly speaking, the SYSDCE operation can be divided into three phases: input sequence storage, cyclic mean compute and CIR estimate.

As soon as the *start* signal is asserted, an $N + CP$ data samples (vector \mathbf{x} and cyclic prefix, respectively) can be read from the input port IN . After excluding CP samples corresponding to the cyclic prefix, the rest of samples are rearranged and stored in the memory bank of DATINF. When this process is finished, the CU configures the MSYSMVM unit and during N_P cycles it reads P parallel data per cycle from DATINF and computes the cyclic mean \mathbf{y} . Once this phase is finished, the obtained vector \mathbf{y} together with ICLUT data are fed to the MSYSMVM again for performing the product expressed in (4). Finally, after $P + 1$ cycles, *done* flag is asserted and one by one the coefficients of the channel estimated $\hat{\mathbf{h}}$ are sent to the bus H_OUT . It is worth mentioning that the SYSDCE can be configured to compute only the cyclic mean if *mode* input control signal has been set to zero. In this case, *cm_flag* out is asserted to indicate that a valid results are available in CM_OUT bus. Thus, the channel estimator is prepared for another data sequence processing. A deeper explanation about each component of the SYSDCE architecture will be given in the subsections.

A. Modified systolic matrix-vector multiplier (MSYSMVM)

The fundamental operation to perform by SYSDCE is a matrix-vector multiplication which is high time-processing demanding. The hardware design for solving this operation is the most critical part in the architecture. The obvious strategy for accelerate MVM consists in compute as many operations as possible, with the penalty of high consume of FPGA resources. Therefore, this paper proposes a modification of the systolic MVM presented in [13, Ch. 3] in order to obtain a good performance with reasonable resources consumption. This modification allows to compute the cyclic

mean using *partitioning* method with the same systolic array reported. Fig. 2b shows the processor element (PE), which is the atomic digital signal processing module in MSYSMVM. It processes three flows: the data flow from the ICLUT or DATINF, the input registers values and the data produced by the previous adjacent PE.

In the MSYSMVM design was considered that the number of PEs needed (AP size) is P , which matches with the dimensions of matrix Γ and vector \mathbf{y} respectively. The projection vector $\mathbf{d} = [1 \ 0]^T$ (see details in [13]) with a vector schedule $\mathbf{s} = [1 \ 1]^T$ were used. The pipelining period for this design is one and the computing time for the full MVM is $2P - 1$ clock periods.

For computing the cyclic mean using the MSYSMVM module, the original structure of PE was modified with an additional multiplexer. For that reason, the PE can perform all trivial multiplications by bypassing the data from the input of the complex multiplier directly to the complex adder.

B. Data input feeder (DATINF)

Similar to almost any systolic array, the MSYSMVM needs that the data, —which will be fed to each of its PEs— must be given in a defined order before to process it. In the proposed approach, the module DATINF is responsible for performing this task. It is composed by an array of P memories, each with a depth of N_P , organized as a memory bank as shown in Fig. 3. DATINF reads $N + P$ data from IN bus, it identifies and removes the first P data corresponding to CP. Subsequently, this module rearranges this sequence (correspondence to $x(k)$) in $\frac{N_P}{P}$ blocks of size $P \times P$ in order to form $\mathbf{B}_0, \mathbf{B}_1, \dots, \mathbf{B}_{\frac{N_P}{P}}$. Therefore, the N stored data can be viewed as a $N_P \times P$ matrix, where each individual memory in the bank stores one column of each block and the blocks are stored consecutively one after the other, as depicted in Fig. 3.

Each datum of the input sequence \mathbf{x} has associated three addresses that defines its location inside the memory bank: block number (blk_num), memory number (mem_num) and memory address (mem_addr). The DATINF must generate these addresses using the following expressions:

$$blk_num = \left\lfloor \frac{k \times N_P}{N \times P} \right\rfloor, \quad k = 0, 1, \dots, N - 1, \quad (9a)$$

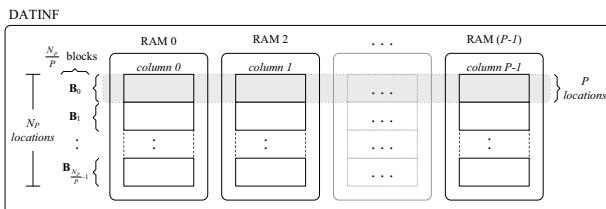
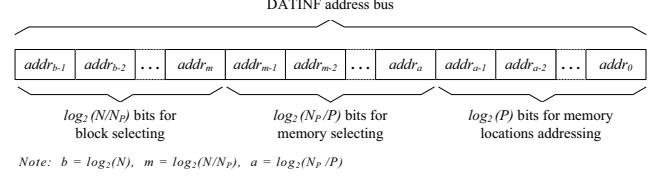


Figure 3. Data block organization in the DATINF



Note: $b = \log_2(N)$, $m = \log_2(N/N_P)$, $a = \log_2(N_P / P)$

Figure 4. Hard-wired addressing for memory bank.

$$mem_num = \left\lfloor \frac{k \times N_P - blk_num \times N \times P}{N} \right\rfloor, \quad (9b)$$

$$mem_addr = (k \bmod P) + (P \times blk_num), \quad (9c)$$

where k is the k -th element of \mathbf{x} and $\lfloor \cdot \rfloor$ denotes the *floor* operator.

In order to minimize the hardware consumption, a “hard-wired” addressing approach was built for the memory bank. As shown in Fig. 4, the $\log_2(N)$ bits corresponding to DATINF *address bus* are split into three parts. The first $\log_2(\frac{N}{N_P})$ most significant bits (MSB) are used for block selecting, the next $\log_2(\frac{N_P}{P})$ MSB are used to select a particular memory in the bank and the remainder $\log_2(P)$ bits are used to address individually each of the locations in the selected memory.

C. Inverse C look-up table (ICLUT)

The values of the circulant matrix \mathbf{C}^{-1} are constants that can be precomputed once off-line and stored in a look-up table (LUT) ROM. Only the first column data are necessary because the remainders columns are shifted versions of the first one. Consequently, the ROM location’s number required for the LUT is just P . If traditional design is used then the LUT would be designed with a multi-port ROM of P locations, but it will be synthesized by the employed compiler tool as an array of P single port ROMs. Therefore,

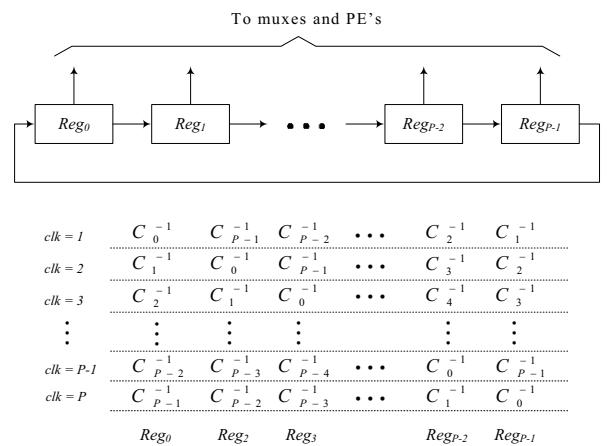


Figure 5. Simplified architecture of inverse C look-up table (ICLUT) and its corresponding outputs values.

the number of memory locations is increased to P^2 . A novel solution was designed with an array of P registers operating as a circular buffer. This is called “inverse C look-up table” (ICLUT) and it saves $P(P - 1)$ memory locations. The first row values of \mathbf{C}^{-1} are stored in the registers. Next, one rotation is applied in each tick of the clock for changing the register’s outputs, as indicated in Fig. 5.

V. IMPLEMENTATION AND SIMULATIONS RESULTS

The SYSDCE architecture was implemented in RTL level using Verilog hardware description language. It is able to process input sequences of length $N = 512$ with $CP = P$ and $P = 8$. It was synthesized and targeted in Xilinx Virtex-5 XC5VLX110T FPGA. Default settings and no “user constraints” were selected in the EDA tool Xilinx ISE v11. No IP core or pre-designed component were used. All signals are represented in signed fixed-point two’s complement and non-rounding scheme was considered.

Table I summarizes the synthesis results for the proposed estimator. The values in the parenthesis in each feature indicate the total of corresponding available resources in the FPGA. The results in Table I reveal a frequency operation of 115.247 MHz with a minimal consumption (except DSP48Es) with respect to the total resources of the FPGA.

It is difficult to compare directly the proposed channel estimators with the others previously presented in [9] and [10], because the differences in technology, paradigms used and testing conditions. In [9] the estimator was implemented under full-software philosophy in TMS320C6713 DSP with a 300 MHz external clock. For that reason, the comparison between SYSDCE and this system was not possible. A hybrid software-hardware FPGA implementation of the DDST receiver is described in [10] but regarding the module corresponding to the channel estimation, only the arithmetic mean was accelerated by a dedicated co-processor. The MVM operation described in (3) was implemented in software. None of the above contributions present results from point of view of the mean square error (MSE) in the channel estimated or signal to quantization noise ratio (SQNR) for fixed-point performance analysis.

Other important parameter of the proposed estimator is the number of cycles required for performing the tasks

TABLE I. SYNTHESIS RESULTS OF THE SYSDCE

Input length (without CP)	(N)	512
Frequency	(MHz)	115.247
Slice registers	(69120)	1370 (1%)
Slice LUTs	(69120)	2587 (3%)
Fully used LUT-FF pairs	(3348)	609 (18%)
Block RAMs	(148)	8 (5%)
DSP48Es	(64)	32 (50%)

TABLE II. CHANNEL ESTIMATOR THROUGHPUTS COMPARISON

Channel estimator	Input length	Cycles/estimation	CT (us)	TP (MS/s)	TP/area (MS/s/slices/)
SYSDCE (Cyclic mean mode)	512	593	5.145	101.06	25.539e3
SYSDCE (Channel estimator mode)	512	610	5.293	98.243	24.828e3
Arithmetic mean coprocessor in [12]	NA	2238	20	NA	NA

estimation. Particularly, the cyclic mean requires

$$cycles_{\hat{y}} = (N + P) + (N_P + P + 1). \quad (10)$$

The first term in (10) corresponds to the input storage phase and the second to the $\frac{N_P}{P}$ MVM operations involved in the cyclic mean task. Furthermore, the number of cycles required for the CIR estimator is:

$$cycles_{\hat{h}} = cycles_{\hat{y}} + 2P + 1. \quad (11)$$

Consider the set of metrics listed in Table II to compare the performance of the SYSDCE system. The processing time (PT) is the time elapsed from beginning of cyclic mean or channel estimation process until its computing has finished. The throughput (TP) per area is another useful metric, a higher value of this ratio indicates that the system implementation is better. As expected, the proposed architecture provides a better performance compared to the arithmetic mean co-processor used in [10]. The validity of the provided architectures is granted by comparing their results with the floating-point simulation golden model programmed in Matlab, in terms of channel estimation error versus signal to noise ratio (SNR). Thereby, the following scenario (similar to used in [6]) was considered. It is considered the transmission of $N = 512$ symbols obtained from a 4-QAM modulator. The channel is randomly generated at each Monte Carlo trial and it is assumed to be Rayleigh with length

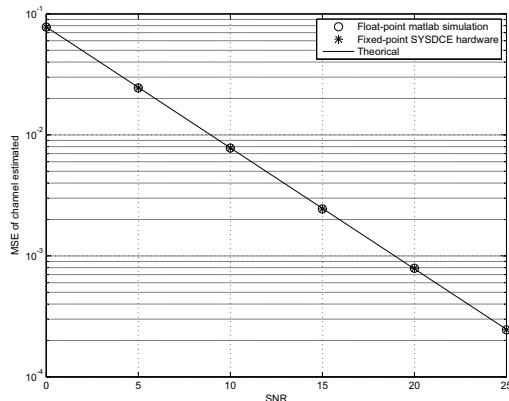


Figure 6. MSE performance of the SYSDCE hardware for 300 Monte Carlo trials.

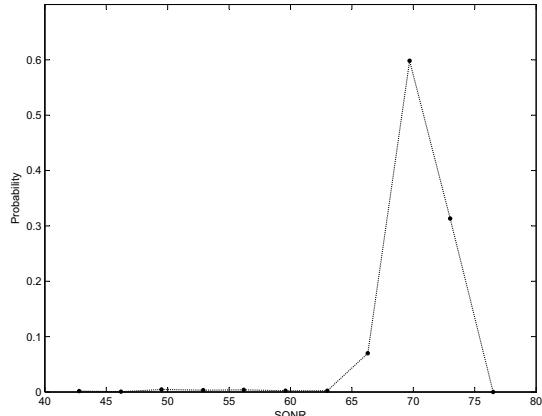


Figure 7. SQNR probability density function of SYSDCE architecture for 300 Monte Carlo trials.

$L = 8$. The power of training sequence is set to 20% of σ_s^2 with a period P equal to L .

The Fig. 6 shows the MSE of channel estimated, which is averaged over 300 Monte Carlo simulation for each value of SNR. Note that the MSE of the hardware estimator is too close to the theoretical line [4] and almost indistinguishable with respect to the golden model. On the other hand, Fig. 7 presents the probability density function (PDF) of the SYSDCE hardware, obtained for the same Monte Carlo trials. Analyzing such PDF, it can be commented that the fixed-point performance in average is about 70 dB in terms of SQNR.

VI. CONCLUSIONS

In this paper, it has been presented an architecture based on AP for DDST channel estimation, which represents the first channel estimator implementation under the full-HW philosophy for a DDST receiver. The architecture presents high throughput and reduced FPGA resources consumption, achieving a good trade-off between performance and area utilization. Also, it is possible to observe a great flexibility and re-usability because the same systolic array is used for two different tasks (operations): cyclic mean and channel estimation. The design can be easily modified (by means of partitioning strategy) for processing channels of different lengths. The validity and performance of SYSDCE approach has been verified by Monte Carlo simulations, where an SQNR of 70 dB in average is achieved while no significant differences with the floating point golden model. The provided results show that ST/DDST concepts can be effectively utilized in a current and future wireless communications standards.

ACKNOWLEDGMENT

This work was supported by PROMEP ITSON-92, CONACYT 164501, Jal-158899 and CONACYT 84559-Y research grants.

REFERENCES

- [1] A. Goljahani, N. Benvenuto, S. Tomasin, and L. Vangelista, “Superimposed sequence versus pilot aided channel estimations for next generation dvb-t systems,” *Broadcasting, IEEE Transactions on*, vol. 55, no. 1, pp. 140 –144, march 2009.
- [2] B. Farhang-Boroujeny, “Pilot-based channel identification: proposal for semi-blind identification of communication channels,” *Electronics Letters*, vol. 31, no. 13, pp. 1044–1046, 22 June 1995.
- [3] S. Haykin and K.J. Ray Liu, *Handbook on Array Processing and Sensor Networks*, Wiley, 2009.
- [4] E. Alameda-Hernandez, D.C. McLernon, A.G. Orozco-Lugo, M.M. Lara, and M. Ghogho, “Frame/training sequence synchronization and dc-offset removal for (data-dependent) superimposed training based channel estimation,” *IEEE Trans. Signal Process.*, vol. 55, no. 6, pp. 2557–2569, June 2007.
- [5] M. Ghogho and Ananthram Swami, “Improved channel estimation using superimposed training,” in *Proc. IEEE 5th Workshop on Signal Processing Advances in Wireless Communications*, 11–14 July 2004, pp. 110–114.
- [6] M. Ghogho, D. McLernon, E. Alameda-Hernandez, and A. Swami, “Channel estimation and symbol detection for block transmission using data-dependent superimposed training,” *IEEE Signal Process. Lett.*, vol. 12, no. 3, pp. 226–229, 2005.
- [7] O. Longoria-Gandara, R. Parra-Michel, M. Bazdresch, and A. G. Orozco-Lugo, “Iterative mean removal superimposed training for siso and mimo channel estimation,” *International Journal of Digital Multimedia Broadcasting*, p. 9, 2008.
- [8] R. Carrasco-Alvarez, R. Parra-Michel, A. G. Orozco-Lugo, and J. K. Tugnait, “Enhanced channel estimation using superimposed training based on universal basis expansion,” *IEEE Trans. Signal Process.*, vol. 57, no. 3, pp. 1217–1222, 2009.
- [9] V. Najera-Bello, “Design and construction of a digital communications system based on implicit training,” M.S. thesis, CINVESTAV-IPN, 2008.
- [10] Fernando Martín del Campo, René Cumplido, Roberto Pérez-Andrade, and A. G. Orozco-Lugo, “A system on a programmable chip architecture for data-dependent superimposed training channel estimation,” *International Journal of Reconfigurable Computing*, vol. 2009, 2009.
- [11] A.G. Orozco-Lugo, M.M. Lara, and D.C. McLernon, “Channel estimation using implicit training,” *IEEE Trans. Signal Process.*, vol. 52, no. 1, pp. 240–254, Jan 2004.
- [12] N. Petkov, *Systolic Parallel Processing*, Elsevier Science Inc., New York, NY, USA, 1992.
- [13] S. Kung, *VLSI Array Processors*, Prentice Hall, 1985.

Apéndice B

Artículos publicados en revistas indexadas

CONFIGURABLE TRANSMITTER AND SYSTOLIC CHANNEL ESTIMATOR ARCHITECTURES FOR DATA-DEPENDENT SUPERIMPOSED TRAINING COMMUNICATIONS SYSTEMS

E. Romero-Aguirre, R. Parra-Michel
*Dept. of Electrical Engineering
CINVESTAV-GDL
Zapopan, Jalisco
eromero, rparra}@gdl.cinvestav.mx*

Roberto Carrasco-Alvarez
*Dept. of Electronic Engineering
UDG-CUCEI
Guadalajara, Jalisco
roberto.carrasco@red.cucei.udg.mx*

A.G. Orozco-Lugo
*Dept. of Electrical Engineering
CINVESTAV-DF
México, D.F.
aorozco@cinvestav.mx*

Abstract—The use of superimposed training (ST) in wireless communications systems has emerged as an alternative in recent years because it saves valuable bandwidth by adding a training periodic sequence to the data signal instead of multiplexing them. However, although ST and one of its variants, known as data-dependent ST (DDST), have been an active research topic, only few physical implementations of that systems have been reported to date. In this work, a configurable ST/DDST transmitter and architecture based on array processors (AP) for DDST channel estimation are presented. Both architectures; designed under full-hardware paradigm; were described using Verilog HDL, targeted in Xilinx Virtex-5 XC5VLX110T and they were compared with existent approaches. The synthesis results showed a FPGA slice consumption of 1% for the transmitter and 3% for the estimator with 160 and 115 MHz operating frequencies respectively. The signal-to-quantization-noise ratio (SQNR) performance of the transmitter is about 82 dB to support 4/16/64-QAM modulation in both operating mode, ST or DDST. A Monte Carlo simulation demonstrates that the mean square error (MSE) of the channel estimator implemented in hardware is practically the same than the one obtained with the floating-point golden model. The high performance and reduced hardware of the proposed architectures leads to the conclusion that the DDST concept can be applied in current communications standards.

Keywords-Channel estimation; data-dependent; superimposed training; FPGA architectures; implicit training; systolic arrays.

I. INTRODUCTION

Presently, there is need to develop communications systems capable of transmitting/receiving various types of information (data, voice, video, etc.) at high speed. Nevertheless, designing these systems is always extremely difficult task and therefore the system must be broken down into several stages each with a specific task. The complexity of each stage is higher when the system operates in a wireless environment because the additional challenges that should be facing due to the complex nature of the channel and its susceptibility to several types of interference.

As it is not possible to avoid the influence of the channel on a transmitted data sent through it, an option is to

characterize the channel parameters with enough precision, so that their effects can be reverted in the receiver. For that reason, channel estimation stage is a key part of any reliable wireless system because a correct channel estimation leads to a reduction of the bit error rate (BER). The channel estimator must deal with multiple phenomena such as multipath propagation and frequency Doppler (due to the mobility of the users). In order to deal with these problems, current communication standards specifies the transmission of pilot signals which are known in the receiver, allowing an ease estimation of the communication channel. The way of transmit such pilot signals can be classified in two major branches: pilot assisted transmission (PAT) —where pilot and data signals are multiplexed in time, frequency, code, space or in a combination of the mentioned domains—and implicit training (IT), a technique proposed recently where the pilot signal is hidden in the data transmitted. PAT is the technique implemented in actual standards, such as, WiMax, WiFi, Bluetooth, etc. It presents the advantage that pilots and data relies on orthogonal subspaces allowing a simple separation of them in the receiver; however, it is necessary to decrease the available bandwidth for data in order to transmit the pilot signal. On the other hand, IT overcomes this problem because all the time, data and pilot signal are transmitted; nevertheless, it leads to a transmission of such signals into non-orthogonal subspaces. Despite the aforementioned, IT has been recognized as a feasible alternative for future communication standards [1].

The simplest form to carry out IT is to add (superimpose) the pilot signal to the data. This approach is known as superimposed training (ST), first proposed in [2] and enhanced by diverse authors which results are summarized in [3, Ch. 6]. In [4], [5], [6], [7], [8] was presented a refinement of ST known as data-dependent superimpose training (DDST), this technique makes it possible to null the interference of data during the estimation process via the addition of a new training sequence—which depends on the transmitted data—together with the data and the ST sequence.

Because the benefits that offer ST/DDST, it is necessary to develop efficient implementations of these algorithms. Although these techniques have been widely studied, to this point, there exist few reported practical implementations in the literature. In fact, almost all of them are approximations based on floating-point and software. In [9], the algorithms are programmed in a digital signal processor (DSP) for a low rate communication system, while in [10] the proposed implementation is developed into a embedded microprocessor with hardware accelerators inside of a FPGA. At ReConFig 2011, we have presented a full-hardware architecture — with high throughput, low hardware consumption and high degree of re-usability— for the channel estimation stage of an ST/DDST receiver [11]. Its novelty consisted of that a systolic array processors (AP) was used for performing the entire estimation process instead of two separated signal processing modules. In this paper, we present a extended version of that paper, where a hardware-efficient architecture for configurable ST/DDST transmitter that supports 4/16/64-QAM constellations is used to complement the results presented in [11], because now, all transmitted data —in each Monte Carlo trial— are generated by the proposed transmitter hardware instead of the transmitter simulation model programmed in Matlab.

The rest of the paper is organized as follows. Section II presents the system model being considered, the ST/DDST transmitter structure, the channel estimation algorithm and the cyclic mean reformulation onto systolic APs. Section III describes in detail the full-hardware architectures for the configurable ST/DDST transmitter. Section IV proposes an architecture based on SA processor for the DDST channel estimator. In Section V, the performance evaluation of the proposed architectures is carried out. Conclusions are set down in Section VI.

Notation: Lowercase (uppercase) bold letters denote column vectors (matrices). Operators $(\mathbf{A})^H$, $(\mathbf{A})^T$ and $(\mathbf{A})^{-1}$, denote the Hermitian, transpose and inverse operations of matrix \mathbf{A} . $\mathbf{1}_n$ represents a column vector of length n with all its elements equal to one, similarly, $\mathbf{0}_n$ represents an all-zeros column vector of length n . \mathbf{I}_n is the identity matrix of size $n \times n$. $[\mathbf{a}]_k$ denotes the k -th element of vector \mathbf{a} , $[\mathbf{a}]_{m:n}$ denotes a vector conformed with the elements of \mathbf{a} as follows: $[[\mathbf{a}]_m, [\mathbf{a}]_{m+1}, \dots, [\mathbf{a}]_n]^T$. \otimes represents Kronecker product. Finally, $E(\cdot)$ represents the expectation operator.

II. SYSTEM MODEL

This Section is devoted to introduce the DDST algorithm mentioned previously. Suppose a single carrier, baseband communication system based on DDST as the presented in Fig. 1. The transmitted signal $x(k)$ is conformed by the sum of the data sequence $b(k)$, the training sequence $c(k)$ and the data dependent training sequence $e(k)$. The index k helps to enumerate the samples of such signals which are transmitted at a rate equal to $1/T$. $c(k)$ is a periodic sequence with

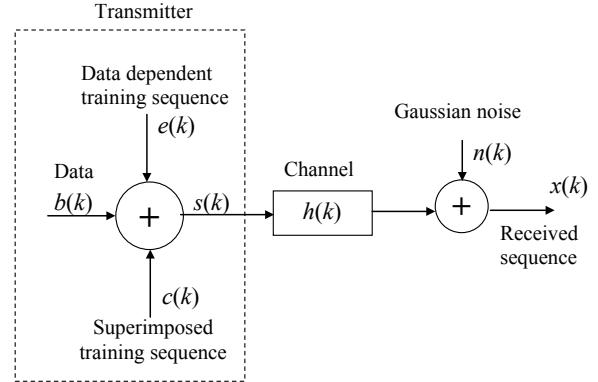


Figure 1. Digital communication system model considered.

period equal to P and power equal to σ_c^2 [12]. It is assumed that the data sequence is a zero-mean, stationary stochastic process with power equal to σ_b^2 where the symbols of such process come from a equiprobable alphabet. The sequence $e(k)$ is constructed as mentioned in [5]. $s(k)$ is propagated through the communication channel $h(k)$ whose time impulse response is conformed by the convolution of the system filters and the propagation medium impulse responses (all of them assumed to be time-invariant). Such channel can be modeled as a finite impulse response (FIR) filter with L time-invariant coefficients as much. Finally, the distorted signal by the channel is contaminated with the noise $n(k)$ for conforming the received signal $x(k)$. $n(k)$ is a zero-mean white Gaussian noise, which possess variance equal to σ_n^2 . It is assumed the transmission of blocks of N symbols, which are preceded by a cyclic prefix of length $CP \geq L$. It is also assumed perfect block synchronization, which allows to fix $P = L$. For ease of implementation, it is assumed that N is a multiple of P and P is a power of two.

Thus, the received signal after removing the cyclic prefix can be expressed in a matrix form as:

$$\mathbf{x} = \mathbf{H}(\mathbf{b} + \mathbf{c} + \mathbf{e}) + \mathbf{n}, \quad (1)$$

where \mathbf{H} is a circulant matrix whose first row is given by $[\mathbf{h}^T, \mathbf{0}_{N-L}^T]$ where \mathbf{h} is a vector containing the coefficients of the channel impulse response (CIR). Similarly, \mathbf{x} , \mathbf{b} , \mathbf{c} , \mathbf{e} and \mathbf{n} are vectors equal to $[\mathbf{x}]_k = x(k)$, $[\mathbf{b}]_k = b(k)$, $[\mathbf{c}]_k = c(k)$, $[\mathbf{e}]_k = e(k)$ and $[\mathbf{n}]_k = n(k)$ respectively, with $0 \leq k \leq N - 1$.

A. Digital transmitter with ST/DDST included

Fig. 2 depicts the discrete-time baseband block diagram of the (data-dependent) superimposed training transmitter. This is a modified version of the IT transmitter presented in [3]. From Fig. 2, it can be noted that the key component of the transmitter is the sequence transformation block. It serves to implicitly embed the training sequences onto data

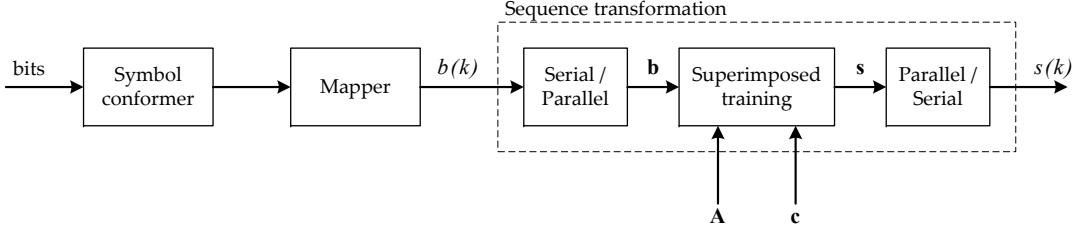


Figure 2. Block diagram of the digital baseband (data-dependent) superimposed training transmitter

sequence \mathbf{b} by the affine transformation expressed as

$$\mathbf{s} = \mathbf{Ab} + \mathbf{c}, \quad (2)$$

where \mathbf{s} represent the complex baseband discrete-time transmitted signal, \mathbf{A} is a precoding matrix and \mathbf{c} refers to vector obtained by replicating N_P times a one period of the training signal \mathbf{c}_{OCl} of size P , i.e.

$$\mathbf{c} = \mathbf{1}_{N_P} \otimes \mathbf{c}_{OCl}, \quad (3)$$

where $N_P = \frac{N}{P}$, $[\mathbf{c}_{OCl}]_n = c_{OCl}(n)$ and such sequence is given by [12]:

$$c_{OCl}(n) = \sigma_c e^{j\frac{\pi}{P}(n(n+v))} \quad (4)$$

with $v = 1$ when P is odd, $v = 2$ if P is even, and $n = 0, \dots, P-1$.

The precoding matrix allows to modify the training technique according to:

$$Training = \begin{cases} \mathbf{A} = \mathbf{I}_N, \mathbf{c} \neq 0 & \text{for ST} \\ \mathbf{A} = \mathbf{I}_N - \mathbf{G}, \mathbf{c} \neq 0 & \text{for DDST} \end{cases} \quad (5)$$

with

$$\mathbf{G} = \mathbf{1}_{N_P \times 1} \otimes \mathbf{K}, \quad (6a)$$

and

$$\mathbf{K} = \mathbf{1}_{1 \times N_P} \otimes \mathbf{I}_P, \quad (6b)$$

where \mathbf{G} and \mathbf{K} are matrices of sizes $N \times N$ and $P \times N$ respectively.

In the DDST case, the N -length vector \mathbf{e} containing the data-dependent sequence (DDS) can be obtained from (2) and using (5)-(6b) as

$$\mathbf{e} = -\frac{1}{N_P} \mathbf{Gb}. \quad (7)$$

B. Channel estimation using DDST

It is possible to observe that due to the periodicity of $c(k)$, $s(k)$ will have a periodic signal embedded with a period equal to P . Taking advantage of this characteristic, an estimate of the cyclic mean of the received signal is utilized for performing the estimate of the channel. Such cyclic mean estimator can be defined as:

$$\mathbf{y} = \mathbf{Jx}, \quad (8)$$

where \mathbf{y} is a column vector of length P whose elements are the estimated of the cyclic mean of \mathbf{x} and \mathbf{J} is given by:

$$\mathbf{J} = \frac{1}{N_P} (\mathbf{1}_{N_P}^T \otimes \mathbf{I}_P). \quad (9)$$

According to [4], the estimation of the CIR is given by:

$$\hat{\mathbf{h}} = \mathbf{\Gamma y}, \quad (10)$$

where $\hat{\mathbf{h}}$ is a vector containing the estimated CIR coefficients, $\mathbf{\Gamma}$ is a matrix formed by the first L rows of \mathbf{C}^{-1} and \mathbf{C} is a circulant matrix of size $P \times P$ formed by vector $[c(0), c(1), \dots, c(P-1)]^T$.

C. Cyclic mean algorithm using array processors and partitioning

The next analysis describes how the cyclic mean is obtained using a systolic array that computes a matrix-vector multiplication (MVM). Consider (8), where it is not possible to perform directly the MVM operation due to the Kronecker product involved. To avoid this cumbersome operator, the same equation can be reformulated as follows:

$$\mathbf{y} = \frac{1}{N_P} (\mathbf{x} \mathbf{1}_{N_P}), \quad (11)$$

where \mathbf{x} is a matrix of size $P \times N_P$ which is defined as follow:

$$\mathbf{x} = \begin{bmatrix} [\mathbf{x}]_0 & [\mathbf{x}]_P & [\mathbf{x}]_{2P} & \cdots & [\mathbf{x}]_{(N_P-1)P} \\ [\mathbf{x}]_1 & [\mathbf{x}]_{P+1} & [\mathbf{x}]_{2P+1} & \cdots & [\mathbf{x}]_{(N_P-1)P+1} \\ [\mathbf{x}]_2 & [\mathbf{x}]_{P+2} & [\mathbf{x}]_{2P+2} & \cdots & [\mathbf{x}]_{(N_P-1)P+2} \\ \vdots & \vdots & \vdots & & \vdots \\ [\mathbf{x}]_{P-1} & [\mathbf{x}]_{2P-1} & [\mathbf{x}]_{3P-1} & \cdots & [\mathbf{x}]_{N_P P-1} \end{bmatrix}. \quad (12)$$

An architecture based on AP for computing (11) would be impractical from point of view of hardware consumption because it will need N_P processor elements (PEs). This problem is known as *problem-size dependent array* where the algorithm requires a systolic AP whose size depends on the complexity of the problem to be solved. However, it is possible to map the cyclic mean algorithm to a systolic AP of a smaller size using the *partitioning* method [13, Ch. 12]. Considers \mathbf{x} to be partitioned in blocks of size chosen to match a systolic array size P then (12) becomes

$$\mathbf{x} = [\mathbf{B}_0 | \mathbf{B}_1 | \cdots | \mathbf{B}_{\frac{N_P}{P}-1}], \quad (13)$$

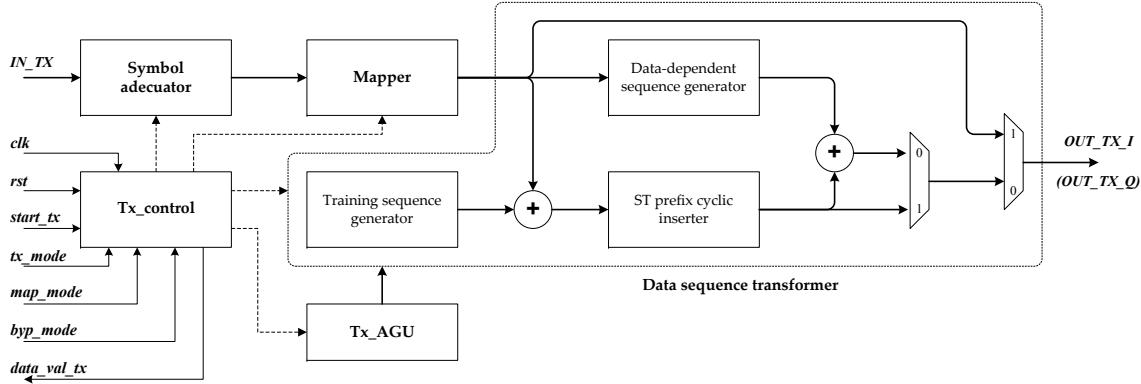


Figure 3. Digital architecture of the configurable ST/DDST transmitter.

where

$$\mathbf{B}_i = \begin{bmatrix} [\mathbf{x}]_{iP^2} & [\mathbf{x}]_{iP^2+P} & \cdots & [\mathbf{x}]_{iP^2+(P-1)P} \\ [\mathbf{x}]_{iP^2+1} & [\mathbf{x}]_{iP^2+P+1} & \cdots & [\mathbf{x}]_{iP^2+(P-1)P+1} \\ [\mathbf{x}]_{iP^2+2} & [\mathbf{x}]_{iP^2+P+2} & \cdots & [\mathbf{x}]_{iP^2+(P-1)P+2} \\ \vdots & \vdots & & \vdots \\ [\mathbf{x}]_{iP^2+P-1} & [\mathbf{x}]_{iP^2+2P-1} & \cdots & [\mathbf{x}]_{iP^2+P^2-1} \end{bmatrix},$$

for $i = 0, \dots, \frac{N_P}{P} - 1$.

In similar way, $\mathbf{1}_{N_P}$ is partitioned in $\frac{N_P}{P}$ unitary vectors $\mathbf{1}_P$. Substituting (13) in (11), the cyclic mean with *partitioning* is concisely expressed as:

$$\mathbf{y} = \frac{1}{N_P} (\mathbf{B}_0 \mathbf{1}_P + \mathbf{B}_1 \mathbf{1}_P + \cdots + \mathbf{B}_{\frac{N_P}{P}-1} \mathbf{1}_P). \quad (14)$$

Therefore, the array of PEs will process a one pair of \mathbf{B} and $\mathbf{1}_P$ blocks after another in a sequential manner together with partial results.

III. A CONFIGURABLE ST/DDST TRANSMITTER ARCHITECTURE

Considering the explained in Section II-A, the architecture shown in Fig. 3 is proposed for the transmitter. It is composed of the five hardware modules: the symbol adeuator, the mapper, the data sequence transformer, the Tx_control and the Tx_AGU. The reconfigurability feature of the architecture allows to switch between two operating mode: ST or DDST; in order to send data blocks with a cyclic prefix attached. In both modes, the transmitter hardware supports 4/16/64-QAM constellations.

In the next subsections, additional details about the main transmitter modules will be described.

A. Symbol adeuator

The design of this module is widely conditioned by the features of the mapper. By early account, a key aspect exploited in the mapper design, it consists of the fact that the

4-QAM and 16-QAM constellations are contained in Grey-coded 64-QAM one, as shown in Fig. 4. For that reason, the symbol adeuator is necessary because not all the same point-numbers in the three constellations are mapped to the same complex symbol output. For example, while the point-number 2 of the 4-QAM constellation is mapped to $-1 + j$ symbol, 16-QAM will map this point-number to $3 - 3j$ and 64-QAM will map to $3 + 5j$.

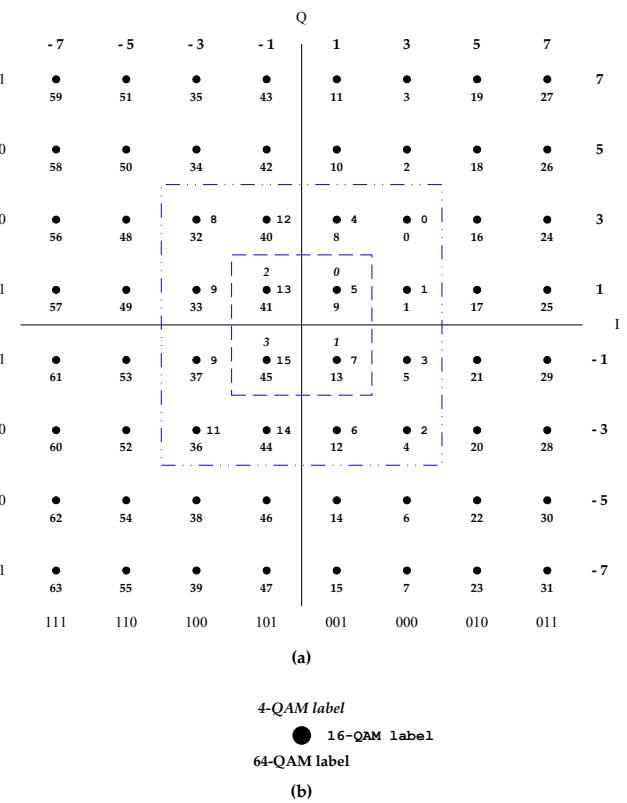


Figure 4. Grey-coded 64-QAM constellation used in the mapper module; a) The 4-QAM and 16-QAM constellations are delimited with a dashed lines, b) Label guide for identifying the constellation point-number.

B. Mapper

As stated in Section 3 III-A, in the mapper design is only required the 64-QAM constellation. In this work, an memory-efficient scheme is propose to build that constellation, whose eight possible values (1, 3, 5, 7, -1, -3, -5, -7) of the I and Q axes are stored in the *constellation LUT*. Additionally, the mapper has to normalize the complex symbols based on two criteria: the constellation order and power assigned to each of the sequences involved. Thus, a normalization constant *Norm_Mapp_Cte* that combines the two criteria is given by

$$\text{Norm_Mapp_Cte} = \sigma_b \times \text{Norm_QAM_Cte} \quad (15)$$

where

$$\text{Norm_QAM_Cte} = \begin{cases} 1/\sqrt{2}, & \text{for 4-QAM} \\ 1/\sqrt{10}, & \text{for 16-QAM} \\ 1/\sqrt{42}, & \text{for 64-QAM,} \end{cases} \quad (16)$$

with

$$\sigma_b^2 + \sigma_c^2 = 1 \text{ for ST} \quad (17a)$$

and

$$\sigma_{b+e}^2 + \sigma_c^2 = 1 \text{ for DDST.} \quad (17b)$$

The mapper architecture designed is depicted in Fig. 5. The *constellation LUT* was implemented with a dual-port ROM with eight memory locations depth. On the contrary, in the *normalization LUT*, the ROM depth was 16 locations.

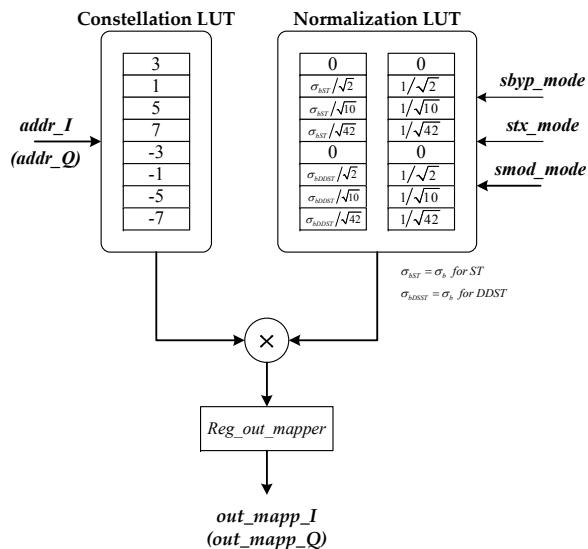


Figure 5. Hardware-efficient 4/16/64-QAM mapper with ST/DDST incorporated.

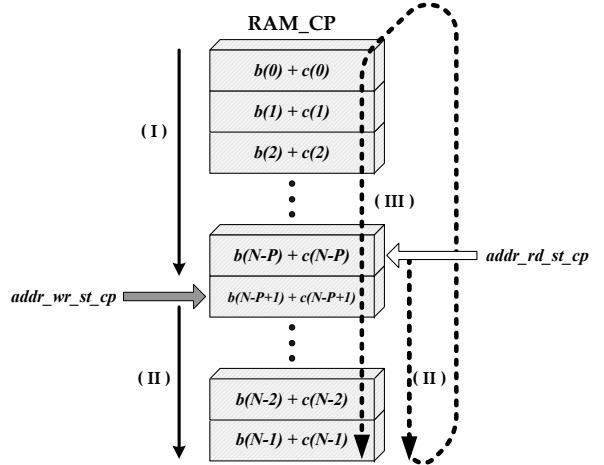


Figure 6. ST cyclic prefix generation and insertion process

C. Data sequence transformer

The data sequence transformer is the greater complexity module of the transmitter. Thus, its design was broken down into three submodules, whose individual architectures are described in the following paragraphs.

1) *Training sequence generator*: Analyzing (4), it can notice that the parameters σ_c^2 , N and P , needed to generate the training sequence are known in advance and they remain constants during the transmitter operating. Hence, the P values of the training sequence can be calculated off-line, quantized and stored in a LUT. This LUT is read N_P times in order to expand the training sequence length, as indicated in (3), and it can be superimposed, element by element, with the data sequence by the complex adder.

2) *ST cyclic prefix insertion submodule*: There are several problems to arise because the way in that the prefix cyclic is generated and its position where it is attached in the ST sequence.

- Since the prefix cyclic is conformed of the last P data of the sequence ST, it can only be generated from this sequence until it has been completely processed.
- Given that, in all the $N + P$ data to be transmitted, the first P data correspond to the cyclic prefix, it is necessary to use a memory buffer in order to store the remaining N data (ST sequence) and thus prevent data loss.

A dual-port RAM (*RAM_CP*) of depth N was used for the ST cyclic prefix insertion submodule designing. The *RAM_CP* have two independent address buses; one for data reading (*addr_rd_st_cp*) and writing (*addr_wr_st_cp*): This feature allows to read and write data simultaneously to/from the *RAM_CP*. The process for generating and attaching a cyclic prefix in the ST sequence, can be summarized in the following steps (Fig. 6).

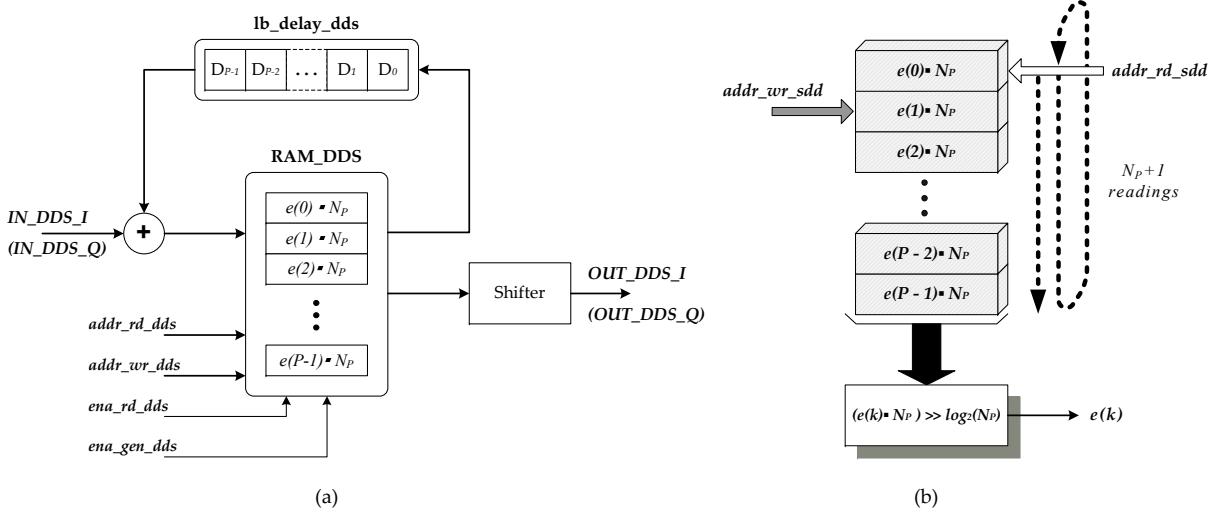


Figure 7. Data-dependent sequence generator submodule; a) Simplified architecture, b) Pictorial representation of the $e(k)$ sequence generation

- I. When the $(N - P + 1)th$ datum is stored in RAM_CP , the previous datum stored is addressed by $addr_rd_st$ bus.
- II. During P clock cycles the ST sequence storing and reading take place in the RAM_CP .
- III. The ST sequence storing in the RAM_CP is stopped. However the data reading will continue for N cycles.

3) *Data-dependent sequence generator*: The operation of this submodule is based on (6a)-(7), which implies to compute two high demanding processing operations: a MVM and the Kronecker product. Moreover, similar to the cyclic prefix insertion case, the DDS can only be generated from data sequence $b(k)$ until it has been completely processed. In consequence, the following adaptations should be made to the original equations in order to ease its mapping-to-hardware process.

- I. The $b(k)$ sequence is rearrange into a matrix of size $P \times N_P$, according to:

$$[\mathbf{B}^T]_{i,j} = [\mathbf{b}]_{iP+j} = b(iP + j) \quad (18)$$

for $i = 0, \dots, N_P - 1$ and $j = 0, 1, \dots, P - 1$.

- II. The mean of the each rows of the matrix \mathbf{B} is obtained.
- III. The P mean results are replicated $N_P + 1$ times in order to obtain the \mathbf{e} vector and P data for DDST cyclic prefix purposes.

Fig. 7 shows the hardware architecture of DDS generator. Its novel design avoids the $b(k)$ sequence rearranging by the loop-back shift register lb_delay_dds . This register generates a P symbols delay in order to align the data for each \mathbf{B} matrix rows. So, the data rows can be added “on-the-fly” by the complex adder without the data input stream is stopped. The sum results are stored in the RAM_DDS , after its entire contents is read $N_P + 1$ times and each datum is divided by

N_P in the *shifter* block. Finally, the results are sent to the DDS generator outputs.

IV. SYSTOLIC CHANNEL ESTIMATOR ARCHITECTURE

This Section introduces an architecture for the DDST-based channel estimation process. Its design is based on MVM operation, which is carried out in systolic way into AP. The main idea in the system design is to reuse the same systolic array for computing the cyclic mean of the received data. The proposed architecture, called in this paper as “systolic DDST channel estimator” (SYSDCE) is depicted in Fig. 8a. It can be identified four functional units: a modified systolic matrix-vector multiplier (MSYSMVM), a data input feeder (DATINF), an inverse C look-up table (ICLUT) and a control unit (CU). Broadly speaking, the SYSDCE operation can be divided into three phases: input sequence storage, cyclic mean compute and CIR estimate.

As soon as the *start* signal is asserted, an $N + P$ data samples (vector \mathbf{x} and cyclic prefix, respectively) can be read from the input port *IN*. After excluding the samples corresponding to the cyclic prefix, the rest of samples are rearranged and stored in the memory bank of DATINF. When this process is finished, the CU configures the MSYSMVM unit and during N_P cycles it reads P parallel data per cycle from DATINF and computes the cyclic mean \mathbf{y} . Once this phase is finished, the obtained vector \mathbf{y} together with ICLUT data are fed to the MSYSMVM again for performing the product expressed in (10). Finally, after $P + 1$ cycles, the *done* flag is asserted and one by one the coefficients of the channel estimated $\hat{\mathbf{h}}$ are sent to the bus *H_OUT*. It is worth mentioning that the SYSDCE can be configured to compute only the cyclic mean if *mode* input control signal has been set to zero. In this case, the *cm_flag* out is asserted to indicate that a valid results are available in *CM_OUT* bus. Thus,

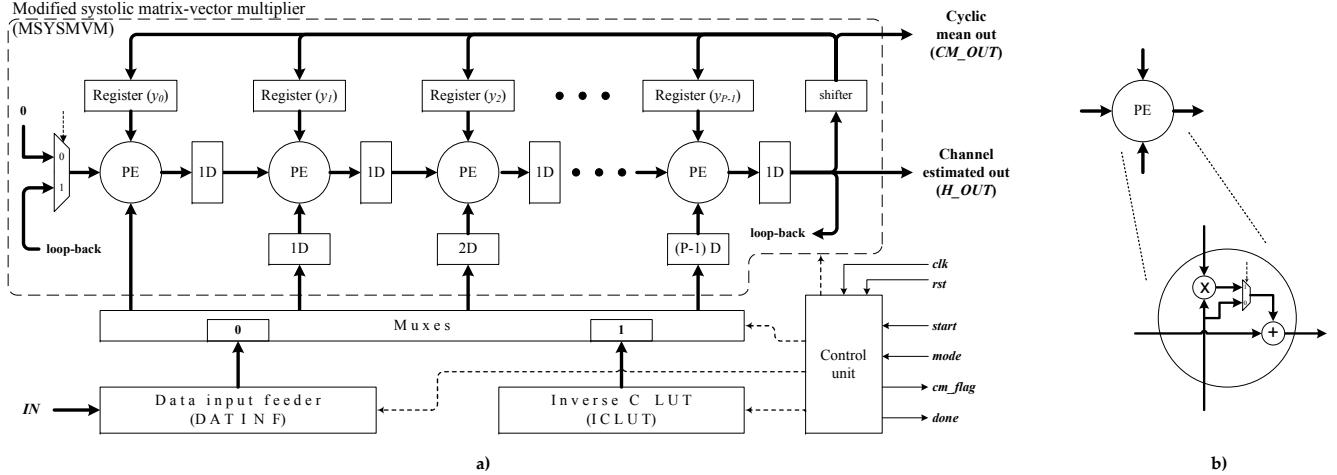


Figure 8. Systolic channel estimator for DDST receiver: a) simplified architecture, b) the PE module.

the channel estimator is prepared for another data sequence processing. A deeper explanation about each component of the SYSDCE architecture will be given in the subsections.

A. Modified systolic matrix-vector multiplier (MSYSMVM)

The fundamental operation to perform by SYSDCE is a matrix-vector multiplication which is high time-processing demanding. The hardware design for solving this operation is the most critical part in the architecture. The obvious strategy for accelerate MVM consists of in compute as many operations as possible, with the penalty of a great consumption of FPGA resources. Therefore, this paper proposes a modification of the systolic MVM presented in [14, Ch. 3] in order to obtain a good performance with reasonable resources consumption. This modification allows to compute the cyclic mean using *partitioning* method with the same systolic array reported. Fig. 8b shows the processor element (PE), which is the atomic digital signal processing module in MSYSMVM. It processes three flows: the data flow from the ICLUT or DATINF, the input registers values and the data produced by the previous adjacent PE.

In the MSYSMVM design was considered that the number of PEs needed (AP size) is P , which matches with the dimensions of matrix Γ and vector \mathbf{y} respectively. The projection vector $\mathbf{d} = [1 \ 0]^T$ (see details in [14]) was used with a vector schedule $\mathbf{s} = [1 \ 1]^T$. The pipelining period for this design is equal to 1 and the computing time for the full MVM is $2P - 1$ clock periods.

For computing the cyclic mean using the MSYSMVM module, the original structure of PE was modified with an additional multiplexer. For that reason, the PE can perform all trivial multiplications by bypassing the data from the input of the complex multiplier directly to the complex adder.

B. Data input feeder (DATINF)

Similar to almost any systolic array, the MSYSMVM needs that the data, —which will be fed to each of its PEs— must be given in a defined order before to process it. In the proposed approach, the module DATINF is responsible for performing this task. It is made up of an array of P memories, each with a depth of N_P , organized as a memory bank as shown in Fig. 9. DATINF reads $N + P$ data from IN bus, it identifies and removes the first P data corresponding to CP. Subsequently, this module rearranges this sequence (correspondence to $x(k)$) in $\frac{N_P}{P}$ blocks of size $P \times P$ in order to form $\mathbf{B}_0, \mathbf{B}_1, \dots, \mathbf{B}_{\frac{N_P}{P}}$. Therefore, the N stored data can be viewed as a $N_P \times P$ matrix, where each individual memory in the bank stores one column of each block and the blocks are stored consecutively one after the other, as depicted in Fig. 9.

Each datum of the input sequence \mathbf{x} has associated three addresses that defines its location inside the memory bank: block number (blk_num), memory number (mem_num) and memory address (mem_addr). The DATINF must generate these addresses using the following expressions:

$$blk_num = \left\lfloor \frac{k \times N_P}{N \times P} \right\rfloor, \quad k = 0, 1, \dots, N - 1, \quad (19a)$$

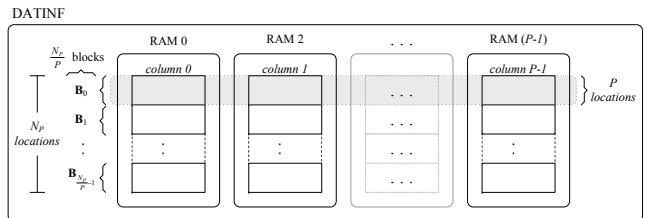


Figure 9. Data block organization in the DATINF

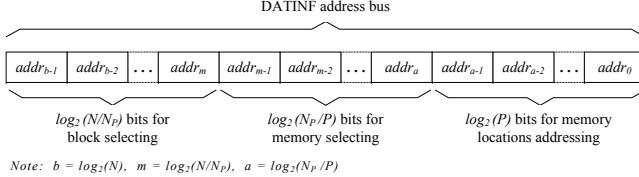


Figure 10. Hard-wired addressing for memory bank.

$$mem_num = \left\lfloor \frac{k \times N_p - blk_num \times N \times P}{N} \right\rfloor, \quad (19b)$$

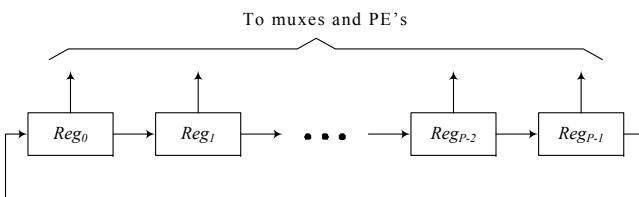
$$mem_addr = (k \bmod P) + (P \times blk_num), \quad (19c)$$

where k is the k -th element of \mathbf{x} and $\lfloor \cdot \rfloor$ denotes the *floor* operator.

In order to minimize the hardware consumption, a “hard-wired” addressing approach was built for the memory bank. As shown in Fig. 10, the $\log_2(N)$ bits corresponding to the DATIN address bus are split into three parts. The first $\log_2(\frac{N}{N_p})$ most significant bits (MSB) are used for block selecting, the next $\log_2(\frac{N_p}{P})$ MSB are used to select a particular memory in the bank and the remaining $\log_2(P)$ bits are used to individually address each of the locations in the selected memory.

C. Inverse C look-up table (ICLUT)

The values of the circulant matrix \mathbf{C}^{-1} are constants that can be precomputed once off-line and stored in a LUT. Only the values of the first column are necessary because the remaining columns are shifted versions of the first one. Consequently, the ROM location’s number required for the LUT is just P . If traditional design is used, then the LUT will be designed with a multi-port ROM of P locations, but it will be synthesized by the employed compiler tool as an array of P single port ROMs. Therefore, the number of



	C_0^{-1}	C_{p-1}^{-1}	C_{p-2}^{-1}	\dots	C_2^{-1}	C_1^{-1}
$clk = 1$	C_0^{-1}	C_{p-1}^{-1}	C_{p-2}^{-1}	\dots	C_2^{-1}	C_1^{-1}
$clk = 2$	C_1^{-1}	C_0^{-1}	C_{p-1}^{-1}	\dots	C_3^{-1}	C_2^{-1}
$clk = 3$	C_2^{-1}	C_1^{-1}	C_0^{-1}	\dots	C_4^{-1}	C_3^{-1}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
$clk = P-1$	C_{p-2}^{-1}	C_{p-3}^{-1}	C_{p-4}^{-1}	\dots	C_0^{-1}	C_{p-1}^{-1}
$clk = P$	C_{p-1}^{-1}	C_{p-2}^{-1}	C_{p-3}^{-1}	\dots	C_1^{-1}	C_0^{-1}
	Reg_0	Reg_1	Reg_2		Reg_{p-2}	Reg_{p-1}

Figure 11. Simplified architecture of inverse C look-up table (ICLUT) and its corresponding outputs values.

memory locations is increased to P^2 . A novel solution was designed with an array of P registers operating as a circular buffer. This is called “inverse C look-up table” (ICLUT) and it saves $P(P-1)$ memory locations. The first row values of \mathbf{C}^{-1} are stored in the registers. Next, one rotation is applied in each tick of the clock to change the register’s outputs, as indicated in Fig. 11.

V. RESULTS

In this Section, the proposed architectures are evaluated. First, the hardware utilization and throughput of the ST/DDST transmitter implementation are presented. After, its functional performance from the point of the signal-to-quantization-noise ratio (SQNR) is analyzed. Next, the FPGA resources consumption and throughput of the SYS-DCE implementation is obtained. Finally, the SYS-DCE functional results specified in terms of the MSE of the channel estimated and SQNR performance are carried out by Monte Carlo simulations and using the transmitter hardware in DDST mode.

A. Implementation and simulation of the transmitter

The configurable ST/DDST transmitter architecture was implemented in RTL level using Verilog hardware description hardware. It is able to transmits ST or DDST data blocks of length N with $CP = P$. The power of training sequence is set to $0.2\sigma_s^2$ with a period $P = 8$. The configurable transmitter was synthesized and targeted in Xilinx Virtex-5 XC5VLX110T FPGA. Default settings and no “user constraints” were selected in the EDA tool Xilinx ISE v11. No IP core or pre-designed component were used. All signals are represented in signed fixed-point two’s complement and non-rounding scheme was considered.

Table I summarizes the synthesis results for the proposed ST/DDST transmitter. Analyzing this table it can be noted a operating frequency of 160 MHz with a symbolic FPGA resource utilization. So, it is clear that excellent area-frequency balance is achieved.

On the other hands, it is difficult to compare directly the proposed transmitter and channel estimator with the others previously presented in [9] and [10], because the differences in technology, paradigms used and testing conditions. In [9], DDST communication system was implemented under

Table I
SYNTHESIS RESULTS OF THE ST/DDST TRANSMITTER

FPGA resource	Used	Available	Utilization
Frequency	160.12	MHz	--
Slice registers	141	69120	<1%
Slice LUTs	437	69120	<1%
Fully used LUT-FF pairs	134	444	30%
IOBs	46	640	7%
BRAMs	4	148	2%

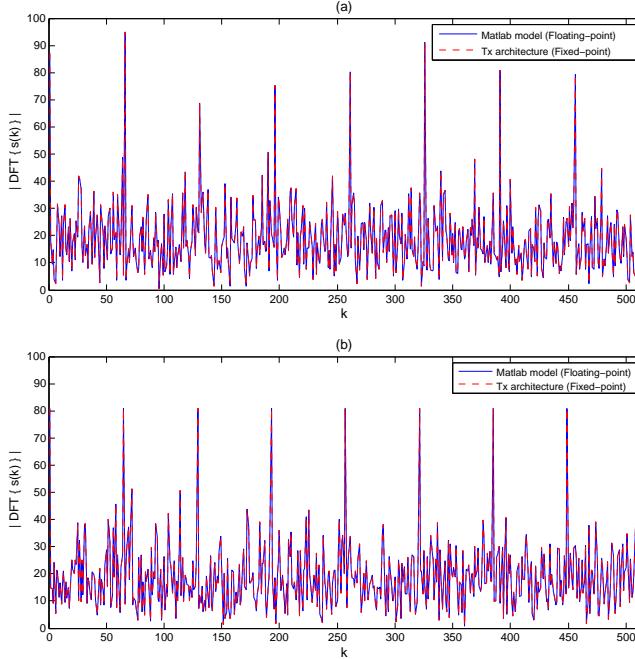


Figure 12. Discrete Fourier transform of the $s(k)$ sequence generated by the transmitter architecture; a) ST mode, b) DDST mode

full-software philosophy in TMS320C6713 DSP with a 300 MHz external clock. A hybrid software-hardware FPGA implementation of the DDST receiver is described in [10]. In both DDST implementations mentioned, the comparison against our transmitter was not possible. In the former because the transmitter was full-software based and the latter only the DDST receiver was implemented.

The transmitter operating validity is presented in Fig. 12. The first graph (Fig. 12a) shows clearly that the transmitter hardware has embedded the training sequence $c(k)$ into $b(k)$. It can be noted, that the data sequence energy is spread in all frequency components. In contrast, the training sequence energy are only concentrated in P equispaced frequency components. Similar behavior occurs in the DDST mode (Fig. 12b), but now the pilots signals also have the same energy. This is unequivocal proof that the transmitter architecture is properly superimposing $c(k)$ and $e(k)$ into $b(k)$.

The SQNR obtained for 100 Monte Carlo trials is monitored, in order to quantify the difference between the $s(k)$ sequence obtained with the hardware transmitter compared with the floating-point transmitter golden model. Thus, the histogram of Fig. 13 represents concisely the results of this test. The most of the occurrence are concentrated in 84 dB.

B. Implementation and simulation of the channel estimator

The SYSDCE architecture was implemented using the same considerations and design parameters of the transmitter. Also, the systolic channel estimator was synthesized and

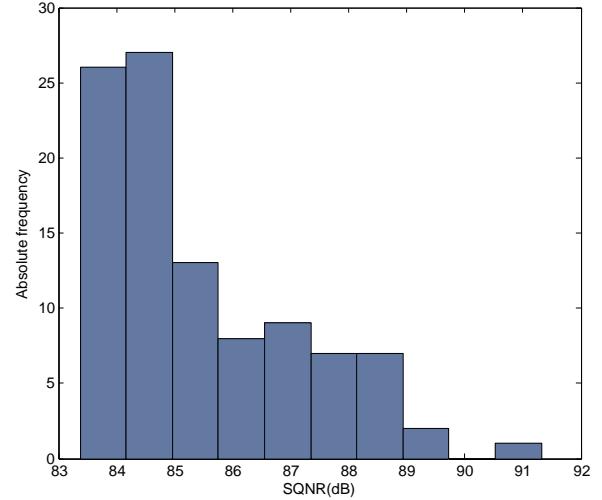


Figure 13. SQNR performance histogram of the ST/DDST architecture for 100 Monte Carlo trials

targeted in the same FPGA.

Table II summarizes the synthesis results for the proposed estimator. The values in the parenthesis in each feature indicate the total of corresponding available resources in the FPGA. The results in Table I reveal a frequency operation of 115.247 MHz with a minimal consumption (except DSP48Es) with respect to the total resources of the FPGA.

Againly, it was not possible to compare the SYSDCE against the existent approaches. In [10], the module corresponding to the channel estimation, only the arithmetic mean was accelerated by a dedicated coprocessor. In this work, the input sequence length was assumed (but it did not explicitly mentioned) to $N=512$ symbols. The MVM operation described in (9) was implemented in software. Also, no results —from point of view of the mean square error (MSE) in the channel estimated or SQNR performance—are presented.

Other important parameter of the proposed estimator is the number of cycles required for performing the tasks estimation. Particularly, the cyclic mean requires

$$cycles_{\hat{y}} = (N + P) + (N_P + P - 1). \quad (20)$$

Table II
SYNTHESIS RESULTS OF THE SYSDCE

Input length (without CP)	(N)	512
Frequency	(MHz)	115.247
Slice registers	(69120)	1370 (1%)
Slice LUTs	(69120)	2587 (3%)
Fully used LUT-FF pairs	(3348)	609 (18%)
Block RAMs	(148)	8 (5%)
DSP48Es	(64)	32 (50%)

Table III
CHANNEL ESTIMATOR THROUGHPUTS COMPARISON

Channel estimator	Input length	Cycles/estimation	CT (us)	TP (MS/s)	TP/area (MS/s/slices)
SYSDEC (Cyclic mean mode)	512	591	5.128	101.40	25.625e3
SYSDEC (Channel estimator mode)	512	606	5.258	98.91	24.996e3
Arithmetic mean coprocessor in [10]	512	2238	20	26.39	NA

The first term in (20) corresponds to the input storage phase and the second to the $\frac{N_p}{P}$ MVM operations involved in the cyclic mean task. Furthermore, the number of cycles required for the CIR estimator is:

$$cycles_{\hat{h}} = cycles_{\hat{s}} + 2P - 1. \quad (21)$$

Consider the set of metrics listed in Table III to compare the performance of the SYSDEC system. The processing time (PT) is the time elapsed from beginning of cyclic mean or channel estimation process until its computing has finished. The throughput (TP) per area is another useful metric, a higher value of this ratio indicates that the system implementation is better. As can be seen from Table III, the proposed architecture provides a better performance compared to the arithmetic mean coprocessor used in [10].

The validity of the provided architectures is granted by comparing their results with the floating-point simulation golden model programmed in Matlab, in terms of channel estimation error versus signal-to-noise ratio (SNR). Thereby,

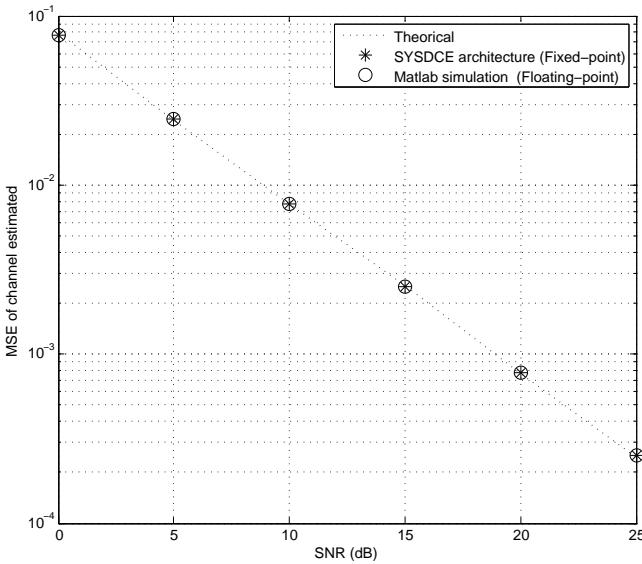


Figure 14. MSE performance of the SYSDEC hardware for 300 Monte Carlo trials.

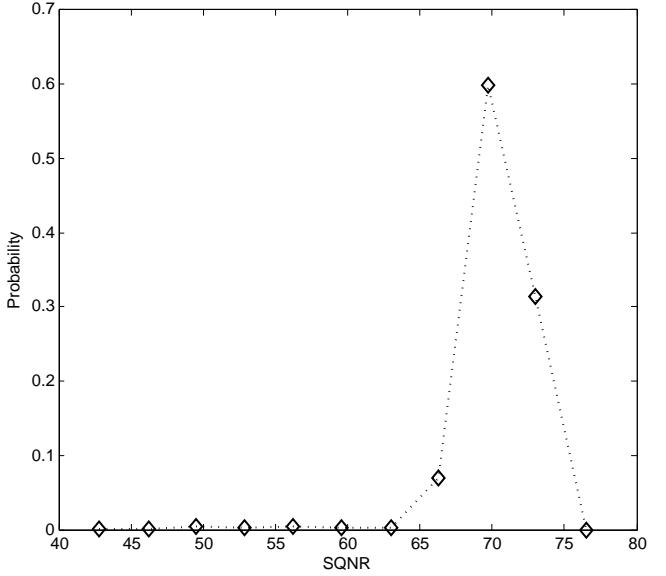


Figure 15. SQNR probability density function of SYSDEC architecture for 300 Monte Carlo trials.

the following scenario (similar to used in [6]) was considered. The hardware transmitter was configured in DDST mode, in order to send data blocks of $N = 512$ symbols obtained from a 4-QAM constellation. The channel is randomly generated at each Monte Carlo trial and it is assumed to be Rayleigh with length $L = 8$. The power of training sequence is set to $0.2\sigma_s^2$ with a period P equal to L .

Fig. 14 shows the MSE of channel estimated, which is averaged over 300 Monte Carlo simulation for each value of SNR. Note that the MSE of the hardware estimator is too close to the theoretical line [4] and almost indistinguishable with respect to the golden model. On the other hand, Fig. 15 presents the probability density function (PDF) of the SYSDEC hardware, obtained for the same Monte Carlo trials. Analyzing such PDF, it can be commented that the fixed-point performance in average is about 68 dB in terms of SQNR.

VI. CONCLUSIONS

In this paper, a digital architectures for transmitter and channel estimation stages of the ST/DDST communications systems have been presented. These architectures represents the first implementations under the full-hardware philosophy for a wireless systems based on ST/DDST. Both architectures presents high throughput and reduced FPGA resources consumption, achieving a good trade-off between performance and area utilization. The proposed transmitter architecture is configurable enough to generate two types of training using three constellation orders. In the SYSDEC hardware, it is possible to observe a great flexibility and re-usability because the same systolic array is used for two different tasks (operations): cyclic mean and channel

estimation. Also, the SYSDCE design can be easily modified (by means of partitioning strategy) for processing channels of different lengths. The validity and performance of these approaches have been verified by Monte Carlo simulations, where an SQNR of 82 db and 68 dB in average are achieved for the transmitter and the SYSDCE, respectively. At the same time both architectures presents a insignificant differences in the performance results when they are compared with their respective floating point golden models. The provided results show that ST/DDST concepts can be effectively utilized in a current and future wireless communications standards.

ACKNOWLEDGMENT

This work was supported by PROMEP ITSON-92, CONACYT-181962 and Mixbaal 158899 research grants.

REFERENCES

- [1] A. Goljahani, N. Benvenuto, S. Tomasin, and L. Vangelista, "Superimposed sequence versus pilot aided channel estimations for next generation dvb-t systems," *Broadcasting, IEEE Transactions on*, vol. 55, no. 1, pp. 140–144, march 2009.
- [2] B. Farhang-Boroujeny, "Pilot-based channel identification: proposal for semi-blind identification of communication channels," *Electronics Letters*, vol. 31, no. 13, pp. 1044–1046, 22 June 1995.
- [3] S. Haykin and K.J. Ray Liu, *Handbook on Array Processing and Sensor Networks*, Wiley, 2009.
- [4] E. Alameda-Hernandez, D.C. McLernon, A.G. Orozco-Lugo, M.M. Lara, and M. Ghogho, "Frame/training sequence synchronization and dc-offset removal for (data-dependent) superimposed training based channel estimation," *IEEE Trans. Signal Process.*, vol. 55, no. 6, pp. 2557–2569, June 2007.
- [5] M. Ghogho and Ananthram Swami, "Improved channel estimation using superimposed training," in *Proc. IEEE 5th Workshop on Signal Processing Advances in Wireless Communications*, 11–14 July 2004, pp. 110–114.
- [6] M. Ghogho, D. McLernon, E. Alameda-Hernandez, and A. Swami, "Channel estimation and symbol detection for block transmission using data-dependent superimposed training," *IEEE Signal Process. Lett.*, vol. 12, no. 3, pp. 226–229, 2005.
- [7] O. Longoria-Gandara, R. Parra-Michel, M. Bazdresch, and A. G. Orozco-Lugo, "Iterative mean removal superimposed training for siso and mimo channel estimation," *International Journal of Digital Multimedia Broadcasting*, p. 9, 2008.
- [8] R. Carrasco-Alvarez, R. Parra-Michel, A. G. Orozco-Lugo, and J. K. Tugnait, "Enhanced channel estimation using superimposed training based on universal basis expansion," *IEEE Trans. Signal Process.*, vol. 57, no. 3, pp. 1217–1222, 2009.
- [9] V. Najera-Bello, "Design and construction of a digital communications system based on implicit training," M.S. thesis, CINVESTAV-IPN, 2008.
- [10] Fernando Martín del Campo, René Cumplido, Roberto Perez-Andrade, and A. G. Orozco-Lugo, "A system on a programmable chip architecture for data-dependent superimposed training channel estimation," *International Journal of Reconfigurable Computing*, vol. 2009, 2009.
- [11] E. Romero-Aguirre, R. Parra-Michel, R. Carrasco-Alvarez, and A.G. Orozco-Lugo, "Architecture based on array processors for data-dependent superimposed training channel estimation," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, 30 2011-dec. 2 2011, pp. 303 –308.
- [12] A.G. Orozco-Lugo, M.M. Lara, and D.C. McLernon, "Channel estimation using implicit training," *IEEE Trans. Signal Process.*, vol. 52, no. 1, pp. 240–254, Jan 2004.
- [13] N. Petkov, *Systolic Parallel Processing*, Elsevier Science Inc., New York, NY, USA, 1992.
- [14] S. Kung, *VLSI Array processors*, Prentice Hall, 1985.

Full-Hardware Architectures for Data-Dependent Superimposed Training Channel Estimation

Eduardo Romero-Aguirre ·
Roberto Carrasco-Alvarez · Ramón Parra-Michel ·
Aldo G. Orozco-Lugo · Antonio F. Mondragón-Torres

Received: 13 April 2012 / Revised: 7 August 2012 / Accepted: 24 September 2012
© Springer Science+Business Media New York 2012

Abstract Channel estimation based on superimposed training (ST) has been an active research topic around the world in recent years, because it offers similar performance when compared to methods based on pilot assisted transmissions (PAT), with the advantage of a better bandwidth utilization. However, physical

This paper is an extension of the paper presented in IEEE Workshop on Signal Processing Systems (SiPS) 2011. The added sections are: (1) An update of the system model (Section 2); (2) Description for the algorithms for DC-offset estimation and synchronization correction (Section 2.2). (3) VLSI implementations of two of the proposed architectures (Section 5.1); (4) A new architecture for DDST channel estimation under more realistic conditions (Section 4); (5) A fixed-point analysis of the new architecture proposed (Section 5.2)

E. Romero-Aguirre (✉) · R. Parra-Michel
Department of Electrical Engineering, CIVESTAV-GDL,
Zapopan, Jalisco, México
e-mail: eromero@gdl.cinvestav.mx

R. Parra-Michel
e-mail: rparra@gdl.cinvestav.mx

R. Carrasco-Alvarez
Department of Electronic Engineering, UDG-CUCEI,
Guadalajara, Jalisco, México
e-mail: roberto.carrasco@red.cucei.udg.mx

A. G. Orozco-Lugo
Department of Electrical Engineering CINVESTAV-DF,
México, D.F., México
e-mail: aorozco@cinvestav.mx

A. F. Mondragón-Torres
Department of Electrical, Computer and
Telecommunications Engineering Technology, RIT,
Rochester, NY, USA
e-mail: afmjee@rit.edu

implementations of such estimators are still under research, and only few approaches have been reported to date. This is due to the computational burden and complexity involved in the algorithms in conjunction with their relative novelty. In order to determine the suitability of the ST-based channel estimation for commercial applications, the performance and complexity analysis of the ST approaches is mandatory. This work proposes two full-hardware channel estimator architectures for a data-dependent superimposed training (DDST) receiver with perfect synchronization and nonexistent DC-offset. These architectures were described using Verilog HDL and targeted in Xilinx Virtex-5 XC5VLX110T FPGA. The synthesis results of such estimators showed a consumption of 3 % and 1 % of total slices available in the FPGA and frequencies operation over 160 MHz. They have also been implemented on a generic 90 nm CMOS process achieving clock frequencies of 187 MHz and 247 MHz while consuming 3.7 mW and 2.74 mW, respectively. In addition, for the first time, a novel architecture that includes channel estimation, training/block synchronization and DC-offset estimation is also proposed. Its fixed-point analysis has been carried out, allowing the design to produce practically equal performance to those achieved with the floating-point models. Finally, the high throughputs and reduced hardware consumptions of the implemented channel estimators, leads to the conclusion that ST/DDST can be utilized in practical communications systems.

Keywords Channel estimation · Data-dependent superimposed training · FPGA architectures · Implicit training · Synchronization · VLSI

1 Introduction

Nowadays, the users of modern communication systems require more efficient systems in order to transmit and receive information at higher rates. To meet this need, it is necessary to propose new algorithms that make an efficient use of available bandwidth resources. Due to the fact that mobility is considered in all modern communications standards, it is required to waste part of the data bandwidth in sending pilots to allow the receiver to estimate the channel. This process can be classified into two major branches:

- Pilot assisted transmissions (PAT).
- Implicit training (IT)

PAT is the technique considered in current communication standards, such as LTE, WiMax, DVB-H, etc. Under this approach, the data and pilot sequences are transmitted in orthogonal spaces, allowing a simple separation of the received pilots for channel estimation and usable data payload, at the expense of large bandwidth waste. On the other hand, IT is a recently-proposed methodology, where the paradigm consists of transmitting the training sequence hidden in the data signal [9, Ch. 6]; i.e., data and training are transmitted in non-orthogonal spaces. This approach presents advantages over PAT techniques, because it is not necessary to waste valuable bandwidth, therefore, it has been recognized as a true alternative for future communication standards [8]. However, the channel estimation process becomes more complicated, requiring analysis of complexity before being able to assess its suitability for practical applications.

The simplest way to carry out IT consists of adding the training signal to the data signal. This approach is known as superimposed training (ST), which was first proposed in [5] and enhanced in [4, 15, 18, 19]. A refinement of ST, known as data dependent superimposed training (DDST) was presented in [2, 6, 7, 12]. This refinement makes it possible to cancel the interference produced by the transmitted data during the channel estimation process. This is achieved by adding to the training sequence utilized in ST, a second training sequence that depends on the transmitted data.

An application of DDST is reported in [8] for DVB-T communication systems. There is demonstrated that the use of DDST provides a higher bit rate (increment of 4 to 10 %) when compares with conventional training schemes. This enhancement of the bit rate motivates the study of new architectures capable to deal with such new training schemes paradigm.

Although the topic of ST is now mature enough from the mathematical modeling point of view, it is

remarkable that in the state-of-the-art there are few works devoted to identifying its suitability for practical applications. An implementation of a DDST system is reported in [14], where the algorithm is programmed in a TMS320C6713 digital signal processor (DSP) with a 300 MHz external clock using floating-point arithmetic. A symbol period of 2 ms with a frequency carrier of 40 KHz was used for the data transmission. A system-on-chip (SoC) paradigm was used in [3] for developing a DDST receiver, which consists of an embedded processor (NIOS II) with hardware accelerators (coprocessors) implemented on a Stratix II FPGA board with an oscillator of 100 MHz. Its performance is expressed in term of the number of times that the coprocessors are faster than its software implementation only, no results about functional efficiency of the implemented channel estimator are given. At [17], we have presented two full-hardware architectures for DDST channel estimation. This work reports both, synthesis and performance results of the proposed channel estimators using fixed-point arithmetic. A systolic DDST channel estimation was reported in [16], where the same systolic array is used for two different tasks (operations): cyclic mean and channel impulse response (CIR) estimation. In this paper, we present an extended version of [17], where it has been included a novel full-hardware architecture for DDST channel estimation under the assumption of lack of synchronization and unknown DC-offset. The mapping of the algorithm to the aforementioned architecture is described in detail and its fixed-point performance is presented.

The rest of paper is organized as follows. Section 2 presents the system model being considered and the channel estimation algorithm. Section 3 describes in detail two full-hardware architectures for the DDST channel estimator with perfect synchronization and nonexistent DC-offset. Section 4 proposes a new full-hardware architecture for the DDST channel estimator under lack of synchronization and unknown DC-offset. In Section 5, the performance evaluation of the proposed architectures is carried out. Finally, the concluding remarks are given in Section 6.

Notation Lowercase (uppercase) bold letters denote column vectors (matrices). Operators $(\mathbf{A})^H$, $(\mathbf{A})^T$ and $(\mathbf{A})^{-1}$, denote the Hermitian, transpose and inverse operations of matrix \mathbf{A} . $\mathbf{1}_n$ represents a column vector of length n with all its elements equal to one, similarly, $\mathbf{0}_n$ represents an all-zeros column vector of length n . \mathbf{I}_n is the identity matrix of size $n \times n$. $[\mathbf{a}]_k$ denotes the k -th element of vector \mathbf{a} , $[\mathbf{a}]_{m:n}$ denotes a vector conformed with the elements of \mathbf{a} as follows: $[[\mathbf{a}]_m, [\mathbf{a}]_{m+1}, \dots, [\mathbf{a}]_n]^T$. \otimes represents Kronecker

product. $\text{circshift}_n(\mathbf{a})$ circular shift down n positions vector \mathbf{a} , while $\text{circshift}_n(\mathbf{A})$ circular shift down n positions the rows of \mathbf{A} . $\|\mathbf{a}\|$ is the Euclidean norm of vector \mathbf{a} . Finally, $E(\cdot)$ represents the expectation operator.

2 System Model

In this Section, the DDST algorithm mentioned previously is mathematically introduced. If we consider a complex base-band representation of a single carrier communication system based on DDST, as presented in Fig. 1, the transmitted signal $s(k)$ is formed by the superposition of the data sequence $b(k)$, the training sequence $c(k)$ and the data-dependent training sequence $e(k)$. It is assumed that $c(k)$ is a periodic sequence with period P , mean value \bar{c} and power equal to σ_c^2 [15]. The sequence $e(k)$ is constructed as mentioned in [7]. Also, it is considered that $E(b(k)) = 0$, $E(|b(k)|^2) = \sigma_b^2$ and $E(|e(k)|^2) = \sigma_e^2$. The index k helps to enumerate the samples of the aforementioned signals, which are transmitted at a rate equal to $1/T$. The transmitted signal is propagated through the time invariant communication channel $h(k)$, which is the abstraction of the propagation medium and the system filters (pulse shaping and matched filters). This channel can be modeled as a causal finite impulse response (FIR) filter, which is assumed to have at most L coefficients. The received signal $x(k)$ is the sum of the signal distorted by the channel, the unknown DC-offset m and a white Gaussian noise $n(k)$ that possesses variance equal to σ_n^2 . Finally, it is considered that the transmission is performed in blocks of N symbols length, preceded by a cyclic prefix of length $CP \geq L$, as explained in [1]. For the implementation of the synchronization algorithm and channel estimation algorithm it is necessary to fix $P \geq 2L + 1$. For simplicity, we have constrained N to be a multiple of P and P to be a power of two.

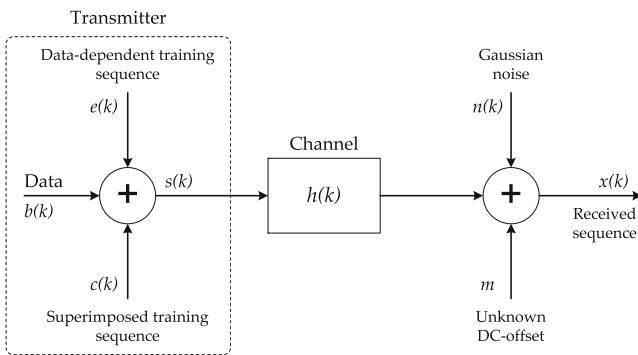


Figure 1 Digital communication system model considered.

The received signal can be expressed as:

$$x(k) = \sum_{l=0}^{L-1} h(l)s(k-l) + n(k) + m, \quad (1)$$

where $0 \leq k \leq M-1$, M is the number of samples received.

2.1 Channel Estimation Using DDST

In order to explain the DDST algorithm, both perfect block and training sequence synchronization are assumed; in other words, the first received sample is the signal with the cyclic prefix removed. It is possible to observe that due to the periodicity of $c(k)$, $s(k)$ will have a periodic signal embedded with a period equal to P . Taking advantage of this characteristic, the DDST-based channel estimation algorithms use an estimate of the cyclic mean of the received signal, which is given by:

$$\mathbf{y} = \mathbf{J}\mathbf{x}, \quad (2)$$

where \mathbf{y} is a column vector of length P whose elements are the estimated cyclic mean of \mathbf{x} , $[\mathbf{x}]_k = x(k)$ for $0 \leq k \leq N-1$ and \mathbf{J} is given by:

$$\mathbf{J} = \frac{1}{N_P} (\mathbf{1}_{N_P}^T \otimes \mathbf{I}_P), \quad (3)$$

where $N_P = N/P$.

According to [2], the estimation of the CIR \mathbf{h} is given by:

$$\hat{\mathbf{h}} = \mathbf{g} - \hat{\mathbf{m}}, \quad (4)$$

where $\hat{\mathbf{h}}$ is an estimate of the vector $\hat{\mathbf{h}} = [\mathbf{h}^T, \mathbf{0}_{P-L}^T]^T$, $\hat{\mathbf{m}}$ is an estimation of the vector $\tilde{\mathbf{m}} = \tilde{m}\mathbf{1}_P$ for $\tilde{m} = \frac{m}{\bar{c}}$, $\mathbf{g} = \mathbf{C}^{-1}\mathbf{y}$ and \mathbf{C} is a circulant matrix of size $P \times P$ formed by vector $[c(0), c(1), \dots, c(P-1)]^T$.

In the case of a nonexistent DC-offset, the condition $P \geq 2L + 1$ could be relaxed to $P = L$. Thus, Eq. 4 becomes:

$$\hat{\mathbf{h}} = \mathbf{C}^{-1}\mathbf{y}, \quad (5)$$

where now $\hat{\mathbf{h}} = \mathbf{h}$.

2.2 DC-Offset and Training Sequence/Block Synchronization

In order to obtain an estimate of the channel using the algorithm presented above, it is necessary first to compensate for the DC-offset and the lack of synchronization. The latter can be viewed as the conjunction of two different problems: training sequence synchronization (TSS) and block synchronization (BS). Thus, the total number of samples for reaching the start of the next

transmitted block can be expressed as: $\tau_s = \tau + P\tau_p$, where $0 \leq \tau \leq P - 1$ indicates the number of necessary samples, such that the received signal will match the beginning of the period of the training sequence $c(k)$. Meanwhile, $0 \leq \tau_p \leq N_P - 1$ indicates the number of periods needed to reach the beginning of the next transmitted block. If a lack of TSS is presented, the cyclic mean estimation presents a cyclic permutation [1]. On the other hand, the lack of BS produces a detriment of the channel estimation performance due to the fact that the data-dependent training sequence is calculated over a data block. Therefore, in order to obtain a good performance it is necessary to be synchronized with the beginning of the transmitted block to ensure the cancellation of the data interference during the estimation process.

The first step for obtaining the synchronization is to store $2(N + P)$ samples of the received signal in the vector $[\mathbf{x}]_k = x(k)$ for $0 \leq k \leq 2(N + P) - 1$. It is possible to observe that at least one transmitted block is contained in this arrangement. Once this is done, it is possible to find the beginning of the period of the training sequence. To obtain TSS it is necessary to remember that a lack of synchronization produces a cyclic permutation of \mathbf{y} , this permutation is in accordance with the number of samples τ that the received signal is displaced from the beginning of the period of $c(k)$; on the other hand, when perfect TSS is ensured, the last $P - L$ elements of \mathbf{g} are on average equal to \tilde{m} . Thus, the TSS algorithm tries to find such a cyclic permutation matrix \mathbf{P}_τ that minimizes the difference between the last $P - L$ elements of \mathbf{g} , i.e. their magnitud values are almost equal. Nevertheless, it is possible to show that a cyclic permutation of \mathbf{y} leads to a cyclic permutation of \mathbf{g} that avoids the search for \mathbf{P}_τ . Thus, TSS is found via the minimization of the following functional:

$$\hat{\tau} = \arg \min_{\tau} \{f(\tilde{\mathbf{h}}_\tau)\}, \quad (6)$$

where $f(\mathbf{v}) = \|\mathbf{v} - \bar{\mathbf{v}}\|$ for $\bar{\mathbf{v}} = [\bar{v}, \dots, \bar{v}]^T$ and \bar{v} the mean value of the elements of \mathbf{v} . $0 \leq \tau \leq P - 1$ is the training displacement, $\tilde{\mathbf{h}}_\tau = [circshift_\tau(\mathbf{C}^{-1})\mathbf{y}]_{L:P-1}$ is the vector conformed by the last $P - L$ elements of $circshift_\tau(\mathbf{C}^{-1})\mathbf{y}$ where \mathbf{g} is calculated with the first N samples of the stored signal \mathbf{x} .

Once the training displacement $\hat{\tau}$ is found, the next step consists of finding the beginning of the transmitted block. The latter means to find the parameter τ_p . For achieving this goal, according to [2], it is possible to apply the same functional used during the estimation of τ with some modifications, thus, τ_p is found by means of:

$$\hat{\tau}_p = \arg \min_{\tau_p} \{f([\mathbf{g}_{\tau_p}]_{L:P-1})\}, \quad (7)$$

where $0 \leq \tau_p \leq N_P - 1$, $\mathbf{g}_{\tau_p} = \mathbf{C}^{-1}\mathbf{y}_{\tau_p}$, $\mathbf{y}_{\tau_p} = \mathbf{J}\mathbf{x}_{\tau_p}$ and $[\mathbf{x}_{\tau_p}]_j = x(j + \hat{\tau} + P\tau_p)$ for $0 \leq j \leq N - 1$.

With the estimation of parameter τ_s , it is possible to identify the beginning of a transmitted data block and extract such block from the stored data \mathbf{x} . So, it is now viable to estimate with the extracted data block the DC-offset, and then, to estimate the CIR applying Eq. 4; where \mathbf{g} is calculated with $x(\hat{\tau} + P\hat{\tau}_p + k)$ for $0 \leq k \leq N - 1$ and the estimate of the DC-offset is given by

$$\hat{\mathbf{m}} = \hat{m}\mathbf{1}_P, \quad \text{for } \hat{m} = \frac{1}{P - L} \sum_{i=L}^{P-1} [\mathbf{g}]_i. \quad (8)$$

3 Channel Estimator Architectures for a DDST Receiver with Perfect Synchronization and Nonexistent DC-Offset

This Section introduces two architectures for the DDST-based channel estimation process. Both architectures are based on three main functional units: a cyclic mean estimator, a systolic array matrix-vector multiplier and a memory buffer. In the first architecture, referred to in this paper as “channel estimation with data feed from memory” (CEDAM), the hardware performs the algorithm using the data previously stored in the memory. In the second one, which we have called “Channel estimation with on-the-fly data feed” (CEOOF), the channel estimation is carried out at the same time as the input stream is being received and stored in the buffer.

It is worth mentioning that in both architectures, P and N must be a power of two. Also, all the signals are complex valued.

3.1 CEDAM Architecture

The CEDAM architecture is presented in Fig. 2. It consists of a main buffer (MB), a parallel cyclic mean estimator (PCME) and a systolic matrix-vector multiplier (SMVM). The process for calculating the channel estimated through each of the components of the architecture is described below.

As soon as the *start* signal is asserted, the PCME unit, during N_P cycles, processes P parallel data of the sequence $x(k)$ simultaneously, taking advantage of the fact that this sequence has already been stored in the MB. Once the cyclic mean \mathbf{y} is obtained from the received sequence $x(k)$, the SMVM unit will perform the product expressed in Eq. 5. After $P + 1$ cycles, the *done* flag is asserted and the channel estimated $\hat{\mathbf{h}}$ values are sent to the *OUT* bus one by one. The functional

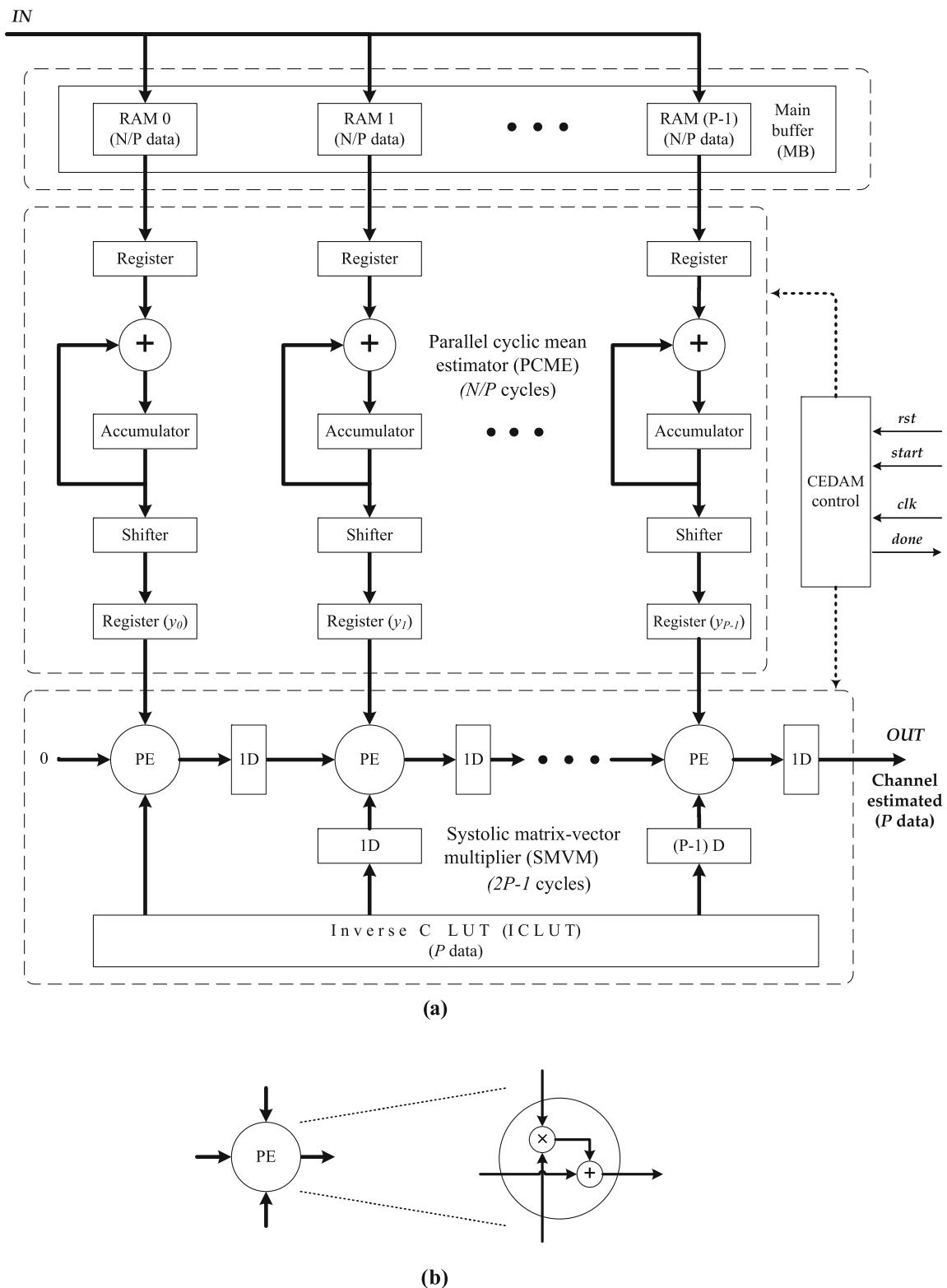
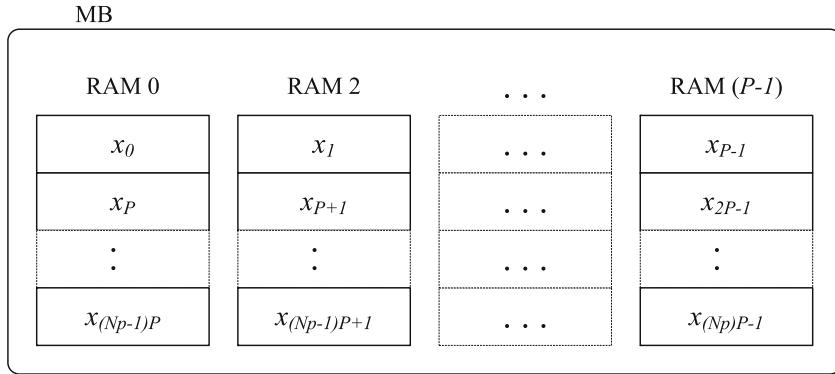


Figure 2 CEDAM estimator; **a** simplified architecture, **b** processor element.

Figure 3 Data organization in the main buffer (MB).



details of each component of the CEDAM architecture are presented in the following subsections.

3.1.1 Main Buffer (MB)

It is made up of an array of P memories, each with a depth of N_p , organized as a memory bank as shown in Fig. 3. At the DDST receiver, the data sequence $x(k)$ must be stored in MB as it is being received. This backup allows that incoming data can be read by the modules added to the new architecture (as proposed in Section 4) for dealing with lack of synchronization and unknown DC-offset.

In accordance with the memory bank features, a “hard-wired” addressing approach is designed for the MB. As depicted in Fig. 4, the $\log_2(N)$ bits corresponding to the MB address bus are divided into two parts. The first P least significant bits (LSB) are used to select a particular memory within the bank and the remaining bits are used to individually address each of the locations in the selected memory. Therefore, the N stored data can be viewed as a $N_p \times P$ matrix, where each individual memory in the bank stores a one column of the matrix, as depicted in Fig. 3. Moreover, the MB data can be read in the same order they were stored or in the same order as they were received.

It is worth noting that the MB introduces a latency of $N + P$ cycles in the CEDAM operation. For this reason, a “ping-pong” strategy using two MBs is proposed (Fig. 5), so that while the CEDAM is processing the data stored in one MB, another MB can be filled

simultaneously with a different input sequence. This CEDAM architecture variant is called “Double-MB CEDAM”.

3.1.2 Parallel Cyclic Mean Estimator (PCME)

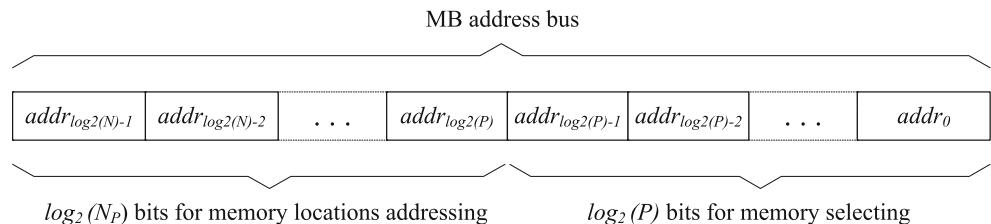
The PCME design is based on a massive-parallel architecture, because it consists of P modules computing P arithmetic means in parallel. Each mean module is made up of one input register, one complex adder, one accumulator register, one shifter and one output register, as shown in Fig. 2.

Once the N data of the $x(k)$ sequence have been rearranged and stored as a matrix in the MB, the PCME hardware reads N_p data from each memory to obtain the sum. From the point of view of the algorithm, this is equivalent to adding the elements in each of the matrix’s columns. Next, the results of each sum are divided by N_p in the shifter block. Finally, the P results are fed in parallel to the SMVM unit.

3.1.3 Systolic Matrix-Vector Multiplier (SMVM)

The fundamental operation to be performed by CEDAM is the matrix operation represented by Eq. 5. From the point of view of hardware complexity, this part of the design is the most critical in the entire architecture. The obvious strategy to accelerate its performance consists of executing as many operations in parallel as possible, but this strategy consumes a high amount of FPGA resources. Therefore, we decided to

Figure 4 Hard-wired addressing for MB.



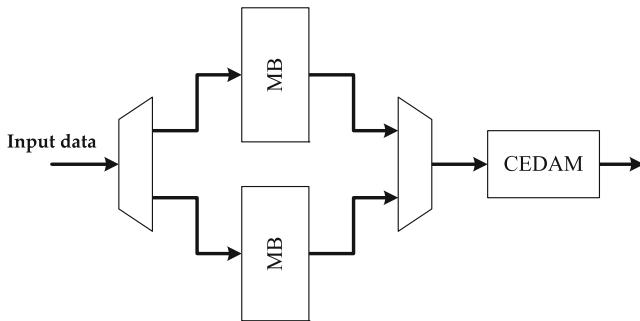


Figure 5 Block diagram of the CEDAM estimator with two MB (Double-MB CEDAM).

design a systolic array processors, similar to the one presented in [10, Ch. 3], in order to obtain good performance and avoid high resource allocation. Another advantage is that the SMVM can be used in conjunction with the partition technique [13, Ch. 5] for calculating matrix-vector multiplications larger than the size of systolic array.

The processor element (PE), shown in Fig. 2b, is the atomic digital signal processing module in the SMVM. It consists of one complex multiplier and one complex adder, that processes three signals: the data from the PCME, the inverse \mathbf{C} look-up table (LUT) and the data produced by the previous adjacent PE.

A systolic design of Eq. 5 was considered. The sizes of matrix \mathbf{C}^{-1} and vector \mathbf{y} are $P \times P$ and $P \times 1$, respectively, so the number of PEs needed is P . The projection vector $\mathbf{d} = [1 \ 0]^T$ (see details in [10]) was used with a vector schedule $\mathbf{s} = [1 \ 1]^T$. The pipelining period for this design is equal to 1 and the computing

time for the matrix-vector multiplication is $2P - 1$ clock periods; however, the first resulting datum is available in $P + 1$ clock cycles.

3.2 Inverse C Look-up Table (ICLUT)

The values of the circulant matrix \mathbf{C}^{-1} are constants that can be precomputed once off-line and stored in a LUT ROM. Only the first column's (row's) data are necessary because the remaining columns (rows) are shifted versions of the first one. Consequently, the ROM location's number required for the LUT is just P . If traditional design is used, then the LUT will be designed with a multi-port ROM of P locations, but it will be synthesized as an array of P single port ROMs. Therefore, the number of memory locations is increased to P^2 . A clever solution was designed with an array of P registers (Reg_0 – Reg_{P-1}) operating as a circular buffer. This is called “inverse C look-up table” (ICLUT) and it saves $P(P - 1)$ memory locations. The first row values of \mathbf{C}^{-1} are stored in the registers. Next, one rotation is applied in each clock transition to change the register's outputs, as indicated in Fig. 6.

3.3 CEOF Architecture

The architecture for this estimator is presented in Fig. 7, where the PCME module is substituted by a serial cyclic mean estimator (SCME) due to the fact that a different paradigm is used. The SCME module in Fig. 7 no longer needs to wait until $N + P$ data have received and stored in MB.

Figure 6 Simplified architecture of “inverse C look-up table” and its corresponding outputs values.

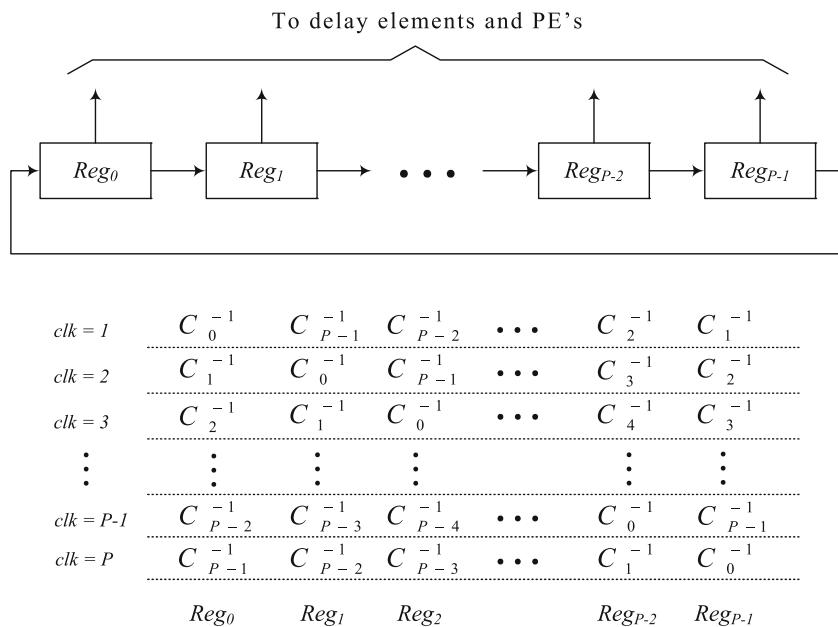
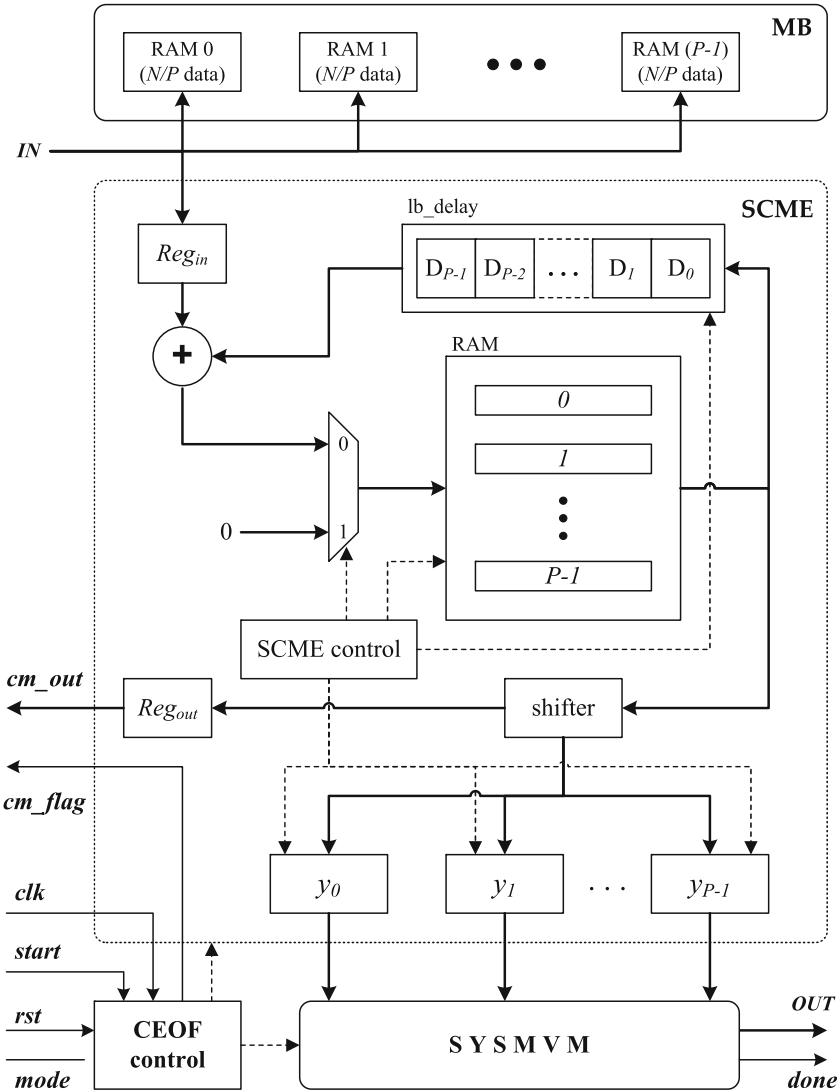


Figure 7 CEOF hardware architecture.



During $N + P$ cycles, the cyclic mean is computed “on-the-fly” as each datum is being received. At the same time, the input sequence is stored and available for other modules. The SCME should be able to identify the first P values corresponding to the cyclic prefix and these values are not taken into account when computing the cyclic mean. In the SCME, the RAM locations are used for storing the P sums and once the computation has finished, all of them must be set to zero together with the registers of the lb_delay module. The MB is not used and the SMVM module remains unchanged in the architecture. Additionally, the CEOF offers two useful functional features:

1. It can operate in continuous flow when the additional buffer ($2P$ depth) is connected to its IN bus.
2. It can be configured to compute either the channel estimation values or the cyclic mean only, depend-

ing upon the $mode$ control signal value. In the latter case, the results are sent to the cm_out bus in serial form and the cm_flag is asserted.

4 Proposed Architecture for DDST Channel Estimator with Synchronization Correction and DC-Offset Estimation

This Section presents the way the CEOF estimator can be used for designing a more sophisticated full-hardware channel estimator that also considers TSS, BS and DC-offset estimation. This new proposed architecture hereafter called “Extended CEOF” (ECEOF) is based on the algorithm presented in Section 2.2 and its design is described in details in following paragraphs.

4.1 Algorithm Adaptations

The main advantage of the algorithm used in ECEOFS hardware is that DC-offset estimation and synchronization are performed jointly. In general, there are four tasks in the algorithm: TSS, BS, DC-offset estimation and true-channel estimation. All of them are highly iterative and sequential procedures. This means that no new process can begin if the current one has not been finished, creating bottlenecks due to the high dependence among processes. In consequence, adaptations should be made to the original equations of the algorithm in order to deal with its “mapping-to-hardware” process efficiently from the point of view of throughput and area consumption.

4.1.1 TSS Equation Adaptations

As stated in Section 2.2, the first CIR estimate is just a circular shifted version of the true-channel estimation. Therefore, one of the possible permutations corresponds to the channel estimation under TSS, so the problem is reduced to obtaining the correct permutation. In Eq. 6, the permutation is defined originally in the matrix \mathbf{C}^{-1} before its multiplication by the cyclic mean, which requires performing P matrix–vector products. However, because the inverse of \mathbf{C} is a circulant matrix, $P - 1$ matrix–vector multiplications were spared when the following property was applied:

$$\text{circshift}_\tau(\mathbf{C}^{-1})\mathbf{y} = \text{circshift}_\tau(\mathbf{C}^{-1}\mathbf{y}). \quad (9)$$

On the other hand, we know that the norm of a vector $\mathbf{v} \in \mathbb{C}$ of length N is given by

$$\|\mathbf{v}\| = \sqrt{\sum_{k=0}^{N-1} (Re([\mathbf{v}]_k)^2 + Im([\mathbf{v}]_k)^2)}. \quad (10)$$

In contrast, the norm expressed in Eq. 6 comes from $(\mathbf{h}_\tau - \bar{\mathbf{h}}_\tau)$ subtraction result. This involves two readings of the same vector results because it is necessary to wait until $\bar{\mathbf{h}}$ is calculated. Therefore, a modified version of the norm factorization, proposed in [3], was applied in order to calculate the norm in one step.

$$\|\mathbf{h} - \bar{\mathbf{h}}\| = \sqrt{A - 2B + C}, \quad (11)$$

where

$$A = \sum_{k=L}^{P-1} [Re([\mathbf{h}]_k)^2 + Im([\mathbf{h}]_k)^2] \quad (12a)$$

$$B = Re(\bar{h}) \sum_{k=L}^{P-1} Re([\mathbf{h}]_k) + Im(\bar{h}) \sum_{k=L}^{P-1} Im([\mathbf{h}]_k) \quad (12b)$$

$$C = (P - L)(Re(\bar{h})^2 + Im(\bar{h})^2). \quad (12c)$$

Furthermore, due to $\|\mathbf{h} - \bar{\mathbf{h}}\| \geq 0$ and taking advantage of the fact that the square root function of a positive number is strictly monotonically increasing, then $\sqrt{a} < \sqrt{b}$ can be substituted by $a < b$. Hence, in each iterations the square root operations can be omitted without affecting the comparison results.

4.1.2 BS Equation Adaptations

The BS correction algorithm involves the same operations of the TSS with the following exceptions:

- The number of iterations is N_P instead of P .
- Contrary to the TSS procedure, in BS the cyclic mean vector \mathbf{y}_{τ_p} should be calculated on each iteration and it is no longer possible to apply Eq. 9. So, it is absolutely necessary to perform N_P matrix–vector multiplications for finding the cyclic mean vector that minimizes Eq. 7.

It should be emphasized, that Eq. 11 is used again for calculating the norm in Eq. 7.

4.2 ECEOFS Architecture Description

The Fig. 8 shows the simplified architecture for this estimator. It is made up of several modules that can be grouped into four functional domains:

1. Data reception.
2. Cyclic mean and channel estimation.
3. Synchronization delay estimation.
4. True-channel estimation.

The dataflow through these domains during the execution of the algorithm is subsequently explained.

As soon as the *start_eceof* signal is asserted, the received sequence provided by input bus *IN_HEST* is stored in MB. Simultaneously, the first N samples are fed to the CEOF module—through the input 0 of MUX1—for calculating the first channel estimation. The P estimated coefficients are stored in the temporal buffer (TB). A memory pointer *mptr_sync_addr* is initialized with $P - L - 1$ by the AGU_SYNC module. In order to find τ , an iterative process takes place between domains 2 and 3.

- (a) The last $P - L$ TB locations are read by domain 2 for TSS functional evaluation.

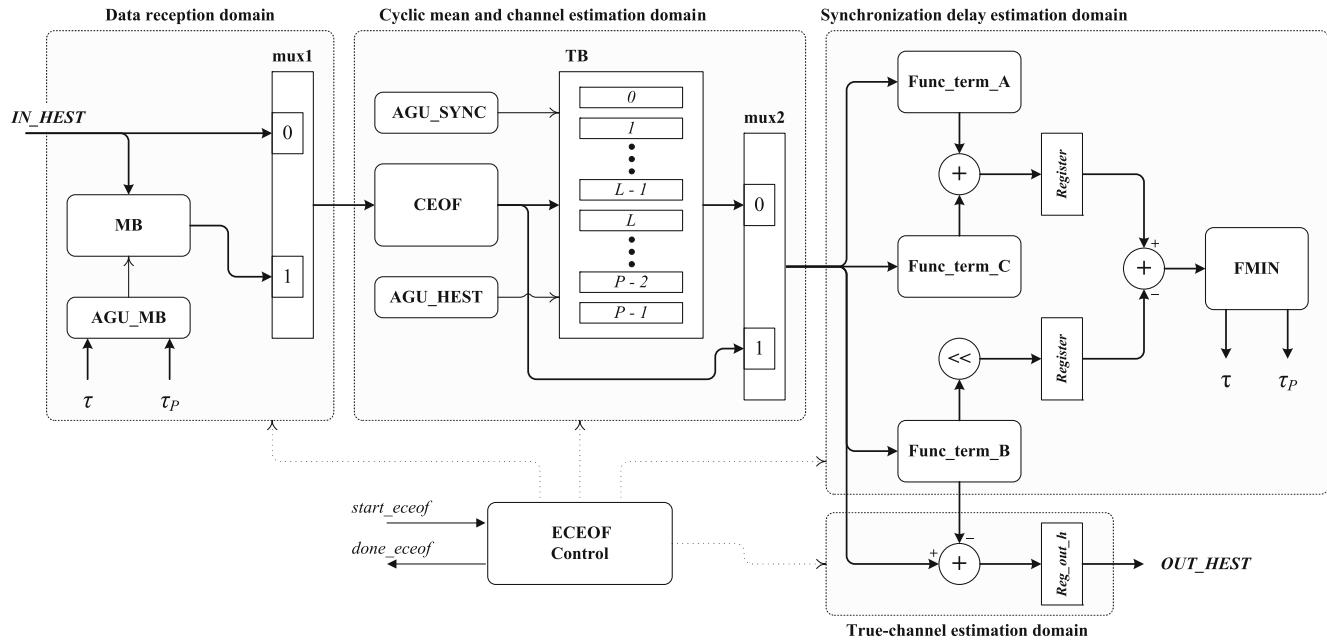


Figure 8 Proposed architecture of the ECEOF.

- (b) In the FMIN module, the previous functional result is compared with the current one. Only the smaller is chosen and stored.
- (c) In the AGU_SYNC module, the *mptr_sync_addr* pointer is incremented by one.

These steps are done P times. The number of the iteration which the smallest value of Eq. 6 was calculated corresponds to τ . Next, τ is used in the AGU_MB module as addressing offset for purposes of TSS correction. In the AGU_MB, a memory pointer *mfp_mb_addr* is

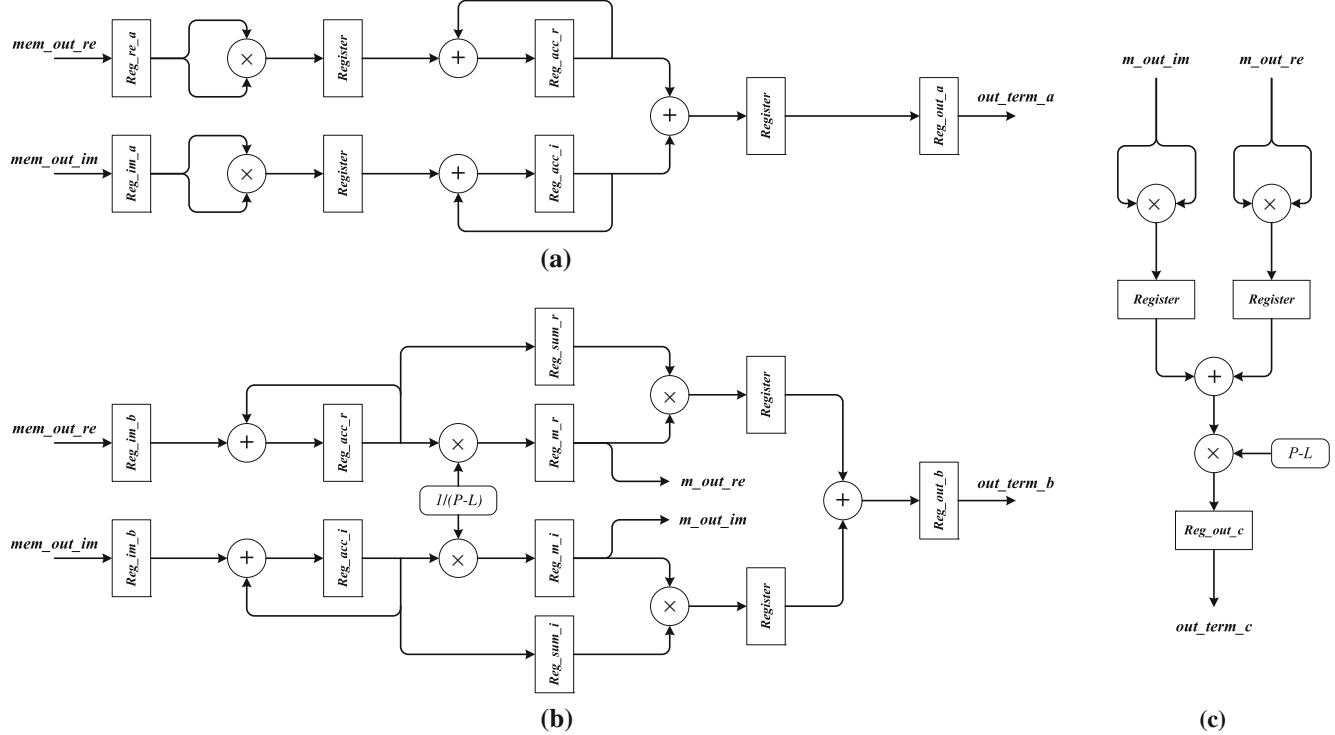


Figure 9 Architecture of the square norm modules; **a** func_term_A, **b** func_term_b, **c** func_term_C.

initialized with $\tau + P$ value. A new iterative process begins with the domains 1–3 for block synchronization.

- (a) A block of N data is read from MB ($[\mathbf{x}]_{mrp_mb_addr-1}$ to $[\mathbf{x}]_{mrp_mb_addr+N-1}$) by the CEOF for \mathbf{y}_{τ_p} and \mathbf{g}_{τ_p} estimated.
- (b) The last $(P - L)$ elements of \mathbf{g}_{τ_p} vector are bypassed to domain 3 for BS functional calculation.
- (c) The previous BS functional result and the current one are compared by the FMIN module. Only the smaller of them is stored.
- (d) The mrp_mb_addr pointer is incremented by P .

These steps are done N_p times. The number of the iteration which the smallest value of Eq. 7 was calculated corresponds to τ_p . An addressing offset to achieve perfect synchronization is obtained by applying $\tau_s = \tau + P\tau_p$; this value is used for mrp_mb_addr pointer updating. After this, a block of N data ($[\mathbf{x}]_{mrp_mb_addr+P-1}$ to $[\mathbf{x}]_{mrp_mb_addr+N+P-1}$) is read from MB by the CEOF estimator. The first L elements of \mathbf{g} are stored in TB and the last $P - L$ elements of the same vector are bypassed to func_term_B module for DC-offset estimating. Finally, the true-channel estimation (Eq. 4) is performed when the memory TB is read by domain 4 and the DC-offset estimated is subtracted from each value read. The CIR coefficients are sent to OUT_HEST bus and the *done_eceof* output is asserted.

4.3 Architecture of Square Norm Function

The main operation of the TSS and BS algorithm is the norm denoted in Eq. 11. However, for the reasons described in Section 4.1, Eqs. 12a–12c were used instead of the direct approach for architecture design of the square norm operation. The hardware modules func_term_A (Fig. 9a), func_term_B (Fig. 9b) and func_term_C (Fig. 9c) were designed for each one of these equations and they were interconnected to allow parallel execution. Thus, the calculation of Eq. 11 is effectively accelerated. Additionally, pipeline registers were inserted between math operations in order to reduce the path delay of each func_term modules.

5 Results

In this Section, the proposed architectures are evaluated. First, the CEDAM and CEOF implementations in FPGA are compared in terms of hardware utilization and throughput. Similar comparison is done for VLSI versions of the estimators in terms of area, complexity and power consumption. Next, their functional performance is analyzed and compared against the floating-

point simulation golden model programmed in Matlab. Finally, a fixed-points analysis of the ECEOF architecture is done and the functional simulation results are compared again with their floating-point counterpart.

5.1 Implementation and Simulation of the CEDAM and CEOF Estimators

5.1.1 FPGA Implementation

The CEDAM and CEOF architectures were implemented in RTL using Verilog hardware description language. These estimators process input sequences of length $N = 512$ with $CP = P$ and $P = 8$. They were synthesized and targeted to a Xilinx Virtex-5 XC5VLX-110T.

In the two implementations the following considerations apply:

1. Default settings and “no user constraints” were selected in Xilinx ISE v11.
2. No pre-designed components available from the core generator library were used.
3. All signals are represented in signed fixed-point using two’s complement.
4. Truncation was the rounding scheme used.
5. Both real and imaginary parts of the complex values are stored together, in the same memory location or register.

In order to find the specific wordlengths and word format suitable for the CEDAM and CEOF architectures, it was necessary to characterize the behavior of the received sequence due to random nature of the channel coefficients. For that reason, an exhaustive statically analysis of the received data was done using 312×10^6 data obtained via 10^5 Montecarlo simulations. Thus, the received sequenced was quantized using a wordlength of 16 bits with a word format of 4 integer bits and 12 bits for fractional part. Table 1 shows detailed wordlengths and word formats used in the implemented architectures. Additionally, as can be seen in such Table, the outputs of the shifters, multipliers and SMVM adders were rounded back to 18 bits of wordlength in order to avoid a great hardware consumption.

It is worth mentioning that it is difficult to compare the proposed channel estimators directly with the others available in the literature [3, 14], due to the lack of available information, the differences in technology, tool versions and testing conditions. A comparison with the implementation reported in [14] is not possible because this system is full-software. In the hybrid software-hardware FPGA implementation of the

Table 1 Wordlengths and word formats used in the CEDAM and CEOF architectures.

	Arithmetic element	CEDAM			CEO		
		Wordlength	Integer bits	Fractional bits	Wordlength	Integer bits	Fractional bits
PCME/SCME	Complex adder inputs	16	4	12	16	4	12
	Complex adder outputs	22	10	12	22	10	12
	Shifter inputs	22	10	12	22	10	14
	Shifter outputs	16	2	14	16	2	14
SMVM	Complex multiplier inputs	16	2/1	14/15	16	2/1	14/15
	Complex multiplier outputs	18	3	15	18	3	15
	Complex adder	18	3	15	18	3	15
ICLUT	ICLUT outputs	16	1	15	16	1	15

DDST receiver described [3], the module corresponding to the channel estimation, only the arithmetic mean was accelerated by dedicated hardware. The matrix-vector operation indicated in Eq. 5 was implemented in software only. The comparison with the systolic channel estimator (SYSDCE) presented in [16] is direct because the same FPGA was used. Table 2 lists the synthesis results for both channel estimators. The values in the parenthesis in each feature indicate the total resources in the FPGA. The results in Table 2 reveal that the frequency operation decreased only marginally in CEOF architecture even though the cyclic mean unit is serial. Similar behavior is observed for the number of slice registers, slice LUTs, block RAMs and DSP48E blocks. One might think that the CEDAM system would consume more resources in the FPGA, but the hardware complexity is almost the same; this is because many of the FPGA internal resources are shared among the other modules inside the architecture. For example, a slice register can “donate” its remaining registers to another entity, likewise a slice LUT multiplexer can share its resources with other modules. Another reason is that the consumptions of both architectures are minimal with respect to the total resources of the FPGA. Note that Table 2 does not include comparison with previous implementations described in [3, 14] because there were no data concerning the items listed in that Table.

Now, it is important to determine an approximation of the number of cycles in both architectures for computing the estimated CIR coefficients. If a single-MB is used in CEDAM estimator then number of cycles is defined by:

$$\text{cycles}_h = (N + P) + N_P + 2P - 1, \quad (13)$$

where the first term corresponds to the previous data storage in MB. This latency is eliminated when the “ping-pong” scheme—showed in Fig. 5—is used jointly with CEDAM, hence Eq. 13 becomes

$$\text{cycles}_h = N_P + 2P - 1. \quad (14)$$

On the other hand, the number of cycles in CEOF is given by:

$$\text{cycles}_h = (N + P) + 2P - 1 \quad (15)$$

Analyzing the set of metrics listed in Table 3, it is possible to compare the performance in both designs. The channel estimation time (CET) is the time elapsed from the beginning of the channel estimation process until its computing has finished. Similarly, the time necessary to calculate the cyclic time is denoted as cyclic mean time (CMT). The throughput (TP) per area is another useful metric for comparison, a higher value of this

Table 2 FPGA implementation results of the CEDAM and CEOF channel estimators.

Channel estimator	Single-MB CEDAM	Double-MB CEDAM	CEO	SYSDCE [16]
Input length (without C)	(N)	512	512	512
Frequency	(MHz)	161.520	161.520	160.190
Slice registers	(69120)	2,371 (3 %)	2,371 (3 %)	1,217 (1 %)
Slice LUTs	(69120)	929 (1 %)	1,194 (1 %)	937 (1 %)
Block RAMs	(148)	8 (5 %)	16 (10 %)	1 (<1 %)
DSP48Es	(64)	32 (50 %)	32 (50 %)	32 (50 %)

Table 3 CEDAM and CEOF throughputs comparison (Virtex-5 FPGA).

Channel estimator	Operation mode	Input length	Cycles/cyclic mean	Cycles/estimation	CMT (us)	CET (us)	TP MS/s	TP/area (MS/s/slices)
CEDAM	Single-MB	512	–	599	–	3.708	138.06	41.83e3
CEDAM	Double-MB	512	–	79	–	0.489	1046.02	293.64e3
CEOF	Cyclic mean	512	520	–	3.246	–	157.73	73.22e3
CEOF	Channel estimation	512	–	537	–	3.352	152.72	70.90e3
SYSDCE [15]	Cyclic mean	512	593	–	5.145	–	101.06	25.539e3
SYSDCE [15]	Channel estimation	512	–	610	–	5.293	98.243	24.828e3
Arithmetic mean in [3]	Coprocessor	NA	2,238	–	20	–	–	–

ratio indicates that the system implementation is better. Table 3 shows that the double-MB CEDAM system has the best performance in all metrics but with the penalty of memory consumption. There are no drastic differences between single-MB CEDAM and CEOF estimators. As expected, the proposed architectures provide far better performance against the arithmetic mean coprocessor used in [3].

5.1.2 VLSI Implementation

The RTL code used to validate the FPGA implementations of the single-MB CEDAM and CEOF estimators were synthesized (with minimum modifications) and placed&routed (P&R) using a generic CMOS standard

cell 90 nm technology with 9 metal layers (Synopsys SAED 90 nm). Figure 10 depicts the VLSI layouts of the aforementioned architectures and Table 4 shows their comparison in terms of:

- Used silicon area (assuming a placement utilization factor of 70 %).
- Complexity in terms of equivalent nand2 gates (assuming a nand2 gate occupies $5.8 \mu\text{m}^2$)
- Maximum operating frequency attainable after P&R (assuming a 150 MHz clock frequency and looking for the worst slack to compute the time margin still available).
- Dynamic power reported by the static timing analysis tool (assuming a typical gate switching rate of 10 %).

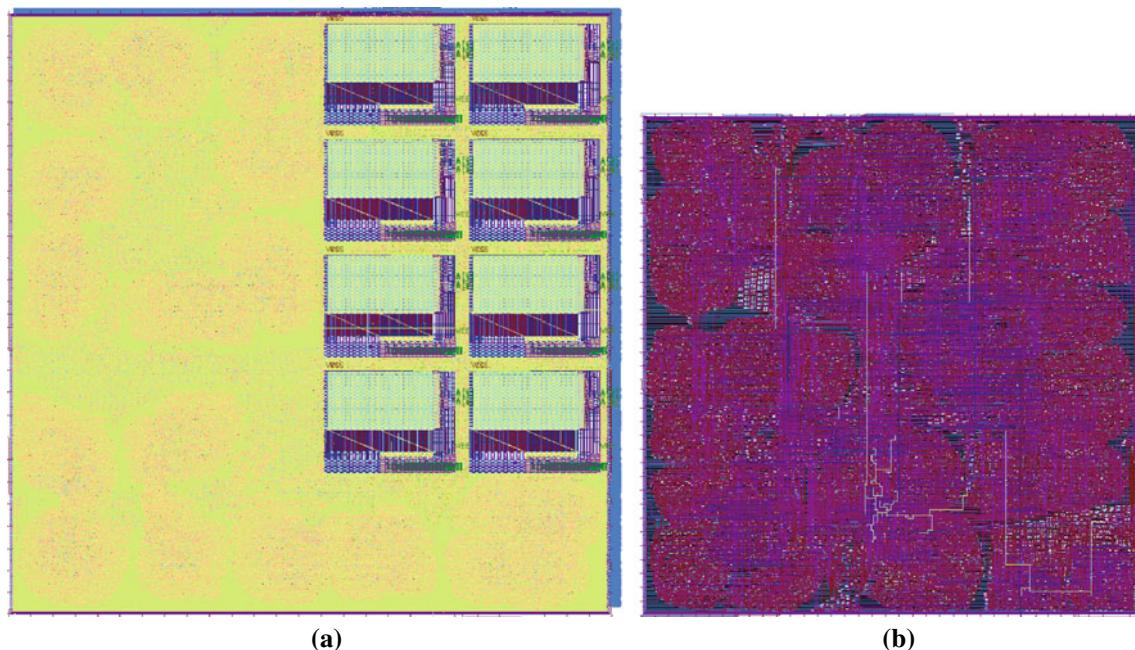
**Figure 10** Layouts of the proposed channel estimators; **a** Single-MB CEDAM, **b** CEOF.

Table 4 VLSI implementations comparison of the single-MB CEDAM and CEOF estimators.

		Single-MB	CEOF	Δ (%)
Area	(μm^2)	823,096	543,544	-33.96
Complexity	(Gates)	141,913	93,715	-33.96
Maximum frequency	(MHz)	187	245	31.02
Dynamic power	(mW)	3.70	2.74	-25.83

Operation conditions: 1.2V, 25 °C

Interconnect model: balance tree

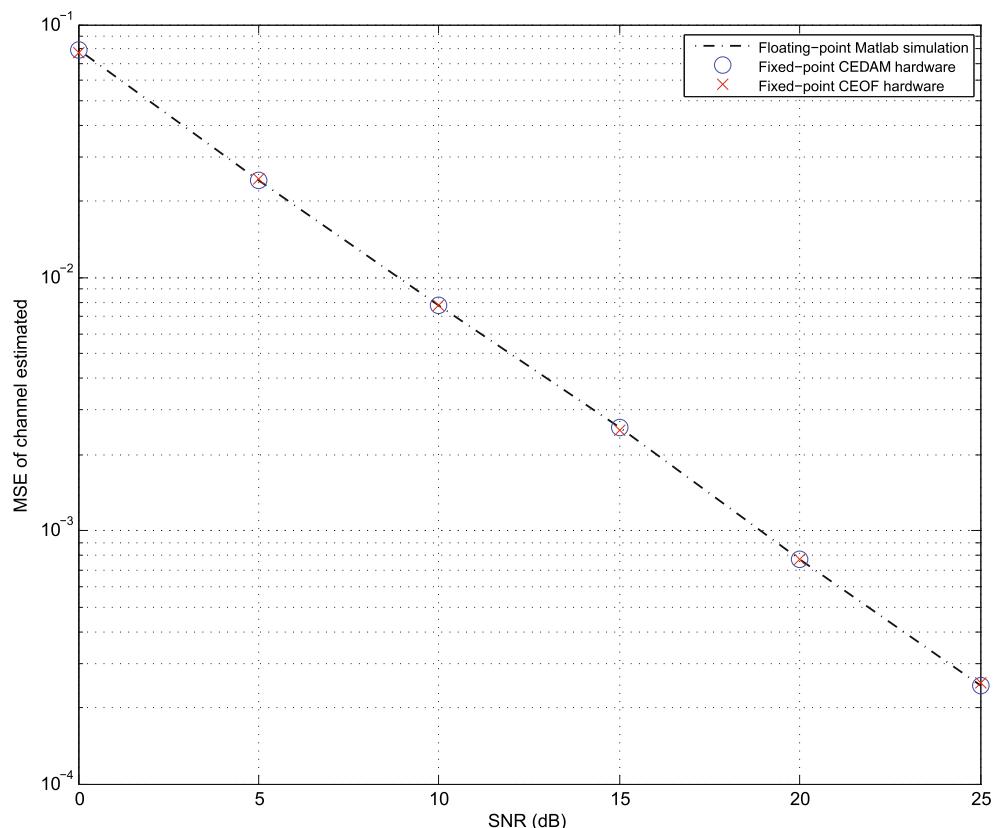
It should be noted that the results presented in Table 4 embody a typical synthesis and P&R process; no efforts were made to improve any of the parameters shown in that Table other than running the processes with the default settings.

In a fair comparison, it can be observed in Table 4 that the CEOF architecture is 34 % smaller when compared to the single-MB CEDAM architecture, 31 % faster and consumes 25 % less power. The single-MB CEDAM architecture makes use of eight banks of 64 words by 32 bit static ram memories. In order to make the area usage more efficient, these were implemented

as hard macros of cells available in the used technology library. Of the total area used of 823,096 μm^2 , the memories use a real state of 334,518 μm^2 which represent 40.64 % of the total area. This is one of the reasons we do not compare just the random logic of the architecture that could be misleading. The CEOF architecture uses memories as well but the requirements are so small (8 words by 32 bits) that have been synthesized using Flip-Flops. Even that a fair comparison between the FPGA and ASIC cannot be obtained due to the difference in technologies (65 vs. 90 nm) and also due to the utilization of components such as DSP48 and block RAMs which are tailored to increase performance and efficiency on FPGAs [11], it was corroborated that the CEOF architecture is more efficient.

5.1.3 Simulation Results

The validity of the architectures provided here is granted by comparing their results with the floating-point golden model, in terms of the mean squared error (MSE) and signal-to-noise ratio (SNR). Therefore, the following scenario (similar to the one used in [6]) was considered. The length of data input is $N = 512$

Figure 11 MSE performance of the CEDAM and CEOF architectures.

4-QAM symbols. The channel is randomly generated at each Monte Carlo run and is assumed to be Rayleigh with $L = 8$. The power of the training sequence is set to $0.2\sigma_s^2$. This sequence possesses a period $P = L$. Also, perfect block synchronization is assumed.

The MSE performance of the CEDAM and CEOF architectures for 300 Monte Carlo trials is shown in Fig. 11. Note that the MSE of the hardware estimators is almost indistinguishable from the golden model. The signal-to-quantization-noise ratio (SQNR) performance histograms of these estimators obtained for the same Monte Carlo trials is presented in Fig. 12. It can be observed that CEDAM and CEOF histograms are identical because the same wordlengths and word formats were used in both architectures. The SQNR ranges in both estimators for cyclic mean estimation and channel estimation are about of 12 dB and the higher mean values take place near 80 and 65 dB respectively.

5.2 Fixed-Point Functional Simulation of the ECEOF Estimator

A fixed-point simulation model of the ECEOF architecture was implemented in Matlab. The model reproduces all the data streams in a similar way as in the RTL level architecture. As mentioned earlier, the goal is to evaluate the SQNR in each estimated variables and the MSE performance of the estimated CIR. In the ECEOF simulation, the following scenario was considered. The length of data input is two blocks of $512 + 16$ 4-QAM symbols. The channel with length $L = 7$ is ran-

Table 5 Wordlength and word format of representative variables of the ECEOF architecture.

Variables	Integer bits	Fractional bits
Received sequence	4	14
TSS/BS functional	10	28
DC-offset	1	15
τ	4	0
τ_p	5	0
Cyclic mean	3	15
CIR coefficients	4	14

domly generated at each realization and its coefficients were complex Gaussian, i.i.d. with unitary variance and rescaled to achieve unitary mean energy. A zero-mean white Gaussian noise, a random DC-offset in the range of $0 \leq m \leq 0.3$ and a random synchronization offset between 0 and $N + P - 1$ were introduced at each Monte Carlo trial. The period of training sequence was set to $P = 16$ and its power was set to $0.167\sigma_s^2$. The length of the cyclic prefix was $CP = 16$. 1,000 Monte Carlo trials were evaluated. The resulting wordlength format for some representative variables in the fixed-point version of the algorithm is shown in Table 5.

In order to measure the performance of the fixed-point ECEOF architecture, the SQNR probability density function (PDF) of the TSS functional, BS functional, DC-offset estimation and true-channel coefficients were estimated (Fig. 13) from the histogram plot which classifies 6,000 SQNR simulation results (1,000 realizations for 0, 5, 10, 15, 20, 25 dB SNR values). The amount of classes (14 classes) of

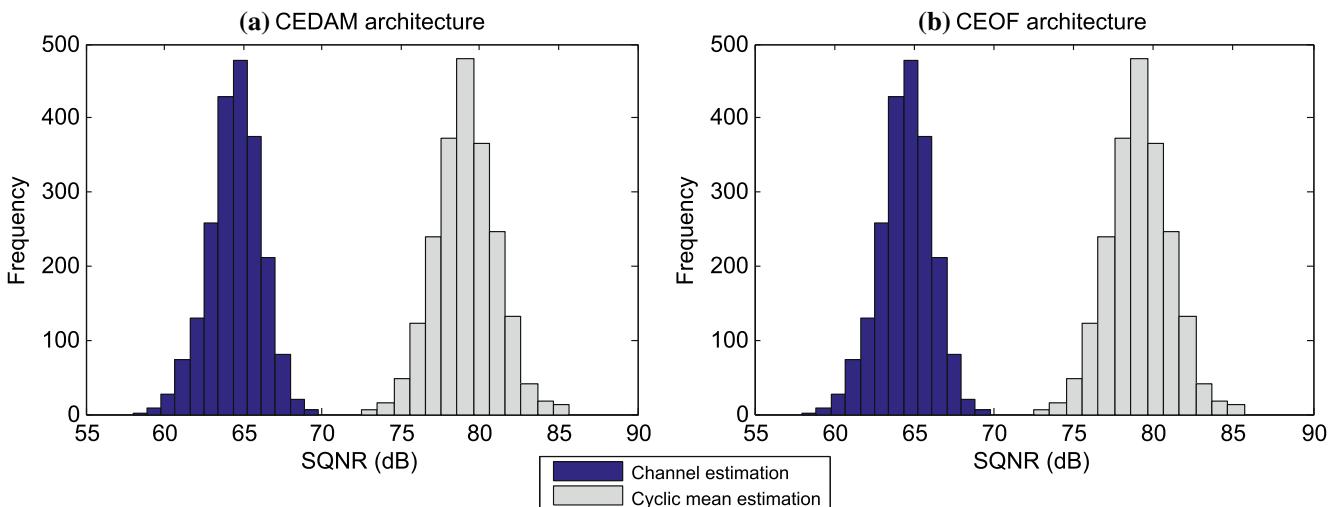


Figure 12 SQNR performance histograms of the implemented channel estimators for 300 Monte Carlo trials; **a** CEDAM architecture, **b** CEOF architecture.

Figure 13 SQNR probability density functions of the TSS functional, BS functional, DC-offset estimation and true-channel estimation in the ECEOF estimator.

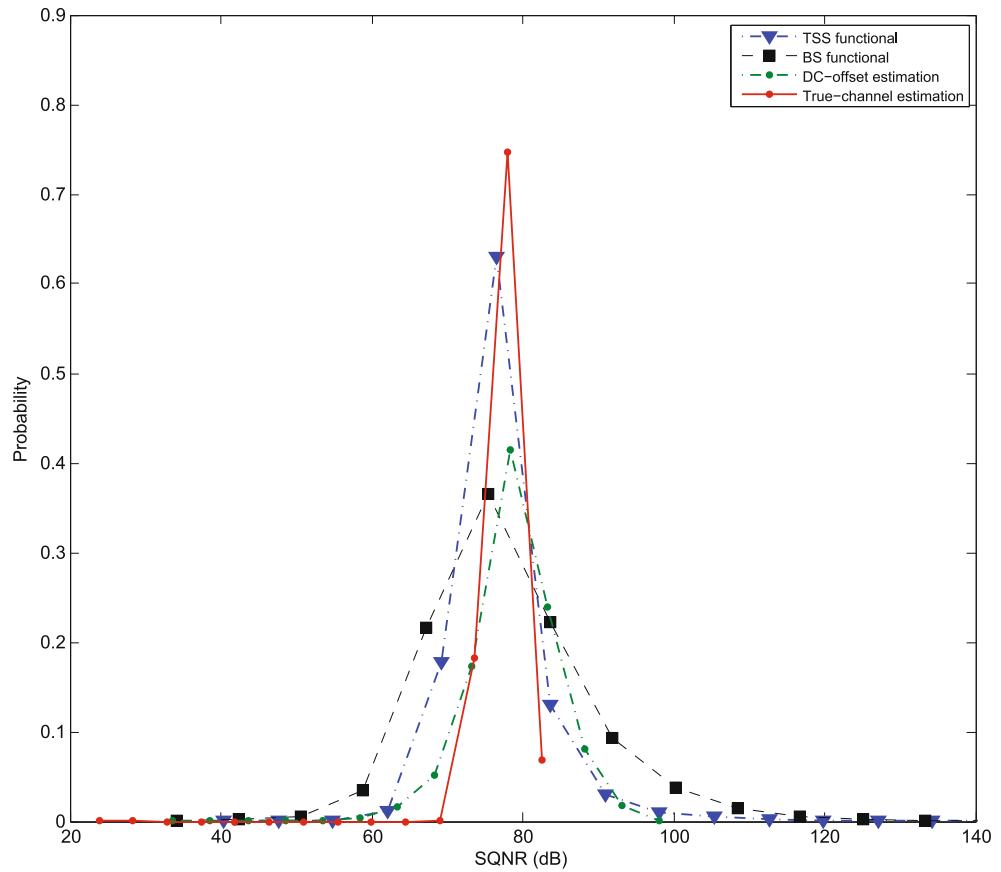
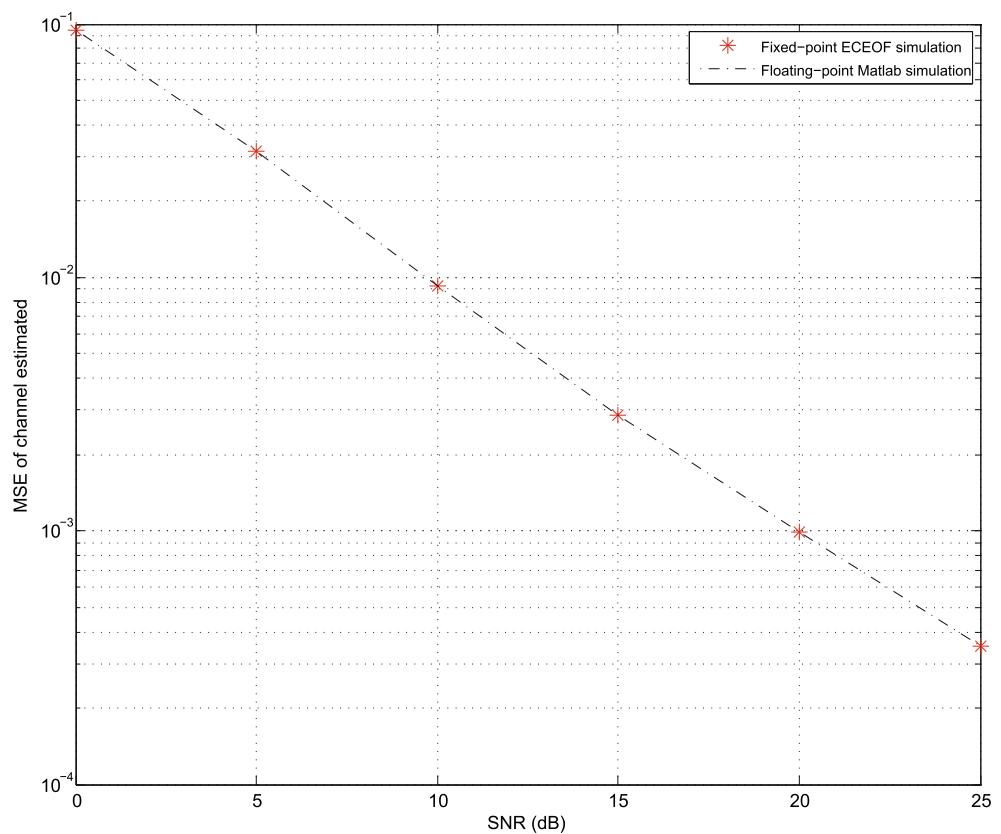


Figure 14 MSE performance of the fixed-point simulation of the ECEOF architecture for 1,000 Monte Carlo trials.



such histogram was calculated according to the Stuger's rule. It should be pointed out that in all PDFs, the higher probability of occurrence is near 76 dB, which represents a similar performance to that achieved in the floating-point model. The average SQNR of the true-channel estimation is 77.47 dB.

The MSE's of the channel estimated for both the floating-point golden model and the fixed-point simulation of the ECEOFOF architecture, are shown in Fig.14. From this Figure, it can be noted that the MSE obtained from ECEOFOF is practically the same as one obtained with the golden model. This behavior demonstrates the proper operation of the proposed estimator.

6 Conclusions

In this paper, three architectures for DDST channel estimation have been introduced. The first two, CEDAM and CEOF, represents the first channel estimators implementations under the full-hardware philosophy for a DDST receiver. The last one (ECEOFOF) is the first full-hardware architecture proposal for DDST channel estimation under more realistic conditions (nonsynchronization and unknown DC-offset). The CEDAM and CEOF designs present high throughputs and reduced FPGA resources consumption, achieving a good trade-off between performance and area utilization. Likewise, their VLSI implementations results proved silicon efficiency, very high operating frequencies and low power consumptions. The validity and performance of these architectures have been verified by Monte Carlo simulations, where an SQNR of 65 dB is achieved, while at the same time, MSE performance practically equal to the floating-point golden model is guaranteed. Concerning the ECEOFOF estimator, the fixed-point analysis shows its correct functionality and excellent SQNR results. The foregoing thus establishes that ST/DDST concepts can be effectively utilized in future communications standards. Remaining to be studied is the maximum throughput that can be achieved using a particular channel equalizer, which is our planned future work.

Acknowledgements This work was supported by PROMEP ITSON-092 and CONACYT 181962 research grants.

References

1. Alameda-Hernandez, E., McLernon, D., Orozco-Lugo, A., Ghogho, M. (2005). Synchronisation and dc-offset estimation for channel estimation using data-dependent superimposed training. In *European Signal Processing Conference (EU-SIPCO 2005)*.
2. Alameda-Hernandez, E., McLernon, D., Orozco-Lugo, A., Lara, M., Ghogho, M. (2007). Frame/training sequence synchronization and dc-offset removal for (data-dependent) superimposed training based channel estimation. *IEEE Transactions on Signal Processing*, 55(6), 2557–2569.
3. Martín del Campo-Ramírez, F. (2008). *Data-dependent superimposed training architecture for wireless communication systems*. Master's thesis, INAOE.
4. Carrasco-Alvarez, R., Parra-Michel, R., Orozco-Lugo, A.G., Tugnait, J.K. (2009). Enhanced channel estimation using superimposed training based on universal basis expansion. *IEEE Transactions on Signal Processing*, 57(3), 1217–1222.
5. Farhang-Boroujeny, B. (1995). Pilot-based channel identification: proposal for semi-blind identification of communication channels. *Electronics Letters*, 31(13), 1044–1046.
6. Ghogho, M., McLernon, D., Alameda-Hernandez, E., Swami, A. (2005). Channel estimation and symbol detection for block transmission using data-dependent superimposed training. *IEEE Signal Processing Letters*, 12(3), 226–229.
7. Ghogho, M., & Swami, A. (2004). Improved channel estimation using superimposed training. In *Proc. IEEE 5th workshop on signal processing advances in wireless communications* (pp. 110–114).
8. Goljahani, A., Benvenuto, N., Tomasin, S., Vangelista, L. (2009). Superimposed sequence versus pilot aided channel estimations for next generation dvb-t systems. *IEEE Transactions on Broadcasting*, 55(1), 140–144.
9. Haykin, S., & Ray Liu, K. (2009). *Handbook on array processing and sensor networks*. Wiley.
10. Kung, S. (1985). *VLSI array processors*. Prentice Hall.
11. Kuon, I., & Rose, J. (2007). Measuring the gap between fpgas and asics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2), 203–215.
12. Longoria-Gandara, O., Parra-Michel, R., Bazdresch, M., Orozco-Lugo, A. (2008). Iterative mean removal superimposed training for SISO and MIMO channel estimation. *International Journal of Digital Multimedia Broadcasting*, vol. 2008, Article ID 535269, 9 pages, 2008. doi:[10.1155/2008/535269](https://doi.org/10.1155/2008/535269).
13. Moldovan, D. (1993). *Parallel processing from applications to systems*. Morgan Kaufman Publishers, Inc.
14. Najera-Bello, V. (2008). *Design and construction of a digital communications system based on implicit training*. Master's thesis, CINVESTAV-IPN.
15. Orozco-Lugo, A., Lara, M., McLernon, D. (2004) Channel estimation using implicit training. *IEEE Transactions on Signal Processing*, 52(1), 240–254.
16. Romero-Aguirre, E., Parra-Michel, R., Carrasco-Alvarez, R., Orozco-Lugo, A. (2011). Architecture based on array processors for data-dependent superimposed training channel estimation. In *2011 international conference on, reconfigurable computing and FPGAs (ReConFig)* (pp. 303–308).
17. Romero-Aguirre, E., Parra-Michel, R., Orozco-Lugo, A.G., Carrasco-Alvarez, R. (2011). Full-hardware architectures for data-dependent superimposed training channel estimation. In *SiPS* (pp. 49–54).
18. Tugnait, J., & Luo, W. (2003). On channel estimation using superimposed training and first-order statistics. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '03)* (Vol. 4, pp. IV–624–7).
19. Tugnait, J., & Meng, X. (2006). On superimposed training for channel estimation: performance analysis, training power allocation, and frame synchronization. *IEEE Transactions on Signal Processing*, 54(2), 752–765.



Eduardo Romero-Aguirre received the B. Sc. degree in Electronics Instrumentation option at the Orizaba Technology Institute (ITO) in 1995, and the M. Sc. degree in Digital Electronic Systems at the National Center for Research and Technology Development (CENIDET). He has carried out diverse projects related systems of acquisition of data. He has worked a full member of research staff at the Sonora Technology Institute (ITSON) since January 2000. Actually he is a Ph.D. student at the Research Center for Advanced Studies of the National Polytechnic Institute (CINVESTAV-IPN). His research interest include communications system based on implicit training, digital implementation of DSP algorithms for communication systems and digital systems design based on microprocessors and microcontrollers.



Ramón Parra-Michel was born in Guadalajara City, Mexico, in 1973. He received the B.Sc. degree in electronics and communications from University of Guadalajara, Jalisco, Mexico, in 1996, the M.Sc. degree in electrical engineering, specializing in communications, from CINVESTAV-IPN, Guadalajara, Mexico, in 1998 and the Ph.D. degree in electrical engineering specialized in digital signal processing for communications from CINVESTAV-IPN, Mexico City, in 2003. He has collaborated with several companies and institutions either in academic or technology projects, such as Siemens, Lucent, Mabe, Mixbaal, Hewlett-Packard and Intel. He is currently a full member of research staff at CINVESTAV-IPN in Guadalajara Unit. His research interests include modeling, simulation, estimation and equalization of communication channels, and digital implementation of DSP algorithms for communication systems.



Roberto Carrasco-Alvarez was born in Mexico City, Mexico, in 1981. He received the B.Sc. degree in electronics from Morelia Technology Institute, Morelia, Michoacan, Mexico, in 2004, the M.Sc. degree in electrical engineering, specializing in telecommunications, from CINVESTAV-IPN, Guadalajara, Mexico, in 2006. He received the Ph.D. degree in electrical engineering from CINVESTAV-IPN. His research interests include signal processing and digital communications. He is currently member of the research staff at CUCEI-University of Guadalajara.



Aldo Orozco-Lugo (S'96–M'00) was born in Guadalajara City, Mexico, in 1970. He received the B.Sc. degree in electronics and communications from University of Guadalajara, Guadalajara, Jalisco, Mexico, in 1993, the M.Sc. degree in electrical engineering, specializing in communications, from CINVESTAV-IPN, Mexico City, Mexico, in 1997 and the Ph.D. degree in electrical engineering specialized in digital signal processing for communications from the University of Leeds, Leeds, U.K., in 2000.

He is currently an associate member of research staff at CINVESTAV-IPN. His research interests include space-time signal processing, wideband channel modeling, analog and digital communication systems, antenna array technology, and radar.



Antonio F. Mondragón-Torres received the B.Sc. degree with honors from Universidad Iberoamericana, Mexico City, Mexico, the M.Sc. degree from Universidad Nacional Autónoma de Mexico, Mexico City, Mexico, and the Ph.D. degree (as a Fullbright-CONACYT scholarship recipient) from Texas A&M University, College Station; all degrees in Electrical Engineering in 1990, 1996, and 2002, respectively. From 1988 to 1995, he worked in a telecommunications company TVS-COM, Mexico City, Mexico, designing teletext products, first as a Design Engineer and later as a Design Manager. In 1995, he joined the Mechanical and Electrical Department, Universidad Iberoamericana as an Associate Professor. From 2002 through 2008 he was with the DSPS R&D Center's Mobile Wireless Communications Technology branch, Texas Instruments Dallas, TX and in 2008 he moved to the nanoMeter Analog Integration Wireless branch where he worked as Analog IP verification technical lead. In 2009 he worked for Intel Guadalajara, Design Center in Mexico as Front-End/Back-End technical lead. In 2009 he joined the Electrical, Computer and Telecommunications Engineering Technology Department at the Rochester Institute of technology where he currently is a tenured track assistant professor. His research interests are analog and digital integrated circuit implementation of communications systems, and System-on-a-Chip methodologies.

Nomenclatura matemática

Símbolo	Definición
\mathbb{Z}	Conjunto de los números enteros
\mathbb{C}	Conjunto de los números complejos
$a[k]$	Señal discreta en el instante de tiempo k
$a(t)$	Señal continua en el instante de tiempo t
$re(a)$	Parte real de a
$im(a)$	Parte imaginaria de a
$conc(\cdot)$	Operador lógico de concatenación
a^*	Complejo conjugado de a
$*$	Operador de convolución
$\lfloor \cdot \rfloor$	Próximo valor entero hacia menos infinito
$\lceil \cdot \rceil$	Próximo valor entero hacia mas infinito
$\langle x \rangle_y$	Operador que indica el residuo de $\frac{x}{y}$
$ \cdot $	Valor absoluto
$E(\cdot)$	Operador de esperanza
$var(\cdot)$	Operador de varianza
$\delta(\cdot)$	Función delta de Dirac
$\sin(\cdot)$	Función seno
$\log_2(\cdot)$	Función logaritmo base 2
\mathbf{a}	Vector columna
$[\mathbf{a}]_i$	i -ésimo elemento de \mathbf{a}
$[\mathbf{A}]_{i,j}$	i -ésimo renglón y j -columna de \mathbf{A}
\mathbf{A}^T	Transpuesta de \mathbf{A}
\mathbf{A}^H	Transpuesta conjugada de \mathbf{A}
\mathbf{A}^{-1}	Inversa de la matriz cuadrada \mathbf{A}
$\mathbf{A}^{[n]}$	Primeras n filas de la matriz \mathbf{A}
$\mathbf{A}_{[n]}$	Ultimas n filas de la matriz \mathbf{A}
$\mathbf{0}_{a \times b}$	Matriz de dimensión $a \times b$ cuyos elementos son todos ceros
$\mathbf{1}_{a \times b}$	Matriz de dimensión $a \times b$ cuyos elementos son todos unos

Símbolo	Definición
\mathbf{I}_a	Matriz identidad de dimensión $a \times a$
$circ(\mathbf{a})$	Matriz cuadrada circulante conformada por el vector \mathbf{a}
$tr(\mathbf{A})$	Traza de la matriz cuadrada \mathbf{A}
$circshift_n(\mathbf{A})$	Desplazamiento circular de n posiciones hacia abajo de las filas de la matriz \mathbf{A} .
$\ \mathbf{a}\ $	Norma de \mathbf{a}
\otimes	Producto de Kronecker
\circledast	Operador de convolución circular
$\circledast\circledast$	Operador de correlación circular

Lista de símbolos y abreviaciones

Abreviación	Descripción	Definición
BRAM	Block RAM	pag. 55
BS	Block Synchronization	pag. 7
CD	Corriente Directa	pag. 6
CP	Cyclic Prefix	pag. 16
CEDAM	Channel Estimator with Data Feed from Memory	pag. 68
CEOFO	Channel Estimator with On-the-Fly Data Feed	pag. 69
CDMA	Code Division Multiple Access	pag. 15
CFO	Carrier Frequency Offset	pag. 7
CIR	Channel Impulse Response	pag. 11
CT	Correlation Time	pag. 131
dB	Decibel	pag. 60
DATINF	Data Input Feeder	pag. 75
DDST	Data Dependent Superimposed Training	pag. 7
DFT	Discrete Fourier Transform	pag. 29
DSP	Digital Signal Processor	pag. 8
ECEOFO	Extended CEOF	pag. 87
FF	Flip–Flop	pag. 130
FDCP	Frequency–Domain Correlator Processor	pag. 125
FFT	Fast Fourier Transform	pag. 11
FIR	Finite Impulse Response	pag. 22
FSM	Finite State Machine	pag. 58
CGCU	Correlator General Control Unit	pag. 125
GSM	Global System for Mobile communication	pag. 6
HDL	Hardware Description Languaje	pag. 47
Hyper-LAN	HIgh PErfomance Radio LAN	pag. 15

Abreviación	Descripción	Definición
ICLUT	Inverse C Look-Up Table	pag. 69
IDFT	Inverse Discrete Fourier Transform	pag. 37
IFC	IFFT/FFT Control	pag. 127
IFFT	Inverse Fast Fourier Transform	pag. 37
IT	Implicit Training	pag. 6
LSB	Least Significant Bit	pag. 58
MFB	Maximization of Frequency Bins	pag. 26
MIMO	Multiple Input Multiple Output	pag. 7
MRST	Iterative Mean Removal Superimposed Training	pag. 7
MSB	Most Significant Bits	pag. 78
MSMVM	Modified Systolic Matrix–Vector Multiplicator	pag. 75
MVM	Matrix–Vector Multiplication	pag. 71
OFDM	Orthogonal Frequency Division Multiplexing	pag. 7
P&R	Placed&Routed	pag. 81
PAPR	Peak-to-Average Power Ratio	pag. 7
PAT	Pilot Assisted Training	pag. 15
PCME	Parallel Cyclic Mean Estimador	pag. 69
PDF	Probability Density Function	pag. 83
PDP	Power Delay Profile	pag. 32
PDS	Procesamiento Digital de Señales	pag. 9
QPSK	Quadratic Phase-Shift Keying	pag. 45
PE	Processor Element	pag. 71
PT	Processing Time	pag. 80
RBE	Reconfigurable Radix-2 Butterfly Element	pag. 127
RIFP	Reconfigurable IFFT/FFT Processor	pag. 125
RTL	Register Transfer Level	pag. 47
SCME	Serial Cyclic Mean Estimador	pag. 72
SDD	Secuencia Dependiente de los Datos	pag. 16
SISO	Single Input Single Output	pag. 7
SMVM	Systolic Matrix–Vector Multiplicator	pag. 69
SYSDCE	Systolic DDST Channel Estimator	pag. 69
ST	Superimposed Training	pag. 6
TB	Temporal Buffer	pag. 90
TDMA	Time Division Multiple Access	pag. 15
TP	Throughput	pag. 81
TSS	Training Sequence Sincronization	pag. 6
UB	Universal Basis	pag. 32

Lista de figuras

1.1.	Diagrama a bloques simplificado de un sistema de comunicación.	5
1.2.	Niveles de jerarquía de los diversos enfoques existentes para implementar un sistema de procesamiento digital.	9
2.1.	Taxonomía de las técnicas de estimación de canal.	14
2.2.	Posición de los pilotos de entrenamiento en sistemas inalámbricos. (a) transmisiones por paquetes y (b) transmisiones continuas. Las áreas sombreadas indican los pilotos (de [44] ©IEEE, 2004).	15
2.3.	Secuencia de datos con entrenamiento superimpuesto.	16
2.4.	Secuencia de datos con entrenamiento superimpuesto dependiente de los datos.	16
2.5.	Diagrama a bloques de un transmisor digital con IT.	17
2.6.	Modelo de canal inalámbrico y perturbaciones más comunes.	19
2.7.	Diagrama a bloques de un receptor con IT.	19
3.1.	Modelo discreto de sistema de comunicación pasa-baja basado en DDST. .	22
3.2.	Sistema de comunicación invariante, discreto y pasa-baja basado en ST/DDST. .	34
3.3.	Estructura del <i>butterfly</i> DIT-2.	38
3.4.	Flujograma de una FFT decimada en tiempo de 8–puntos.	38
3.5.	Estructura del <i>butterfly</i> DIF-2.	39
3.6.	Flujograma de una FFT decimada en frecuencia de 8–puntos.	40
3.7.	Estructura del <i>dragonfly</i> DIF-4: a) Completa, b) Representación simplificada. .	40
4.1.	Arquitectura simplificada de <i>hardware</i> del transmisor ST/DDST configurable. .	48
4.2.	Constelaciones con las que puede operar el transmisor ST/DDST; a) 4–QAM, b) 16–QAM, c) 64–QAM (se delimitan también las constelaciones anteriores).	50
4.3.	El adecuador de símbolos; a) Diagrama, b) Tabla de verdad para adecuación de símbolos QPSK, c) Tabla de verdad para adecuación de símbolos 16–QAM. .	51
4.4.	Arquitectura interna del mapeador.	53
4.5.	ROM para la LUT con la secuencia de entrenamiento.	55

4.6.	Secuencia ST con prefijo cíclico incorporado.	55
4.7.	Submódulo de inserción de prefijo cíclico; a) Implementación con memoria RAM con buses de direccionamiento separado para lectura y escritura, b) Proceso de almacenamiento de la secuencia de datos e inserción del prefijo cíclico.	56
4.8.	Proceso para generar la secuencia de entrenamiento dependiente de los de datos a partir de la secuencia $b(k)$	57
4.9.	Submódulo generador de la secuencia de entrenamiento dependiente de los de datos; a) Arquitectura simplificada, b) Proceso de replicación de $e(j) \cdot N_P$ y división entre N_P para formar $e(k)$	57
4.10.	Estructura interna del generador de direcciones del transmisor ST/DDST. . .	58
4.11.	Esquema general de verificación para las arquitecturas de <i>hardware</i>	60
4.12.	Secuencia de entrenamiento obtenida con el <i>hardware</i> del transmisor y por medio de simulación con Matlab; a) Parte real, b) Parte imaginaria.	60
4.13.	Secuencia de salida ST obtenida tanto con la arquitectura del transmisor como por medio del modelo de simulación con Matlab; a) Parte real, b) Parte imaginaria.	61
4.14.	Transformada discreta de Fourier de la secuencia ST del transmisor. . . .	61
4.15.	Secuencia dependiente de los datos $e(n)$ obtenida tanto con la arquitectura del transmisor como por el modelo de simulación; a) Parte real, b) Parte imaginaria.	62
4.16.	Secuencia de salida DDST obtenida tanto con la arquitectura del transmisor como por medio del modelo de simulación con Matlab; a) Parte real, b) Parte imaginaria.	63
4.17.	Transformada discreta de Fourier de la secuencia DDST del transmisor. .	63
4.18.	Histograma del SQNR de la arquitectura del transmisor ST/DDTS para 100 simulaciones Monte Carlo.	64
5.1.	Arquitectura simplificada de un receptor DDST. (Por legibilidad se ha omitido los módulos generadores de direcciones y de control).	68
5.2.	Arquitectura simplificada del estimador de canal CEDAM.	69
5.3.	Organización de los datos en el <i>buffer</i> de memoria.	70
5.4.	Esquema de direccionamiento <i>hard-wired</i> para el <i>buffer</i> de memoria. . . .	71
5.5.	Arreglo de registros de la LUT para \mathbf{C}^{-1} y sus correspondientes valores de salida.	72
5.6.	Arquitectura de <i>hardware</i> del estimador de canal CEOF.	73
5.7.	Estimador de canal sistólico para un receptor DDST; a) Arquitectura de <i>hardware</i> , b) Estructura interna de su PE.	76
5.8.	Banco de memoria y organización de los datos en el DATINF.	77
5.9.	Direccionamiento de tipo <i>hard-wired</i> para el banco de memoria del alimentador de datos.	78

5.10. Esquemas de diseño VLSI de los estimadores de canal propuestos; a) CEDAM, b) CEOF	82
5.11. Diagrama a bloques del esquema de verificación para las arquitecturas de los estimadores CEDAM, CEOF y SYSDCE.	83
5.12. Estimación de la media cíclica realizada por el módulo SCME del estimador CEOF <i>versus</i> modelo de simulación en Matlab; a) Parte real, b) Parte imaginaria.	84
5.13. Estimación del canal realizada por el estimador CEOF <i>versus</i> modelo de simulación en Matlab; a) Parte real, b) Parte imaginaria.	84
5.14. Rendimiento del MSE de las arquitecturas de estimadores <i>versus</i> el obtenido con el modelo de simulación en Matlab.	85
5.15. Histogramas del rendimiento SQNR para 300 simulaciones Monte Carlo de las arquitecturas de estimadores de canal DDST; a) CEDAM, b) CEOF, c) SYSDCE.	86
5.16. Arquitectura simplificada de <i>hardware</i> del estimador de canal ECEOFO.	89
5.17. Arquitectura interna del módulo AGU_MB.	92
5.18. Arquitectura interna del módulo AGU_SYNC.	92
5.19. El módulo FUNC_TERM_A; a) Arquitectura, b) Temporización.	94
5.20. El módulo FUNC_TERM_B; a) Arquitectura, b) Temporización.	95
5.21. Arquitectura interna del módulo FMIN.	96
5.22. Esquema de verificación para la arquitectura del estimador de canal ECEOFO.	100
5.23. Cronograma de operación del estimador de canal ECEOFO (señales en forma digital).	101
5.24. Cronograma de operación del estimador de canal ECEOFO (visualización de señales en forma analógica).	102
5.25. Diagrama de tiempo del proceso de corrección de TSS en el estimador ECEOFO.	103
5.26. Cronograma de operación del funcional de la norma cuadrada durante el proceso de corrección de TSS.	103
5.27. Señales del funcional de la norma cuadrada durante una de sus evaluaciones dentro del proceso de corrección de BS.	104
5.28. Señales del estimador ECEOFO para el proceso de estimación verdadera de canal.	104
5.29. Estimados verdaderos realizados por el estimador ECEOFO <i>versus</i> modelo de simulación en Matlab; a) Media cíclica, b) Canal.	105
5.30. Rendimiento del MSE del canal estimado de la arquitectura ECEOFO <i>versus</i> el obtenido con el modelo de simulación en Matlab.	106
5.31. SQNR individual, promedio e histograma para 300 simulaciones Monte Carlo del funcional de la norma cuadrada del estimador ECEOFO; a) y b) Corrección de TSS, c) y d) Corrección de BS.	107

5.32. SQNR individual, promedio e histograma para 300 simulaciones Monte Carlo de las salidas del estimador ECEO; a) y b) Desplazamiento de CD, c) y d) Estimado verdadero de la media cíclica, e) y f) Estimado verdadero del canal.	108
5.33. Diagrama a bloques de un procesador FFT basado en memoria.	109
5.34. Arquitectura digital para el procesador FFT/IFFT radix-2 con decimado en tiempo.	110
5.35. Estructura interna del módulo de reordenamiento; a) Interconexión directa, b) Interconexión inversa.	111
5.36. Estructura interna del <i>butterfly</i> DIT-2.	112
5.37. Arquitectura del módulo generador de direcciones para la FFT DIT-2.	113
5.38. Máquina de estados finitos del control DIT2.	114
5.39. Arquitectura digital para el procesador FFT/IFFT radix-2 con decimado en frecuencia.	115
5.40. <i>Hardware</i> para el cálculo de <i>temp</i> usando (5.27) en el AGU_DIF2.	116
5.41. Estructura interna del <i>butterfly</i> DIF-2	116
5.42. Máquina de estados finitos del control DIF2.	117
5.43. Comparación de la FFT de una señal impulso unitario calculada con el procesador FFT DIT-2 <i>versus</i> Matlab ; b) Parte real, c) Parte imaginaria.	119
5.44. Comparación de la FFT de una señal de magnitud unitaria constante calculada con el procesador FFT DIT-2 <i>versus</i> Matlab ; b) Parte real, c) Parte imaginaria.	119
5.45. FFT de la señal $x(n) = 0.4 \cos(0.1\pi n) + 0.55 \cos(0.25\pi n)$ obtenida con el procesador FFT DIF-2 <i>versus</i> Matlab ; b) Parte real, c) Parte imaginaria.	120
5.46. Funciones de densidad de probabilidad del SQNR en los procesadores FFT con $N = 1024$; a) Para arquitectura FFT DIF-2 con $Q_W = 16$ bits, b) Para arquitectura FFT DIT-2 con $Q_W = 18$ bits.	120
5.47. Comportamiento del SQNR en la arquitectura FFT DIT-2 de $N = 1024$ muestras ante diferentes anchos de palabra.	121
5.48. Comportamiento del SQNR en la arquitectura FFT DIF-2 para diferentes números de muestras y con anchura de palabra constante.	122
5.49. Diagrama a bloques del flujo de datos en la correlación en el dominio de la frecuencia.	123
5.50. Diagrama a bloques de la correlación en el dominio de la frecuencia sin permutaciones usando FFTs/IFFT con radix-2.	125
5.51. Arquitectura simplificada del correlacionador en el dominio de la frecuencia.	126
5.52. Estados de la CGCU y su duración.	126
5.53. Arquitectura del RIFP.	127
5.54. Arquitectura del <i>butterfly</i> radix-2 con decimado reconfigurable.	128
5.55. Máquina de estados finitos del IFC.	128
5.56. <i>Hardware</i> para el cálculo de <i>temp</i> en el AGU_RIFP.	129

- 5.57. Gráfica de SQNR *versus* ancho de palabra en la arquitectura del FCDP para distintas longitudes de correlación. 132

Lista de tablas

4.1.	Tabla de verdad del transmisor ST/DDST.	49
4.2.	Cuantización de las constantes del mapeador para $\sigma_c^2 = 0.2$ y $Q_W = 16$ bits.	54
4.3.	Cuantización de la secuencia de entrenamiento para $\sigma_c^2 = 0.2$, $P = 8$, $N = 512$ y $Q_W = 16$ bits.	54
4.4.	Resultados de síntesis para el transmisor ST/DDST con $\sigma_c^2 = 0.2$, $P = 8$, $N = 512$ y $Q_W = 16$ bits.	59
5.1.	Longitudes y formatos de palabra usados en las arquitecturas CEDAM, CEOF y SYSDCE.	79
5.2.	Resultados de síntesis para los estimadores de canal propuestos ($N = 512$, $P = 8$ y $CP = P$).	80
5.3.	Tabla comparativa del rendimiento en los estimadores propuestos.	81
5.4.	Comparación de las implementaciones VLSI de los estimadores CEDAM y CEOF.	82
5.5.	Rangos dinámicos de las variables relevantes del algoritmo para el estimador ECEO. (Nota *: Es el segundo valor mínimo (antecesor); Nota **: Es la diferencia entre el valor mínimo y su antecesor).	96
5.6.	Longitudes y formatos de palabra empleados en las variables relevantes de la arquitectura ECEO.	97
5.7.	Resultados de síntesis de la arquitectura del estimador de canal ECEO en dos modelos de FPGA Virtex-5.	98
5.8.	Rendimiento del estimador ECEO implementado en el FPGA Virtex-5.	99
5.9.	Resultados de síntesis para procesadores FFT DIT-2 y FFT DIF-2, para tamaños de transformada de 512, 1024, 2048 y 4096 puntos y con ancho de palabra de 16 bits.	118
5.10.	Resultados de síntesis de FDCPs para secuencias cortas en el FPGA Cyclone II EP2C3F672C6 de Altera.	130
5.11.	Resultados de síntesis de FDCPs para secuencias largas en el FPGA Virtex II Pro de Xilinx.	130

5.12. Rendimiento de la arquitectura del FDCP sintetizada en el FPGA Virtex II Pro de Xilinx para diversas longitudes de correlación y anchos de palabras. 131