PROYECTO FIN DE GRADO

TÍTULO: Sistema de Preprocesado para Guerra Electrónica

AUTOR/A: Daniel Delgado Márquez

TITULACIÓN: Grado en Ingeniería de Sonido e Imagen

DIRECTOR/A: Raúl Cacho Martínez

TUTOR/A: Mario San Emeterio de la Parte

DEPARTAMENTO: Departamento de Ingeniería Telemática y Electrónica

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE/A: Daniel de la Prida Caballero

TUTOR/A: Mario San Emeterio de la Parte

SECRETARIO/A: Vicente Hernández Díaz

Fecha de lectura:

Calificación:

El Secretario/La Secretaria,

Resumen

El objetivo de este proyecto titulado “Sistema de Preprocesado para Guerra Electrónica” es el diseño e implementación de un preprocesador, cuyo objetivo es hacer más eficiente el análisis y procesado posterior de señales de radio frecuencia. Al mismo tiempo, optimiza el consumo energético y el ahorro de carga computacional. Está pensado para colaborar en cualquier ámbito, aunque centrándose en el militar y en las telecomunicaciones.

La solución propuesta se ha implementado mediante la simulación de señales con diferentes geometrías, y el uso de algoritmos y procesados de señales digitales, elegidos cuidadosamente tras realizar un análisis comprensivo de las herramientas y algoritmos más efectivos en este ámbito. Algunos de estos procesos son el uso de filtros digitales como el filtro FIR, o el uso de la Transformada Discreta de Fourier (FFT) junto con una esparsificación. Además, se ha llevado a cabo un análisis de los preprocesadores más destacados en la literatura actual, lo que ha aportado un gran valor técnico y estratégico al proyecto.

Desde el punto de vista tecnológico, el sistema se construye manteniendo una arquitectura modular, en la que cada etapa implementada es totalmente independiente, estando estructurado en funciones auxiliares, y siguiendo unas pautas marcadas para garantizar la máxima compatibilidad con HDL, ya que uno de los objetivos finales consiste en diseñar el sistema de forma que permita su futura integración en una FPGA.

En este proyecto se pretende seguir una metodología en cascada en tres fases, comenzando por un análisis del estado del arte de la EW, y de las diferentes tecnologías y arquitecturas. En segundo lugar, se abordará la implementación del código, y el diseño digital del simulador. Por último, se realizarán y documentarán las pruebas funcionales, que validarán el correcto funcionamiento del diseño, y el cumplimiento de los requisitos establecidos.

El enfoque de las pruebas se centra en realizar pruebas unitarias en cada microservicio, que consistirá en casos de uso aislados. Tras estas pruebas unitarias, el siguiente paso es realizar las pruebas de integración. Estas consisten en verificar cómo funciona la interacción entre microservicios, y dada la arquitectura modular, estas pruebas comprobarán el intercambio de información entre componentes. Se llevarán a cabo con la ayuda y supervisión del director técnico del proyecto.

En cuanto a las herramientas utilizadas, se ha optado deliberadamente por un software libre, de código abierto y gratuito, gracias a la licencia de la universidad, como son MATLAB y Simulink. Esto ha permitido reducir los costes de desarrollo del sistema, y simultáneamente facilita su escalabilidad. El resto del material requerido durante el proyecto ha sido abonado por la empresa INETUM, que ha marcado pautas técnicas clave y ha ofrecido un apoyo constante durante el proceso.

Como trabajo futuro se propone adaptar y ampliar las primeras etapas del procesado, con el fin de validar su funcionamiento con señales reales, y en situaciones reales. Parte de este trabajo consistiría en crear una versión hardware compatible con una FPGA.

Abstract

The objective of this project, titled “Preprocessing System for Electronic Warfare,” is to design and implement a preprocessor to make the subsequent analysis and processing of radio frequency signals more efficient. At the same time, it optimizes energy consumption and computational load savings. It is designed to be used in any field, although it focuses on military and telecommunications applications.

The proposed solution has been implemented by simulating signals with different geometries and using digital signal processing algorithms, carefully chosen after a comprehensive analysis of the most effective tools and algorithms in this field. Some of these processes include the use of digital filters such as the FIR filter, or the use of the Discrete Fourier Transform (FFT) together with sparsification. In addition, an analysis of the most prominent preprocessors in the current literature has been carried out, which has added great technical and strategic value to the project.

From a technological point of view, the system is built maintaining a modular architecture, in which each stage implemented is completely independent, structured in auxiliary functions, and following guidelines to ensure maximum compatibility with HDL, since the objective of the project is to design the system in a way that enables its future integration into an FPGA, thereby facilitating its conversion to HDL code and simplifying subsequent development work.

This project aims to follow a three-phase waterfall methodology, beginning with an analysis of the state of the art in EW and the different technologies and architectures. Secondly, the implementation of the code and the digital design of the simulator will be addressed. Finally, functional tests will be carried out and documented to validate the correct functioning of the design and compliance with the established requirements.

The testing approach focuses on performing unit tests on each microservice, which will consist of identified use cases. After these unit tests, the next step is to perform integration tests. These consist of testing how the interaction between microservices works, and given the modular architecture, these tests will check the exchange of information between components. These tests will be carried out with the help and supervision of the project's technical director.

In terms of the tools used, a deliberate choice has been made to use free, open-source software, thanks to the university's license, such as MATLAB and Simulink. This has reduced the system's development costs and, at the same time, facilitates its scalability. The rest of the material required during the project has been paid for by the company INETUM, which has set key technical guidelines and provided constant support throughout the process.

Future work will involve adapting and expanding the first stages of processing in order to test the entire system and validate its operation with real signals in real situations. Part of this work will consist of creating a hardware version compatible with an FPGA and monitoring its operation.

Índice de contenidos

Lista de Figuras

Lista de Tablas

Lista de Acrónimos

ADC: Analog to Digital Conversor, 9

COTS: Commercial Off-The-Shelf, 27

DASA: Digital Assisted Analogue technology, 19
DCA: Digital to Analog Conversor, 9
DDC: Digital Down-Conversion, 9

EA: Electronic Attack, 5
ECM: Electronic Counter Measures, 5
ELINT: Electronic Intelligence, 8
EP: Electronic Protection, 5
ESM: Electronic Support Measures, 5
EW: Electronic Warfare, 1

FDR2 -BD: Fast Data Reduction Recommendation tool for tabular big data, 17
FFT: Fast Fourier Transform, 11
FPGA: Field Programmable Gate Array, 1, 11

GWs: Gateways, 16

HDL: Hardware Description Language, 12

IA: Inteligencia Artificial, 6
IFFT: Inverse Fast Fourier Transform, 25
IoT: Internet of Things, 6
ITAR: International Traffic in Arms Regulations, 20

LZMA: Lempel–Ziv–Markov chain algorithm, 16

MDCT: Modified Discrete Cosine Transform, 18
MSE: Mean Squared Error, 26

NGF: Next Generation Fighter, 19

ODS: Objetivos de Desarrollo Sostenible, 73

PDW: Pulse Descriptor Word, 9
PRI: Pulse Repetition Interval, 8
PW: Pulse Width, 8

RC: Remote Carriers, 19
RF: Radio Frequency, 8

SNR: Signal-to-Noise Ratio, 27
SoC: System on Chip, 9

TOA: Time Of Arrival, 8

1. Introducción

Este apartado abre el proyecto. En él se expone la motivación que lo ha estimulado, junto con el proyecto real que ha servido de referencia, además de los objetivos que se persiguen y cómo se ha organizado el contenido del manuscrito.

1.1 Marco y motivación del proyecto

En un contexto marcado por avances tecnológicos constantes y una creciente dependencia de los sistemas electrónicos, la seguridad y la defensa suponen un reto cada vez más importante. Lamentablemente, hoy en día las guerras han vuelto a ocupar un lugar recurrente en el discurso social, político y tecnológico. Conflictos que parecían propios del pasado han resurgido con una crudeza inesperada, poniendo de nuevo sobre la mesa la importancia de contar con sistemas de defensa eficaces, rápidos y, sobre todo, adaptados a los retos del presente. En este escenario, la tecnología no es un elemento secundario, sino una pieza clave. Y es que los combates ya no solo se libran con armas y vehículos físicos sobre el terreno, sino también en dominios invisibles, como el espectro electromagnético.

Es aquí donde entra en juego la Guerra Electrónica (EW – Electronic Warfare). Esta disciplina se centra en el uso del espectro electromagnético para interrumpir, interceptar o manipular las comunicaciones y sistemas electrónicos del adversario, al tiempo que se protege la propia infraestructura de ataques similares. De hecho, organismos como la OTAN (Organización del Tratado del Atlántico Norte) reconocen la guerra electrónica como una capacidad crítica para las operaciones modernas [1].

En un entorno donde cada milisegundo cuenta, la capacidad de procesar señales de forma eficiente y en tiempo real, puede marcar la diferencia entre el éxito operativo y el fracaso. La cantidad de información recibida es inmensa y, muchas veces, caótica. Sin un mecanismo que reduzca y depure esa información, los sistemas de análisis quedarían saturados.

El proyecto que se presenta en esta memoria viene motivado precisamente por ese desafío. Se propone el diseño de un preprocesador para señales en un contexto de EW, con la capacidad de limpiar, y reducir datos recibidos, reduciendo la carga de información, facilitando su tratamiento posterior y agilizando la carga computacional. Todo ello, con la vista puesta en que su diseño habilite su futura implementación en una FPGA (Field Programmable Gate Array).

El desarrollo de este proyecto se llevará a cabo en el marco de la empresa INETUM España [2], siguiendo sus directrices y estándares para garantizar la calidad y el cumplimiento de los objetivos establecidos en la sección 3. Especificaciones y restricciones de diseño. La empresa INETUM que está especializada en tecnologías de la información y consultoría, ha desarrollado soluciones avanzadas en EW, inteligencia y control para las Fuerzas Armadas españolas [3]. Sin embargo, el proyecto no se enmarca dentro de ningún proyecto actual de la empresa, aunque servirá como prueba de concepto para el desarrollo de proyectos futuros.

Se ha tomado como base un proyecto concreto que servirá de punto de partida y de inspiración. En la siguiente sección (1.1.1 Proyecto real de referencia) se presentará en detalle este proyecto de referencia, que no solo ha marcado el rumbo técnico del trabajo, sino que también ha permitido identificar necesidades clave.

1.1.1 Proyecto real de referencia

En este apartado se va a explicar el proyecto real en el que se basa este trabajo, y las modificaciones que se van a hacer sobre él.

En la Figura 1, se muestra el diagrama de bloques del proyecto real que se llevaría a cabo, y que se ha usado como referencia. La señal empezaría siendo enviada desde una base de telecomunicaciones militar, que sería recibida y procesada por un avión de combate o algo similar. En este camino, la señal puede verse afectada por interferencias o ruido, lo que compromete la integridad y claridad de la comunicación, complicando el análisis y lectura de la señal.

Las interferencias pueden proceder tanto de fuentes naturales como tormentas, o de fuentes artificiales como radares enemigos. El ruido, por su parte, generalmente es una perturbación aleatoria, generado por componentes electrónicos o ruido ambiental captado durante la transmisión. Esta mezcla degrada la señal y complica al receptor, en este caso al avión militar, realizar la lectura y el análisis correctamente.

Es aquí donde interviene el preprocesador, actuando justo nada más recibir la señal, filtrando y procesando la señal procurando conseguir una señal más limpia, reducida y sencilla, facilitando el trabajo al procesador que interpretará la señal.

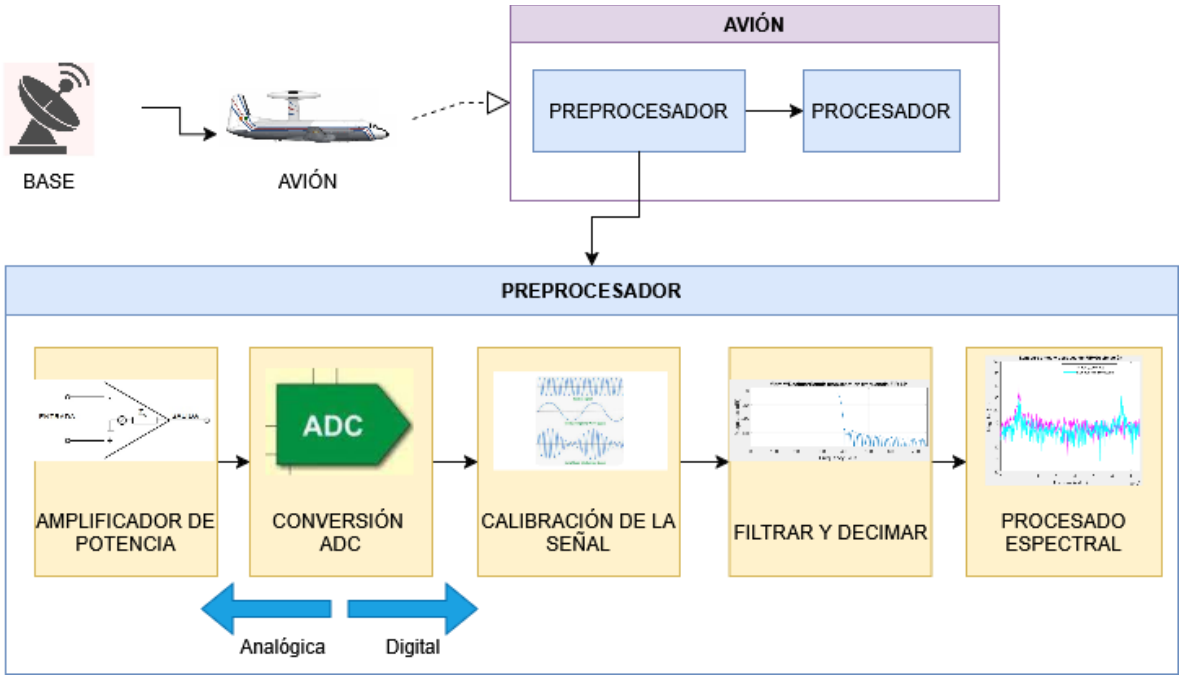


Figura 1: Diagrama de bloques general del proyecto real

La idea del proyecto sería trabajar con señales reales sin necesidad de que sean generadas. Sin embargo, como se ha explicado, las señales reales se exponen a una alta componente de variabilidad, por lo que se descarta como opción viable para su implementación en este proyecto, y en su lugar se va a trabajar con señales simuladas directamente desde el software de programación.

La decisión de trabajar con señales simuladas en este proyecto, ha permitido que se pueda prescindir de las dos primeras etapas, que en sistemas reales serían esenciales. En primer lugar, la etapa del amplificador de potencia de la señal, ya que la señal recibida suele requerir amplificación para poder aprovechar plenamente su rango dinámico y llevar a cabo la conversión analógica a digital.

La segunda etapa sería la conversión de la señal analógica a digital (ADC), que se encarga de transformar la señal analógica continua recibida, en una señal digital discreta mediante dos procesos fundamentales: muestreo y cuantización. Sin este paso, no sería posible garantizar que la señal sea manipulada eficientemente mediante técnicas digitales, como el filtrado, decimación y procesado espectral que se realizan en el proyecto.

1.2 Objetivos técnicos y académicos

Este proyecto de fin de carrera tiene los siguientes objetivos técnicos:

- Implementar un sistema de preprocesado eficiente, que reduzca información manteniendo la calidad de la señal.
- Diseñar e implementar un modelo digital del sistema de preprocesado.
- Documentar y explicar las pruebas realizadas que verifican el correcto funcionamiento del sistema de preprocesado.
- Crear un boceto de un diseño que pueda llevarse a cabo más adelante a una FPGA.

En el ámbito académico, el proyectista tiene como objetivo adquirir las siguientes competencias y habilidades:

- Desarrollar un pensamiento crítico y analítico capaz de descomponer problemas técnicos complejos en tareas más sencillas y abordables.
- Obtener una comprensión más profunda de lo que supone diseñar e implementar de forma autónoma un proyecto que puede llegar a utilizarse en situaciones reales.
- Aprender a tomar decisiones fundamentadas al elegir entre distintas metodologías.

1.3 Estructura del resto de la memoria

Este manuscrito se organiza en los siguientes capítulos:

- **Marco tecnológico**: Explica y contextualiza al lector sobre la necesidad de este proyecto, y ofrece una visión general de las tecnologías actuales en sistemas de preprocesado, y herramientas y algoritmos aplicables a este trabajo.
- **Especificaciones y restricciones de diseño:** Define los requisitos del sistema de calidad y de operatividad, además de las condiciones que debe tener un sistema de preprocesado ideal.
- **Descripción de la propuesta:** Detalla el diseño arquitectónico, los procesos o etapas que implementa el código, destacando los algoritmos empleados. Así mismo, explica y detalla el diseño del modelo digital llevado a cabo.
- **Resultados**: Presenta los resultados experimentales de cada etapa del sistema, dividido en Pruebas Unitarias, Pruebas de Integración [4], Métricas de calidad, y validación de los requisitos establecidos.
- **Presupuesto**: Desglosa los aspectos financieros del proyecto, incluidos los recursos materiales y técnicos que han sido necesarios.
- **Impacto del proyecto**: Evalúa las implicaciones que el proyecto puede tener a nivel ambiental, de seguridad, y ético-social, valorando su impacto más allá de lo puramente técnico.
- **Conclusiones**: Resume las aportaciones clave, las lecciones aprendidas gracias al desarrollo, y esboza los pasos futuros que permitirían afinar y fortalecer el proyecto.
- **Anexo I Manual de usuario**: Incluye documentación adicional para el lector, explicando detalladamente como probar y ejecutar todos los archivos implementados.

2. Marco tecnológico

La Guerra Electrónica (EW) gira alrededor de tres grandes pilares. Estos ejes trabajan juntos para mantener el dominio del espectro electromagnético [5].

1) El primero es el Ataque Electrónico (EA – Electronic Attack)) o contramedidas electrónicas (ECM – Electronic Counter Measures). Consiste en el uso de energía electromagnética, armas de energía dirigida o armas anti-radiación. Su objetivo es atacar al personal, instalaciones o sistemas de armas, con la intención de degradar, neutralizar o destruir la capacidad de combate del enemigo. Un ejemplo de EA consiste en apagar por completo todos los sistemas de uso militar.

2) Luego está la Protección Electrónica (EP – Electronic Protection). Estas, son todas aquellas acciones tomadas para proteger al personal, instalaciones o sistemas de armas frente a cualquier ataque electromagnético que perjudique en cualquier aspecto a la capacidad de combate propia. Conceptualmente se entiende como el "escudo" ante posibles EAs.

3) Por último, el Apoyo Electrónico (ES – Electronic Support) o Medidas de Apoyo Electrónico (ESM – Electronic Support Measures). Son acciones llevadas a cabo para rastrear, interceptar, y analizar fuentes de energía electromagnética radiadas intencionalmente o no, con el propósito de reconocer amenazas y revelar información crítica.

Estos tres pilares son esenciales y deben funcionar coordinados. Aunque, más allá de esta coordinación, en un mundo donde la tecnología avanza a pasos agigantados, cada vez nos enfrentamos a amenazas más sofisticadas, señales difíciles de interpretar y entornos saturados de información. Así surge la necesidad fundamental y urgente de contar con sistemas rápidos, eficaces y avanzados de preprocesado señales.

En lo restante a este capítulo se expone la necesidad social y tecnológica que enmarca el sistema propuesto en este proyecto (sección 2.1). Un estudio comprensivo entre otros sistemas de preprocesado del mercado (sección 2.2). Un análisis comprensivo de las herramientas más representativas del mercado actual para el desarrollo del sistema (sección 2.3). Por último, un análisis exhaustivo de los posibles algoritmos que pueden satisfacer las necesidades del proyecto (sección 2.4).

2.1 Necesidad Social y Tecnológica de un Sistema de Preprocesado de EW

Como se ha comentado, el desarrollo acelerado de las tecnologías de comunicación y el creciente protagonismo del entorno electromagnético en las operaciones militares han situado a la EW como una de las capacidades estratégicas más relevantes en los conflictos modernos. La reciente invasión rusa de Ucrania con el lanzamiento de drones y la importancia de las comunicaciones a través de radiofrecuencia, han puesto de manifiesto cómo el control

del espectro electromagnético puede condicionar el éxito de las operaciones sobre el terreno [6]. La EW no solo permite neutralizar las comunicaciones y sistemas de navegación del adversario, sino que se ha consolidado como un factor clave para asegurar la superioridad informativa y operativa en los conflictos modernos. En este nuevo escenario, la digitalización de los sistemas militares, la proliferación de sensores y plataformas no tripuladas, así como la necesidad de transmitir grandes volúmenes de información en tiempo real, han provocado una expansión considerable de las bandas de frecuencia utilizadas en todo tipo de operaciones.

Este fenómeno no solo se limita al ámbito militar, sino que también afecta a sectores civiles [7], donde el crecimiento de las redes de telecomunicaciones, la irrupción de tecnologías como el 5G, o la expansión del internet de las cosas (IoT – Internet of Things) que hace referencia a la interconexión de objetos cotidianos a través de internet permitiendo que estos dispositivos recojan y transmitan datos, han saturado aún más un entorno ya de por sí congestionado. Este escenario expone la necesidad de garantizar la ciberseguridad y asegurar la disponibilidad de los servicios esenciales, lo que ha convertido a la monitorización y el preprocesado de señales, en una necesidad transversal que afecta tanto al ámbito de la defensa, como al tejido socioeconómico.

En este contexto, el preprocesador se convierte en un elemento esencial, ya que permite filtrar, segmentar y priorizar las señales antes del procesamiento principal. De esta manera se consigue optimizar los recursos del sistema y reducir la carga computacional, especialmente en los sistemas de EW de banda ultra ancha (UWB - Ultra Wide Band), que es vital para poder gestionar grandes volúmenes de datos de señales sin procesar [8].

Sin embargo, esta necesidad trae consigo una consecuencia directa: la generación de cantidades ingentes de información que deben ser gestionadas de manera eficiente respetando el tiempo útil, esto es, el periodo máximo admisible para procesar una señal, sin que la información obtenida pierda su valor práctico. Esta situación plantea uno de los mayores desafíos en el ámbito de la EW moderna, evitar la saturación de los recursos disponibles. Así, el objetivo principal de un sistema de preprocesado es discriminar de forma temprana la información relevante, de aquella que carece de interés o que puede ser considerada ruido, reduciendo así la carga computacional de las siguientes etapas del sistema, y optimizando los recursos energéticos disponibles [9].

Además, la EW moderna también debe enfrentarse a nuevas formas de agresión que combinan: técnicas tradicionales con ciberataques, y el uso de IA (Inteligencia Artificial) para la generación de señales falsas o la saturación deliberada del entorno electromagnético. Estas amenazas emergentes, obligan a los sistemas de preprocesado a evolucionar, integrando capacidades de análisis más sofisticadas y eficientes, que permitan detectar patrones, correlacionar eventos y anticiparse a las acciones del adversario en un entorno de alta incertidumbre.

Es por todo esto que se plantea como uno de los objetivos unificar el proceso de monitorización, desde el muestreo de información, pasando por el procesamiento y análisis de los datos hasta la visualización final y resumida sobre los parámetros relevantes de la señal.

Volviendo al contexto de la evolución del ancho de banda en las comunicaciones, se propone la comparación entre radios tácticas que ejemplifica de manera precisa el avance tecnológico que ha permitido ampliar las capacidades de transmisión, con repercusiones tanto en el ámbito social como en el tecnológico. Por ejemplo, la radio PR4G V3 del año 2007 que se representa en la Figura 2-a, operativa en la banda VHF (Very High Frequency), está diseñada principalmente para comunicaciones en banda estrecha, operando en canales de 25 kHz y ofreciendo tasas de transmisión de entre 38 y 64 Kbps en algunos casos, lo que restringe la posibilidad de transmitir simultáneamente voz y datos reflejando las limitaciones propias de generaciones anteriores [10] [11].

En contraposición, la radio Harris Falcon III RF-7850S SPR del año 2018 que usan actualmente las fuerzas armadas españolas, representada en la Figura 2-b, refleja la siguiente etapa evolutiva en términos de ancho de banda y versatilidad operativa. Con un rango de ancho de banda que oscila desde los 25 KHz hasta 5 MHz en modos de banda ancha, esta radio posibilita una configuración adaptable que favorece la transmisión simultánea de voz y datos a velocidades significativamente mayores, alcanzando hasta 1.5 Mbps [12] [13].

a. Radio PR4G V3

b. Radio Harris Falcon III

Figura 2: Comparación radios tácticas

Este aumento en el ancho de banda no solo optimiza la eficiencia de las comunicaciones militares, sino que también ilustra cómo el incremento en la capacidad de transmisión ha impulsado una transformación en los ámbitos sociales y tecnológicos. La capacidad para gestionar datos y voz en un mismo canal, sin pérdida de integridad o seguridad, se traduce en una mayor interoperabilidad entre sistemas y una robustez que repercute en la calidad de la comunicación en situaciones críticas.

2.2 Estudio del mercado de sistemas de preprocesado

En esta sección, se va a realizar un análisis entre los distintos sistemas de preprocesado del mercado actual. El estudio se va a centrar en buscar y analizar aquellos que estén pensados o realicen una tarea similar al que se va a implementar.

A grandes rasgos, los dos preprocesadores elegidos parten de la misma premisa, reducir drásticamente la cantidad de datos de radio frecuencia, sin perder la información clave que hace falta para identificar amenazas, y respetando el tiempo útil. Sin embargo, lo hacen con filosofías distintas. A continuación, se explica cómo lo hace cada uno y también sus características generales.

2.2.1 R&S WPU2000 – Wideband Processing Unit

Este preprocesador fue desarrollado por Rohde & Schwarz (Alemania), especialista en guerra electrónica desde los años 60, y fue presentado en 2020 como un núcleo de receptores de Inteligencia Electrónica (ELINT - Electronic Intelligence) [14] navales y aéreos. Se muestra en la Figura 3.

Figura 3: Sistema de preprocesado R&S WPU2000

Su funcionamiento combina: (I) Un sintonizador superheterodino de banda ancha; (II) un digitalizador de 12 bits; (III) ocho canales de conversión digital descendente (DDC) en paralelo, que reducen el ancho de banda hasta 2 GHz, y generan potentes FFT (Transformada Rápida de Fourier) de espectro en tiempo real; (IV) y analizadores de impulsos integrados.

El firmware extrae en tiempo real descriptores de impulsos a velocidades superiores de 10 millones de impulsos por segundo. Al transmitir solo esos registros se ahorra ancho de banda y almacenamiento, lo que permite a los operadores ignorar el espectro irrelevante y centrarse sólo en los emisores de amenaza [15] [16]. Algunos datos característicos que extrae son:

- Tiempo de llegada (TOA – Time Of Arrival): Marca temporal del instante en que comienza el pulso, medida normalmente en nanosegundos (ns). Permite ordenar los pulsos y calcular intervalos entre ellos.
- Anchura del pulso (PW – Pulse Width): Duración del pulso entre su flanco de subida y de bajada, habitualmente al 50% de amplitud.
- Intervalo de repetición (PRI - Pulse Repetition Interval): Tiempo que transcurre entre el inicio de dos pulsos consecutivos del mismo emisor.

- Frecuencia portadora (RF – Radio Frequency): Frecuencia central instantánea del pulso, medida en hertzios (Hz).

El WPU2000 tiene cobertura de frecuencias y ancho de banda muy amplios, generación directa de PDW (Pulse Descriptor Word) haciendo uso de bibliotecas externas privadas, y usa un chasis completo suponiendo un consumo muy elevado, y menor comodidad a la hora de integrarlo en un sistema más compacto.

2.2.2 Pentek/Mercury Quartz 5950 – RFSoC Processor

Es el primer modelo de la serie Quartz, lanzado en 2018. Integra un conjunto de circuitos integrados (SoC - System on Chip) llamado Xilinx Zynq UltraScale RFSoC. Este formato implica que las muestras entran directamente en la lógica programable, lo que acelera el procesamiento y simplifica el diseño de la placa. Se muestra en la Figura 4.

Figura 4: Sistema de preprocesado Pentek/Mercury Quartz 5950

Está formado por ocho convertidores Analógico-Digital (ADC – Analog to Digital Conversor) de 4 GSPS (Giga Samples Per Second) y 12 bits, y ocho convertidores Digital-Analógico (DCA – Digital to Analog Conversor) de 14 bits y 6 GSPS, que comparten la estructura FPGA [17] [18].

Cada ADC incluye: (I) Una cadena de conversión descendente (DDC - Digital Down-Conversion) integrada, que es un bloque de procesamiento digital que toma una señal muestreada en y la transforma a banda base, reduciendo al mismo tiempo la tasa de muestreo; (II) y un decimador FPGA adicional, que añade un bloque CIC + FIR – filtros digitales – logrando reducciones de datos de hasta 128 veces menor que antes de sacar los datos.

Además, la IP preinstalada permite capturar señales transitorias, almacenar datos de RF en memoria y generar chirps – señal cuya frecuencia cambia de forma continua a lo largo de la duración – de modo que muchas pruebas de EW y de radar, se pueden realizar directamente sin escribir nada de HDL.

Todo esto ofrece una arquitectura abierta integrable en COTS (Commercial Off-The-Shelf) [19]; una reducción masiva del volumen de muestras; tamaño y peso reducidos; y una FPGA programable por el usuario para crear filtros personalizados.

Las ventajas que aporta este preprocesador son que integra todo en un chip, ya que el RFSoC integra tanto los módulos ADC/DAC como la FPGA en la misma oblea; que no hace falta codificar una IP propia; y que ofrece sincronización multitarjeta para arquitecturas distribuidas, como colocar varias tarjetas en distintos sistemas.

2.2.3 Conclusiones del análisis de los preprocesadores del mercado.

Tras el análisis realizado de los dos preprocesadores, que se consideran más parecidos tanto en el contexto de enfoque funcional, como en el tipo de aplicación, en este caso EW. Ambos comparten una lógica centrada en la reducción y depuración eficiente de señales de radio frecuencia.

Comparando el primero, "*R&S WPU2000* ", con la propuesta desarrollada por el autor de este proyecto, ambos buscan la misma función de reducir la cantidad de información, pero sin la necesidad de catalogar aspectos de la señal, y con mayor modularidad para adaptarse a proyectos gracias a que su diseño habilita la futura integración en una FPGA, lo que supone menor tamaño y consumo.

Y, con el segundo *"Pentek/Mercury Quartz 5950"*, la diferencia es que el diseño implementado durante este proyecto, permite separar la lógica de procesado del soporte mecánico. Esta característica facilita que se pueda acoplar a digitalizadores externos, y que se pueda adaptar a presupuestos más ajustados.

Gracias al análisis, se puede sacar en conclusión que el sistema de preprocesado que se va a implementar y explicar en este manuscrito, va a ser más compacto para facilitar su integración, priorizando un bajo consumo y coste de producción. Además de, ofrecer ideas reveladoras como la implementación de la decimación, filtros digitales y facilitar la escalabilidad con una FPGA.

2.3 Estudio del mercado de herramientas

En este apartado se pretende hacer una comparativa de las distintas herramientas que se han estudiado y analizado para llevar a cabo la implementación del proyecto. Para ello, se va a tener en cuenta principalmente que la herramienta ofrezca las siguientes condiciones.

1) Rapidez: Una de las más importantes características de un preprocesador es que sea rápido, ya que, si tarda lo que tardaría un procesador corriente, su diseño no sería productivo.
2) Sencillez: Dado el tiempo del que se dispone para realizar el TFG, lo ideal es una herramienta con una interfaz sencilla e intuitiva con la que el alumno no pierda mucho tiempo en familiarizarse.
3) Compatibilidad con RF: La herramienta debe ser capaz de recibir y trabajar con señales de RF, dado que son con las que se va a trabajar.
4) Compatibilidad con un FPGA: El propósito final del preprocesador es habilitar su posterior integración en una FPGA, por lo que, si la herramienta no ofrece unas condiciones de escalabilidad adecuadas para ello, no será conveniente.

2.3.1 Apache Spark y Apache Flink

En primer lugar, una de las primeras opciones dado su conocido prestigio ha sido **Apache Spark**. Su potencia para procesar grandes volúmenes de información en paralelo es indiscutible, y eso la convierte en una herramienta muy atractiva. Pero la realidad es que, al meterse a fondo, su complejidad se hace evidente: requiere una curva de aprendizaje considerable, y su coste computacional puede dispararse con facilidad. Para un proyecto como este, centrado en señales de RF y con un enfoque hacia la eficiencia, es una herramienta muy compleja y con un gran coste computacional.

Además, también se ha estudiado la herramienta parecida **Apache Flink**. Su arquitectura orientada a flujos continuos y su baja latencia son impresionantes. Sin embargo, esa precisión viene acompañada de una rigidez que exige una infraestructura robusta y, sobre todo, experiencia técnica muy específica.

Por otro lado, ninguna de las dos herramientas proporciona bibliotecas nativas para el procesamiento digital de señales, como la transformada rápida de Fourier (FFT - Fast Fourier Transform), que son comunes en RF. Además, no están pensadas para correr en FPGAs, ni mucho menos optimizadas para ello [20].

Por último, otro aspecto que se ha tenido en cuenta es el entorno en el que trabajan. Tanto Apache Spark como Flink están profundamente integrados en ecosistemas como Java o Scala, lo que implica adoptar una arquitectura de desarrollo compleja, y que a la hora de enfrentarse a problemas donde se necesita manipular datos, gestionar transformaciones específicas o controlar el flujo de ejecución, imponen una estructura demasiado estricta.

2.3.2 Octave y Matlab

En segundo lugar, se ha optado por analizar las herramientas **Octave** y **Matlab**, dos entornos de programación que se centran en el análisis numérico. *Matlab* es una herramienta de pago desarrollada por *Mathworks* ampliamente conocida, caracterizada por su robustez, facilidad de uso, compatibilidad y versatilidad. Pero sobre todo, lo que destaca de esta herramienta es su gran comunidad y soporte técnico, que ofrece una explicación de todas sus funcionalidades con descripciones detalladas y ejemplos [21].

Valorando las condiciones comentadas anteriormente, Matlab es una herramienta rápida, con gran cantidad de toolboxes especializados en el procesamiento avanzado de señales, y con herramientas de diseño para FPGA como HDL Coder (Hardware Description Language) [22]. Además, el autor de este proyecto posee un amplio y previo conocimiento sobre su funcionamiento, ya que se ha usado en varias asignaturas del grado cursado, y la universidad facilita una licencia de uso gratuito.

Por otro lado, la herramienta *Octave* es gratuita y es una alternativa de código abierto que busca replicar muchas de las funciones de Matlab*,* pero valorando las mismas condiciones, tiene funciones similares, pero menos optimizadas y completas, carece de la variedad de funcionalidades mencionadas sobre el procesamiento de señales, y la generación de código HDL necesario para integrar el preprocesador en una FPGA. Además, al ser de código abierto depende de la comunidad para la resolución de problemas, lo que puede no ser ideal para usuarios que necesitan respuestas rápidas [23].

2.3.3 Conclusión del análisis comparativo de herramientas para la implementación

Por todo lo comentado, finalmente se ha optado por elegir *Matlab* como herramienta de programación para el desarrollo y la implementación del proyecto. Se muestra en la Tabla 1 una tabla comparativa a modo de resumen.

Tabla 1: Comparación de herramientas analizadas para implementar el código

Herramienta	*Rapidez*	*Sencillez*	*Compatibilidad RF*	*Compatibilidad FPGA*
Apache Spark	Sí	No	No	No
Apache Flink	Sí	No	No	No
Octave	Sí	Sí	Sí (básica)	No
Matlab	Sí	Sí	Sí	Sí

2.3.4 LabVIEW y Simulink

Al considerar el diseño del modelo digital del preprocesador de EW, destacan principalmente dos herramientas: **LabVIEW** [24] y **Simulink** [25].

En primer lugar, LabVIEW, desarrollado por *National Instruments*, es conocido por su enfoque gráfico intuitivo. Utiliza diagramas de bloques interconectados, lo que facilita enormemente el trabajo a la hora de hacer una representación visual directa, con una interfaz gráfica sencilla.

Por otro lado, Simulink, que es parte del entorno Matlab, está diseñado específicamente para modelar, simular y analizar sistemas dinámicos complejos estructurado en bloques. Su integración directa con Matlab proporciona una flexibilidad increíble, especialmente útil para la implementación de algoritmos complejos. Aunque su curva de aprendizaje pueda ser algo más pronunciada, Simulink ofrece un nivel de detalle y control que LabVIEW difícilmente puede igualar en aplicaciones de procesamiento de señales. Además, Simulink tiene el mismo soporte técnico mencionado anteriormente que Matlab, que para su uso es de gran ayuda.

En términos de compatibilidad con radiofrecuencia, ambas herramientas tienen capacidades importantes, aunque Simulink ofrece un entorno más especializado para simulaciones detalladas y análisis en profundidad.

En cuanto a la compatibilidad con FPGA, LabVIEW cuenta con herramientas como LabVIEW FPGA, que permiten una implementación relativamente sencilla de algoritmos en hardware específico. Simulink, por su parte, ofrece herramientas o bloques de trabajo especializados en HDL, que brindan mayor precisión y control detallado en el diseño de sistemas optimizados para FPGA.

2.3.5 Conclusión del análisis comparativo de herramientas para el diseño digital

Teniendo en cuenta los cuatro aspectos clave enumerados al principio, las dos herramientas son muy buenas opciones. Pero, aprovechando que el código se va a implementar en Matlab que tiene integración directa con Simulink, y que como se ha comentado antes, la universidad ofrece una licencia gratuita para su uso, se ha decidido que se va a realizar el diseño del modelo digital en Simulink.

Cabe destacar que LabVIEW ofrece una versión gratuita pero muy limitada en comparación con la licencia de Simulink. En la Tabla 2 se muestra el resumen del análisis comparativo.

Tabla 2: Comparación de herramientas analizadas para el diseño del modelo digital

Herramienta	*Rapidez*	*Sencillez*	*Compatibilidad RF y FPGA*	*Licencia gratuita*
LabVIEW	Sí	Sí	Sí	No
Simulink	Sí	Sí (menos)	Sí	Sí

2.4 Estudio de algoritmos y técnicas de reducción y compresión de datos

En esta sección se van a analizar posibles soluciones para afrontar el problema de la sobreinformación. Para ello se van a analizar distintos algoritmos que sean eficientes en la reducción de datos.

2.4.1 Huffman Adaptativo

En primer lugar, un estudio de la Universidad Espíritu Santo llamado: "Optimización del uso de ancho de banda en los enlaces de transmisión de datos por medio de algoritmos de compresión" [26], defiende que una buena solución es comprimir el ancho de banda. Según dicho estudio, la razón más importante para comprimir información es que el ancho de banda es un recurso limitado, lo cual influye en la cantidad de bits que se pueden transmitir por un medio y en un tiempo determinado.

Tras realizar pruebas con los tres algoritmos (RLE, LZW y Huffman Adaptativo), mediante el software *Matlab* como herramienta de simulación, el estudio afirma que el algoritmo que demostró ser más eficiente ante todas las circunstancias, además de ser el más estable en sus pruebas fue **Huffman Adaptativo** (promedio de ahorro de ancho de banda 25,99% según el estudio).

La principal limitación de este trabajo recae en que las pruebas se realizaron únicamente simulando condiciones ideales de transmisión. Por lo tanto, los resultados que se obtuvieron no toman en cuenta factores externos que puedan alterar el desarrollo de las pruebas tales como interferencia.

Con el fin de valorar su rendimiento, se ha implementado un prototipo básico basado en codificación Huffman. Sin embargo, tras realizar varias pruebas con señales cargadas de altas frecuencias y ruido, con mucha variación entre valores simulando lo que sería una señal real, se ha demostrado su limitación en condiciones no ideales. La Figura 5 que representa el tiempo de compresión frente a la duración de la señal en tres distintas frecuencias, demuestra que no es un algoritmo eficiente a la hora de comprimir señales.

En la Tabla 3 que recoge los datos exportados de la Figura 5, se calcula que para una señal de f = 5MHz y de 0.1 s de duración, su tiempo de compresión es de 9.334.10 ms, es decir el tiempo de compresión es aproximadamente 93 veces el tiempo de duración de la señal. Extrapolándolo a la vida real, no es viable para un preprocesador de guerra electrónica, ya que como se ha comentado en la sección 2.3 Estudio del mercado de herramientas, es imprescindible que sea rápido, del orden de los microsegundos. Lo ideal sería 100 µs para que pueda ser considerado rápido [27].

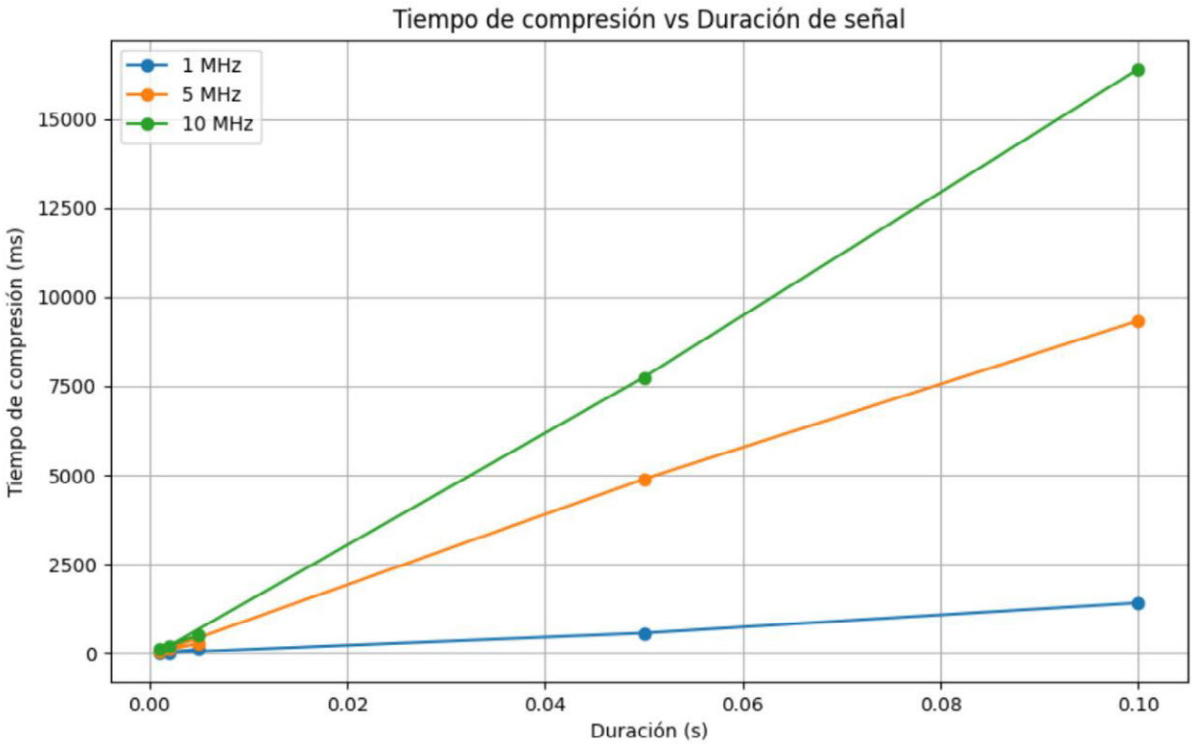


Figura 5: Algoritmo de Huffman - Tiempo de compresión vs Duración de la señal

Tabla 3: Algoritmo Huffman - Datos de tiempo de compresión

Fs (MHz)	Duración (s)	Tiempo (ms)
1.0	0.005	107.79
1.0	0.001	12.26
1.0	0.002	16.76
1.0	0.050	564.10
1.0	0.100	1408.06
5.0	0.005	276.31
5.0	0.001	44.46
5.0	0.002	137.16
5.0	0.050	4886.19
5.0	0.100	9334.10
10.0	0.005	516.60
10.0	0.001	102.21
10.0	0.002	206.19
10.0	0.050	7748.17
10.0	0.100	16404.09

2.4.2 Algoritmo LZMA

Otra opción es la compresión de datos mediante el **algoritmo LZMA** (Lempel–Ziv–Markov chain algorithm), reconocido por su alta tasa de compresión. Este algoritmo destaca por combinar técnicas de búsqueda en diccionarios con modelos estadísticos, lo que permite detectar patrones redundantes en los datos de manera muy eficiente, para después comprimirlos.

Además, se ha decidido analizar LZMA ya que cuenta con implementaciones optimizadas para aceleradores hardware como FPGA, tal como se describe en el artículo científico "Efficient sequencing data compression and FPGA acceleration based on a two-step framework" [28]. En dicho artículo, se presenta una arquitectura que alcanza velocidades de compresión superiores a 100 MB/s con notables resultados de compresión, lo que lo convierte en una referencia en contextos donde el espacio de almacenamiento es una prioridad.

Sin embargo, tras evaluar su aplicabilidad a nuestro contexto, se ha decidido descartarlo. La razón principal es la complejidad del LZMA debido a: (i) es un algoritmo que requiere un uso intensivo de memoria y lógica de búsqueda variable lo que aumenta el riesgo de errores; (ii) la lógica que sigue introduce varios ciclos de procesamiento por byte, lo que penaliza la velocidad de compresión en tiempo útil, y (iii) su portabilidad en Matlab sería demasiado compleja, debido a que sería necesario integrar ejecutables o librerías de terceros (Python). Esto sumado a las demás características comentadas, su implementación en HDL para su posterior integración en una FPGA se complica. Para el futuro, se van a priorizar soluciones sencillas pero efectivas y trasladables a FPGAs.

2.4.3 Predicciones Q-Learning

Otra opción que considerar, es la reducción de datos basada en **predicciones Q-Learning** para ajustar los intervalos de muestreo, según el artículo científico "Estrategias basadas en predicciones para reducir las transmisiones de datos en el loT" [29]. El artículo propone una técnica innovadora de reducción de datos en WSN (Wireless Sensor Networks ó Redes de Sensores Inalámbricos) basada en la predicción de las mediciones, lo que permite disminuir significativamente las transmisiones de datos.

La idea central es que, dado que los nodos sensores no pueden procesar grandes volúmenes de información o transmitir a altas velocidades, se aprovechan las capacidades superiores de los GWs (Gateways), que no tienen restricciones críticas de cómputo ni comunicación. Estos GW pueden ejecutar algoritmos avanzados para predecir los datos que se medirán en el futuro. Además, se implementa un mecanismo basado en Reinforcement Learning, específicamente Q-Learning, para ajustar dinámicamente los intervalos de muestreo de los nodos en función de los cambios detectados en el entorno. Así, solo se transmiten aquellos datos que realmente aportan nueva información, evitando enviar mediciones redundantes.

Tras estudiarlo y analizarlo, este modelo de predicciones requiere del uso de elevados recursos y expone una alta componente de variabilidad, por lo que se descarta como opción viable para su implementación en este proyecto.

2.4.4 Fast Data Reduction Recommendation tool for tabular big data

También se ha investigado un enfoque metodológico de condensación de datos, para la reducción de grandes conjuntos de datos tabulares en problemas de clasificación, denominado **FDR2 -BD** (Fast Data Reduction Recommendation tool for tabular big data) [30]. Los objetivos de esta propuesta son los siguientes:

1) Determinar si un conjunto de datos es reducible o no mediante un submuestreo estratificado uniforme.
2) Estimar el porcentaje máximo de reducción posible y las características más importantes.

Su robustez se basa en un proceso de hiperparametrización, en el que se tienen en cuenta todos los datos siguiendo un procedimiento k-fold, que es una técnica estadística, utilizada en el aprendizaje automático, que consiste en dividir el conjunto de datos en K subconjuntos diferentes. Después, estos subconjuntos se evalúan K veces, obteniendo coeficientes de rendimiento de manera individual en cada iteración. Su principal ventaja es que es rápido y escalable mediante el uso de operaciones paralelas totalmente optimizadas proporcionadas por Apache Spark.

Desgraciadamente, este algoritmo está pensado y diseñado específicamente para trabajar con Apache Spark, y aunque ofrece un alto rendimiento en el procesamiento de grandes volúmenes de datos, su curva de aprendizaje y complejidad de implementación comprometería la viabilidad dentro de los plazos establecidos del TFG, por lo que se va a prescindir de dicha herramienta y por consiguiente del algoritmo FDR2 -BD.

2.4.5 Transformada Rápida de Fourier

Por último, se ha analizado la combinación entre **la FFT (Transformada Rápida de Fourier) junto con la esparsificación** [31]. El proceso consiste en, a través de la FFT pasar la señal al dominio de la frecuencia. De este modo convertimos una señal densa y ruidosa a una representación espectral donde aparecen los picos informativos más relevantes de la señal, llamados armónicos. Después, gracias a la esparsificación lo que se consigue es identificar y almacenar los k coeficientes con mayor energía, y descartar el resto que normalmente son ruido o componentes irrelevantes. El resultado no es solo una reducción de memoria y compresión de la información, también una clara estructura que facilita su posterior filtrado y almacenamiento. Es un proceso muy eficiente además de rápido. Debido a que la FFT ya es rápida, si además la señal esparce su energía implica menor acceso a RAM y por lo tanto más rapidez.

Por otro lado, tiene plena compatibilidad con Matlab y Simulink. Matlab incorpora funciones propias que realizan la FFT que facilita mucho la posterior implementación de la

esparsificación. Ese mismo script puede exportarse mediante Matlab a HDL hasta bloques en Simulink listos para la síntesis FPGA.

2.4.6 Conclusión del análisis comparativo de algoritmos

Por todo lo comentado, finalmente se ha decidido elegir la FFT con esparsificación como el algoritmo a implementar en este proyecto. Un buen ejemplo que ilustra muy bien esta idea es el MP3 [32]. Durante la codificación, el audio se somete a una FFT (y posteriormente a un MDCT (Modified Discrete Cosine Transform ó Transformada de Coseno Discreta Modificada)) para observar su espectro, y un modelo psicoacústico decide qué picos van a ser audibles y cuáles quedarán tapados por el fenómeno del enmascaramiento. Después, los relevantes se cuantizan y se comprimen con Huffman para que ocupen lo menos posible. Al final, es una filosofía muy parecida a la elegida, el análisis realizado de todos los distintos algoritmos ha buscado siempre el compromiso entre mantener la calidad de la señal recuperada, y la cantidad de información comprimida.

En la Tabla 4 se ha realizado una comparación de los 4 aspectos clave a la hora de elegir el algoritmo: rapidez, eficiencia y compatibilidad con Matlab y FPGA.

Tabla 4: Tabla comparativa entre algoritmos

Algoritmo	*Rapidez computacional*	*Eficiencia en reducción de datos*	*Compatibilidad Matlab*	*Compatibilidad FPGA*
Huffman Adaptativo	Sí	No	Sí (básica)	Sí
LZMA	No	Sí	No	No
Q-Learning	No	Sí	Sí	No
FDR2 -BD	Sí	Sí	No	No
FFT + esparsificación	Sí	¨Sí	Sí	Sí

3. Especificaciones y restricciones de diseño

En este capítulo, se van a numerar y comentar los requisitos que debe tener un preprocesador de EW completo, es decir, un preprocesador ideal.

3.1 Condiciones y aspectos de un preprocesador ideal

A continuación, se van a numerar los aspectos que debe tener un preprocesador ideal, es decir, todas las funciones o etapas que idealmente debe satisfacer un preprocesador completo.

1) Capacidad de procesamiento de muestras brutas detectadas en tiempo útil. Esta etapa combina la tecnología analógica asistida digitalmente DASA (Digital Assisted Analogue technology) para mejorar la precisión de la señal analógica, técnicas de reducción de datos para racionalizar los datos filtrando y comprimiendo sólo la información relevante y la eliminación de ruido, así como interfaces de alta velocidad y gran ancho de banda.
2) Procesamiento de todas las muestras respetando el tiempo útil y generación de información que resuma e integre todas las señales detectadas, además de algoritmos de detección rápida de alarmas para identificar y señalar rápidamente posibles amenazas en real.

Estas dos funciones juntas, crean un marco eficaz que transforma los datos brutos en información procesable, dotando al sistema de la capacidad de gestionar entornos de señal complejos y de alta velocidad y de sentar las bases para una respuesta eficaz a las amenazas en fases posteriores.

3) Sincronización de todos los datos mediante una referencia de reloj externa. Para correlacionar y fusionar señales provenientes de múltiples fuentes o sensores, es imprescindible una sincronización precisa. Esto se traduce en la implementación de mecanismos de sincronización de tiempo (como el uso de relojes de alta precisión y protocolos de sincronización) y en la gestión de buffers de datos que permitan alinear y combinar flujos de información sin pérdidas ni desincronizaciones.
4) Almacenamiento masivo de muestras brutas o pulsos seleccionados. Esta característica garantiza la conservación de la información crítica de las señales, y así se consigue un posterior análisis mucho más preciso.
5) Otros buses de comunicación internos (entre módulos) y externos. Una comunicación fluida entre los módulos internos y sistemas externos es clave para asegurar que los datos fluyan con una latencia moderada.
6) Escalabilidad de la solución de preprocesador para poder ser utilizada tanto en Caza de Próxima generación NGF (Next Generation Fighter) como en Operadores Remotos RC (Remote Carriers).
7) Priorizar la tecnología europea en el estado del arte, asegurando siempre el cumplimiento del Reglamento sobre el tráfico internacional de armas ITAR

(International Traffic in Arms Regulations) [33], que regula las importaciones y exportaciones de artículos y servicios relacionados con el espacio y la defensa.

8) Abordar el diseño teniendo en cuenta las restricciones típicas de los sistemas aéreos: peso, tamaño, potencia, interfaces, condiciones ambientales...

3.2 Catálogo de requisitos del proyecto

El objetivo de este proyecto es realizar un sistema de preprocesado acotado, pero eficiente. Para ello, exprimiendo las condiciones de un preprocesador ideal explicadas, se han extraído en la Tabla 5 los requisitos que se van a desarrollar y que el preprocesador debe cumplir.

Tabla 5: Catálogo de requisitos del proyecto

Requisito	Descripción	Identificador
Reducción de datos	Debe eliminar información redundante y ruido atenuando la amplitud, pero sin comprometer la calidad de la señal	R.1
Mantener la geometría de la señal original	Debe preservar la forma de la señal (picos, transiciones, etc) asegurando que el contenido útil no se distorsione.	R.2
Mantener la frecuencia de la señal original	Los componentes espectrales clave se conservan, lo que garantiza que tras el filtrado y compresión las frecuencias no se alteran	R.3
Reconstrucción sin imperfecciones	El sistema debe recuperar una señal clara y coherente	R.4
Escalabilidad a FPGA	Debe ser compatible la posterior implementación en una FPGA	R.5

Durante el proyecto se expondrán las condiciones que va manteniendo la señal y, al final en la sección 5.3 Cumplimiento de requisitos se mostrará si los requisitos se cumplen.

4. Descripción de la solución propuesta

Después del cuidadoso análisis realizado, el proceso final a seguir se compone de 9 etapas de procesado, agrupadas en 4 fases diferenciadas. El código se ha estructurado en funciones auxiliares, cada una implementa una fase, respetando una arquitectura modular [34]. Esto es, ninguna función depende de la anterior, cada una lleva a cabo su función de forma totalmente independiente. La señal se va actualizando y procesando en cada función, a excepción de la primera etapa "Generar señal" en la que se simula una señal a través de las funciones de Matlab, que puede ser senoidal, triangular o cuadrada.

Este enfoque, además de facilitar la implementación, engloba el desarrollo de un gemelo digital [35]. Es decir, una réplica virtual que reproduce cómo funcionaría en pequeña escala, un preprocesador ante distintas señales de entrada.

En lo restante a este capítulo, se va a explicar todo lo referente a la propuesta elegida. En primer lugar, las variables declaradas y el formato elegido (Sección 4.1). Después, se explica una a una cada función del proceso (Secciones 4.2 - 4.6). Por último, se expondrá el diseño del modelo digital implementado, y algunas gráficas representativas.

4.1 Acceso y tipo de las variables

Para que sea más sencillo acceder a las variables generales del código, ya que al estar estructurado en funciones todo parámetro que se quiera usar debe ser llamado en la cabecera de la función, se crea al principio del código una estructura de datos llamada "config". Esto permite agrupar datos de diferentes tipos bajo un mismo nombre, utilizando campos con nombres específicos para acceder a esos datos. Así llamando a "config" en cada función, se tiene acceso a todo ese "contenedor" de datos. Además, también facilita mucho el trabajo para el futuro script de pruebas que se implementará.

Otra decisión importante consiste en el tipo de variables que se va a usar. Como ya se ha mencionado, el objetivo final del proyecto consiste en diseñar el sistema de forma que su posterior integración en una FPGA resulte viable y lo más sencilla posible, sentando así las bases para futuros trabajos de implementación en hardware, y para ello al código se le debe aplicar la síntesis HDL. Por esto, se ha buscado el formato más compatible con HDL, sin complicar en exceso el flujo y la estructura del código. El formato elegido ha sido Q1.15 [36], también llamado punto fijo (fi), para garantizar el requisito **R.5** del catálogo de requisitos establecido en la Tabla 5 en el apartado 3.2 Catálogo de requisitos del proyecto.

Su funcionamiento consiste en escalar cada muestra por 2^{15} y almacenarla como un entero de 16 bits, con 1 bit de signo y 15 de fracción, consiguiendo un rango de ±1 sin recurrir al costoso hardware del formato del punto flotante. Esto en la práctica, se traduce en operaciones aritméticas puras, es decir, se convierten las operaciones en sumas y multiplicaciones, beneficiando mucho al FPGA ya que reduce mucho el consumo energético, y se evitan complicaciones de compatibilidad de formatos.

Además, la lógica de saturación incorporada en el formato previene envolvimientos inesperados ("wrap-arounds"). Esto describe cómo se comporta ese overflow, cuando el resultado de una operación supera el extremo superior o inferior del rango representable. En lugar de pararse en el máximo o el mínimo, el contador vuelve a arrancar desde el otro extremo, como un odómetro que tras 9999 pasa a 0000. Esto lo evitamos forzándolo a clavarse en +1 o –1, garantizando que nunca se escapen muestras erráticas y manteniendo la integridad de la señal. Esto se define en el tipo "*fimath*" que se muestra a continuación, además de cómo se define el fi.

```
1 bitsFrac = 15; bitsTot = 16;
2 Tipo_puntoFijo = numerictype(1, bitsTot, bitsFrac);
3 Forma_puntoFijo = fimath('RoundingMethod','Nearest','OverflowAction','Saturate');
4 config.Tipo = Tipo_puntoFijo; config.Forma = Forma_puntoFijo;
5 fi("señal que se quiera guardar", config.Tipo, config.Forma);
```

A continuación, se explica de manera detallada en que consiste cada etapa de procesado. En la Figura 6, se representa en forma de diagrama de bloques el flujo que se va a seguir.

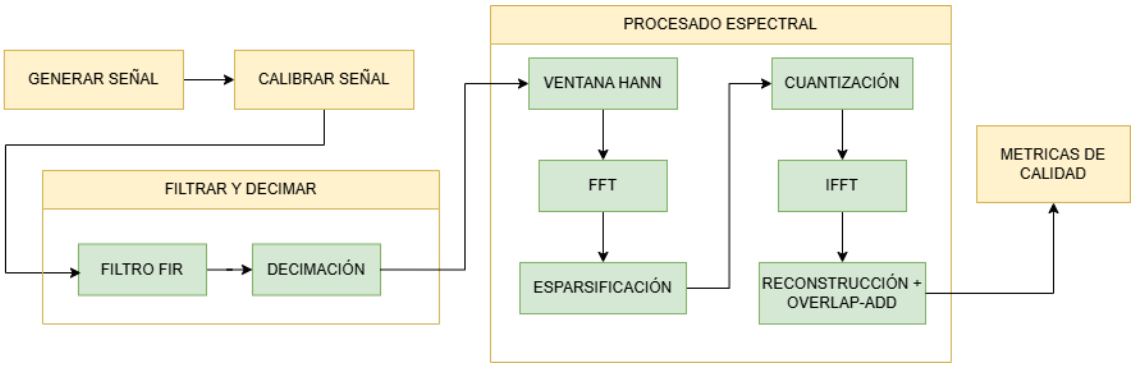


Figura 6: Diagrama de bloques del flujo del preprocesador de EW

4.2 Generar la señal

Como es lógico, se comienza generando una señal. Para ello se dan valores a las variables clave como el periodo, la frecuencia de muestreo, el vector tiempo de la señal y la propia frecuencia de la señal. Esta señal, con el fin de poner a prueba el algoritmo y proceso implementado, puede ser senoidal, triangular o cuadrada, y así comprobar que funciona correctamente. Para elegir el tipo de señal deseada, al principio del código se le debe asignar a la variable "*config.tipoSenal*" el nombre: *'seno', 'cuadrada', 'triangular'*. Después, para simular lo más parecido posible a una señal real interceptada o recibida, a la señal limpia creada se le añade ruido blanco, simulando así el ruido real que tendría.

Es importante recordar que, en el caso de probar este algoritmo con señales reales, sería necesario añadir a esta etapa un amplificador de potencia y un conversor ADC, pero debido a que la señal es simulada con funciones propias de Matlab, no es necesario. Esta decisión se

explicó más a fondo en la sección 1.1.1 Proyecto real de referencia. La cabecera de la función se muestra en la Figura 7.

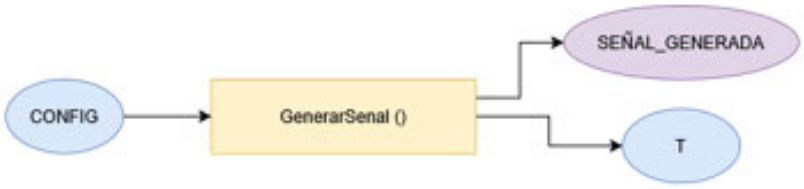


Figura 7: Cabecera de la función Generar Señal

4.3 Calibración de la señal

Como se comentó con anterioridad, se van a utilizar técnicas DASA, y finalmente la técnica elegida ha sido la calibración digital de la señal [37]. Con esta etapa lo que se consigue es reducir los desajustes de offset, ganancia y fase. Esto se hace debido a que cualquier componente de continua (por ejemplo, un desplazamiento hacia arriba o hacia abajo) puede colarse en atapas futuras como el procesamiento espectral o el filtrado, causando imprecisiones y desviando los umbrales de energía.

Además, se escala toda la señal para que el pico se quede justo en ±1. Esto asegura que la señal no va a saturar, y se aprovecha al máximo los bits disponibles en el formato Q1.15, mejorando la resolución, evitando overflow y garantizando una representación precisa de la señal.

Estos pasos además facilitan el trabajo a etapas futuras. (I) Da robustez al filtrado FIR, sin un offset, el filtro evita atenuar por error componentes cercanas a DC (corriente continua, es decir, la componente de f = 0 Hz). (II) Aporta precisión en la FFT y esparsificación, porque la normalización hace que los umbrales de energía se apliquen sobre valores comparables entre bloques, facilitando seleccionar coeficientes espectrales relevantes.

Como se comentó con anterioridad, se prioriza la arquitectura modular, y por ello después de la llamada a la función, se guarda la señal resultante en una variable aparte para su uso futuro, garantizando que la siguiente etapa no depende de esta. Su cabecera se representa en la Figura 8.

Figura 8: Cabecera de la función Calibrar Señal

4.4 Filtrado y decimación

En la siguiente etapa aplicamos filtrado y decimación. Esta etapa actúa como primera capa en el proceso de reducción de datos. Se comienza diseñando un filtro FIR paso bajo de orden (128), de modo que todas las frecuencias por encima de fs/D (frecuencia de muestreo y factor de decimación) quedan atenuadas antes de bajar la tasa de muestreo eliminando el ruido y los armónicos no deseados. En esta etapa concreta, a la hora de definir la implementación del filtro, el orden se reduce a la mitad, aunque en el resto de los procesos se mantendrá el original. Esto se hace por dos motivos: (I) el primero reducir el retardo que introduce el filtro, ya que un filtro de orden N introduce un retardo de N/2 muestras. Por lo tanto, si de reduce a la mitad el orden, se reduce a la mitad el retraso. (II) Al reducir el orden también se reduce a la mitad el número de multiplicaciones y operaciones que se realizan por muestra, lo que reduce los recursos y acelera el filtrado. Queda remarcado una vez más lo prioritaria que es la rapidez del preprocesador.

Se ha elegido un FIR de fase lineal porque ofrece una respuesta uniforme sin distorsionar la señal. Además, al aplicar después un decimado polifásico (dsp.FIRDecimator) se consigue filtrar y reducir la muestra en un solo paso, ahorrando ciclos de cálculo y simplificando la lógica HDL.

Por otro lado, la decimación [38] reduce la frecuencia de muestreo entre el factor de decimación elegido (D = 2), eliminando los datos redundantes y preservando al mismo tiempo las características esenciales de la señal. Al reducir la densidad de datos, se agiliza significativamente el flujo de datos y permite centrar el procesamiento en componentes específicos de la señal. Los dos procesos combinados no solo ayudan a aligerar la carga de procesado a los algoritmos futuros como la FFT y la esparsificación manteniendo la misma información, sino que también disminuye la memoria necesaria para los buffers y acelera el procesamiento del futuro FPGA. Además, la decimación siempre debe acompañarse de un filtrado previo para evitar el fenómeno conocido como “aliasing”, en el cuál si no se filtra adecuadamente antes de decimar, las frecuencias altas que no se han atenuado se pliegan, mezclándose con las bajas y distorsionando la señal (se explica con ejemplos visuales en las pruebas de integración de esta etapa). La cabecera de la función teniendo en cuenta la arquitectura modular, se muestra en la Figura 9.

Figura 9: Cabecera de la función Filtrar Y Decimar

4.5 Procesado espectral

La fase de procesado espectral se divide en 5 etapas distintas como muestra la Figura 3. Se ha decido juntar estas etapas en la misma función, debido a que separarlas era ineficiente ya que para ganar precisión en el procesado espectral se divide la señal en bloques, y devolver en cada fusión todos los bloques o volver a dividir en cada función la señal de nuevo, sería muy repetitivo además de añadir carga computacional innecesaria. La división en bloques nos permite aplicar la FFT con la certeza de que las características de frecuencia de ese instante están bien definidas, y además reduce notablemente la memoria y la carga computacional necesaria. Una vez más queda recalcado lo importante que es que el preprocesador sea rápido y las decisiones que se están tomando para cumplirlo.

4.5.1 Ventana Hann

Dicho esto, con el fin de mejorar la precisión de la FFT, antes de aplicarla se somete a cada bloque a una ventana raíz de Hann. Se ha elegido esta ventana, y concretamente su raíz, debido a que garantiza que al recombinar los bloques con Overlap-Add no aparezcan ni huecos ni picos indeseados ya que suaviza los bordes de cada bloque, facilitando la futura reconstrucción de la señal. Además, preserva la amplitud de cada componente espectral.

4.5.2 Transformada rápida de Fourier (FFT)

Después, se calcula la FFT normalizada de cada bloque enventanado, consiguiendo un espectro limpio, con picos definidos, que facilita enormemente la selección de los coeficientes más relevantes en la siguiente fase.

4.5.3 Esparsificación

Una vez en el dominio de la frecuencia, se aplica la etapa de esparsificación, ya que no tiene sentido quedarse con todo el espectro cuando solo una parte contiene la información importante. Esto se consigue evaluado la energía de cada coeficiente y ordenándolos de manera descendente, y se conservan solo aquellos que suman valor al reconstruir la señal, en función del umbral definido. Este método es doblemente eficiente, ya que se reduce drásticamente la cantidad de datos sin renunciar a la fidelidad perceptual, y al mismo tiempo se aligera camino de la cuantización y la IFFT (Inverse Fast Fourier Transform).

Este ahorro de información no implica necesariamente una reducción en el ancho de banda espectral (en Hz) de la señal, ya que no se ha modificado el contenido frecuencial transmitido ni se ha aplicado ninguna técnica de modulación. Para que exista un ahorro real de ancho de banda, sería necesario que la señal transmitida ocupe menos espectro tras la compresión, lo cual depende del sistema de transmisión posterior, algo que no se contempla en el alcance de este trabajo.

4.5.4 Cuantización

Teniendo ya la señal reducida, se convierten los valores elegidos a punto fijo Q1.15 garantizando la compatibilidad total con HDL. Este proceso es la cuantización. Para conseguirlo se normaliza la escala al rango ±1 para que cada coeficiente ocupe lo máximo sin desbordarse. Esto nos garantiza no perder información y ahorrar recursos, ya que las operaciones sobre enteros son mucho menos costosas que sobre decimales.

4.5.5 Transformada Rápida de Fourier Inversa y Overlap-Add

Una vez se ha destilado el espectro y cuantizado los coeficientes, volvemos al dominio del tiempo. La IFFT recompone cada bloque apoyándose solo en los coeficientes cuantizados que, al trabajar con un conjunto de datos más pequeño y limpio, devuelve la forma parcial de onda más significativa.

Para terminar, se debe multiplicar de nuevo por la misma ventana raíz de Hann para evitar huecos entre los bloques con un solapamiento del 50%, lo que se conoce como Overlap-Add [39, Cap. 18]. El resultado es una señal reconstruida continua y sin imperfecciones, llegando al final de la fase de procesado espectral de la señal.

Todos estos procesos, o etapas, se realizan en la función “procesadoEspectral”, cuya cabecera se muestra en la Figura 10, respetando la arquitectura modular. El parámetro “debug” que devuelve la función, se explica y se utiliza más adelante en el apartado 5. Resultados y Validación.

Figura 10: Cabecera de la función Procesado Espectral

4.6 Métricas de calidad

Por último, al final del código se calculan dos métricas de calidad que reflejan la calidad del proceso, y nos muestran cuánta señal original se ha conservado.

Primero, se calcula el MSE (Mean Squared Error - Error Cuadrático Medio) que revela, de media, cuanto error energético se ha introducido en total en el proceso. Cuanto más pequeño sea mejor, ya que indicaría menos discrepancias. Se muestra cómo se calcula en la ecuación 1.

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(x[decimada] - x[reconstruida]^2) \quad (1)$$

En segundo lugar, se calcula la SNR (Signal-to-Noise Ratio - Relación Señal-Ruido). Este cálculo dice cuánto la potencia de la señal original supera a la potencia del ruido, en decibelios. Lo ideal es que sea un valor alto, y así las imperfecciones serían imperceptibles. En la ecuación 2 se muestra cómo se calcula.

$$SNR_{dB} = 10 \times \ log_{10}(\frac{\sum x[decimada]^2}{\sum x[decimada] - x[reconstruida]^2}) \quad (2)$$

Por último, se representa en la Figura 11 la cabecera de la función:

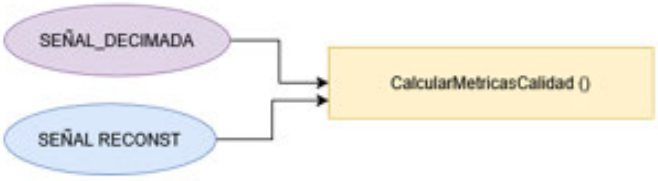


Figura 11: Cabecera de la función Calcular Métricas de Calidad

Las técnicas de reducción de datos descritas pueden mejorarse aún más mediante la integración de la IA, y aunque es un trabajo muy interesante y que aportaría un gran valor al preprocesador, se encuentra fuera del alcance este TFG. Se explica más en profundidad en la sección 8.2 Trabajos futuros.

Todo el proceso está pensado para ser integrado en hardware comercial como los COTS (Commercial Off-The-Shelf) [19], es decir, equipos comerciales listos para usar. Usar aritmética fija y estar estructurado en funciones autónomas facilita esta integración, y es un buen punto de partida para en un posible trabajo futuro aprovechar soluciones ya probadas en el mercado que cumplen con los requisitos técnicos necesarios, y probar con señales reales la calidad del proceso.

4.7 Diseño del modelo digital

Por último, en este apartado, se va a explicar el diseño del modelo digital del preprocesador que se ha implementado en Simulink, y además, se comentará la posible conversión del código a formato HDL, a través de las funciones y métodos que ofrece Simulink.

4.7.1 Propuesta del diseño digital

En esta sección se va a representar y explicar el modelo digital que se ha implementado en Simulink. Este modelo digital llamado *"PreprocesadorEW_Simulink.slx"* hace exactamente lo mismo que el código implementado *("PreprocesadorEW.m")*, pero se estructura en bloques, que son bloques que representan funciones de Matlab. El objetivo de este diseño y esta implementación es facilitar la conversión del código a HDL, ya que una estructura en bloques es mucho más sencilla, compacta e intuitiva que un script de código para HDL.

Como se ha comentado, este diseño que se representa en la Figura 12, hace lo mismo que el código implementado en el script de Matlab con las funciones explicadas de los puntos 4.1 al 4.5. Para comprobar su funcionamiento, se han insertado dos bloques "To Workspace". Estos bloques permiten guardar la señal generada y la señal reconstruida, para representarlas y validar el resultado. Por otra parte, no se ha implementado el bloque de las métricas de calidad, ya que el objetivo de esa función es medir la calidad del código y la reconstrucción, pero dado que el diseño en Simulink tiene como objetivo ofrecer una idea de lo que sería el diseño digital real en HDL, se ha decidido no incluirlo. En su lugar, se implementa en el script *"PreprocesadorEW.m"*

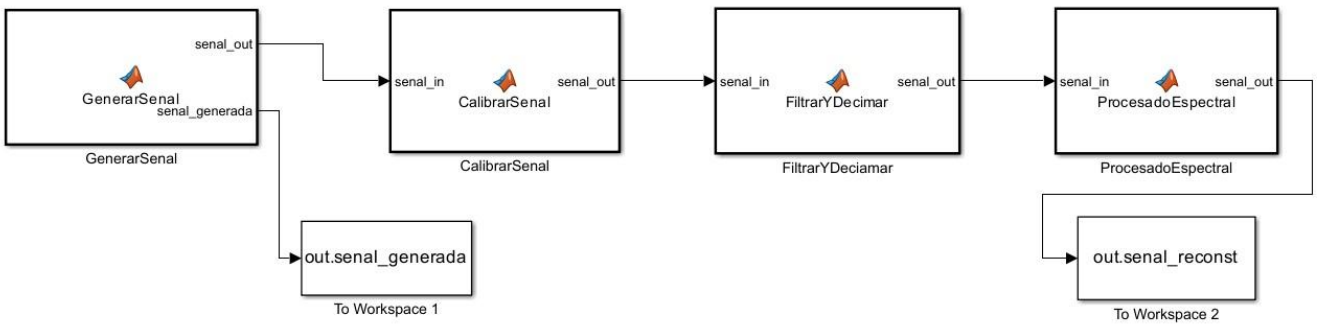


Figura 12: Diseño del modelo digital en bloques - Simulink

Una vez comentado el modelo digital, se van a explicar los resultados obtenidos. Hay que tener en cuenta que, todo el proceso que sufre la señal introduce ciertos retardos. Concretamente esos procesos son aplicar el filtro FIR, la decimación y la posterior reconstrucción. El filtro FIR introduce retardo debido a que su salida no es instantánea, mientras que la decimación altera el tiempo relativo entre muestras. Como resultado, al final del proceso, la señal reconstruida asume estos retardos, y presenta una ligera variación respecto a la original.

La consecuencia de estos retardos es que los primeros instantes de la señal reconstruida, en comparación con los de la señal original, sean nulos. El resultado se muestra con una señal senoidal para mantener un mínimo parecido con lo que sería una señal real, en la Figura 13.

Esto puede parecer algo muy negativo, pero teniendo en cuenta que los primeros instantes de una señal suelen ser ruido e interferencias generadas por la emisión de la señal, y que además esos primeros instantes en los que la señal es nula solamente equivalen a 20 µs, valor tan pequeño que en la vida real sería prácticamente inapreciable, no es tan perjudicial.

Lo importante respecto a la reconstrucción de la señal, es apreciar que la señal sea más "limpia", es decir, que se haya reducido o eliminado el ruido, que se haya atenuado la amplitud, aunque no demasiado, y que se mantenga la forma de la onda, tanto el periodo como la geometría.

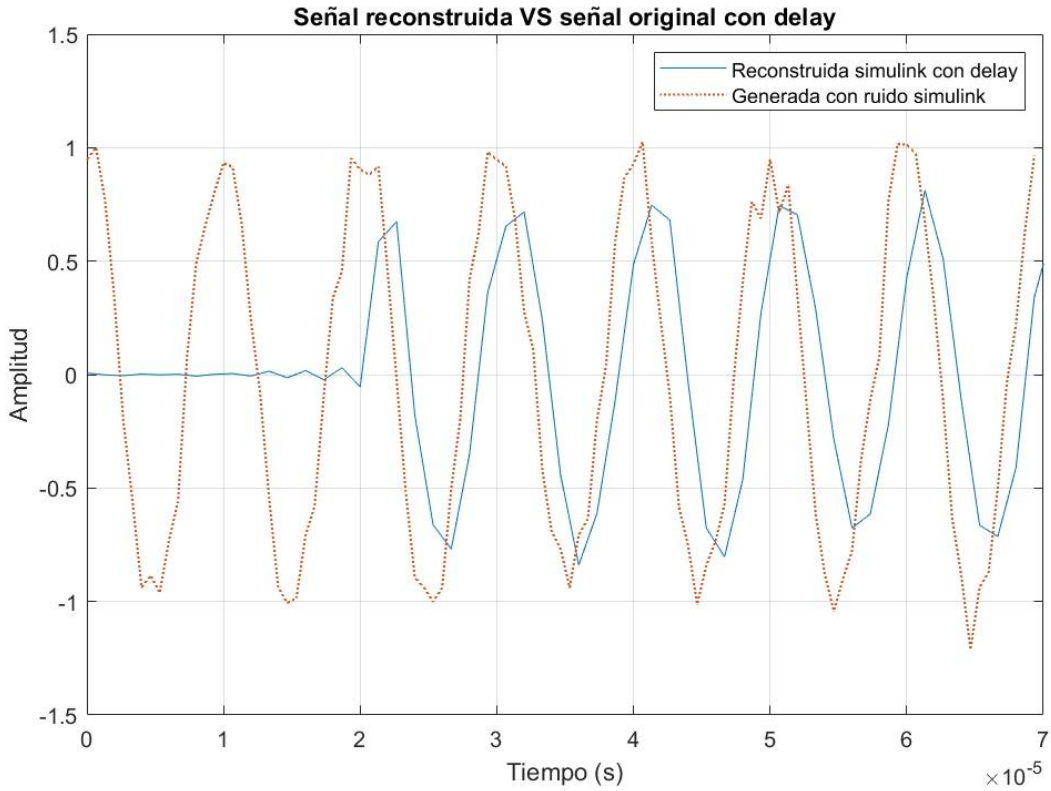


Figura 13: Comparación señal reconstruida VS generada con delay – SIMULINK

El resultado del proceso se puede apreciar que es muy positivo, aunque puede estar mejor. Afortunadamente, es posible reducir o incluso eliminar el delay que ha sufrido la señal a través del siguiente procedimiento.

En primer lugar, se debe calcular el retardo total introducido por el filtro FIR y la decimación. Después, se recortan las muestras afectadas, es decir, las muestras nulas. A continuación, se interpola la señal por el mismo factor que se hizo la decimación, para volver a la frecuencia de

muestreo original establecida. Por último, se quita el retraso que introduce la interpolación que se ha aplicado. La implementación de este proceso se representa a continuación:

```
1  % Cálculo de retardos introducidos
2  retardo_FIRYDecimacion = ceil((ordenFIR / 2) / D);
2  retardo_interp = (ordenFIR / 2) / 2;
3  retardo_total = retardo_FIRYDecimacion + retardo_interp;
4
5  % Alineación e interpolación
6  senal_alineada = senal_reconst((retardo_total+ 1):end);
7  x_reconst_interp = repelem(senal_alineada, D);
8
9  % Señal final sin retardo
10 x_recosnt_final = x_reconst_interp((retardo_total * D) + 1:end);
```

La representación final de la señal reconstruida sin retardo, se muestra en la Figura 14. Lo que se debe de apreciar es la misma señal que en la Figura 13, pero sin el delay.

Por otra parte, interpolar la señal con "*repelem*" (línea 7 del código superior) provoca que la señal final reconstruida presente forma escalonada. Este efecto se puede evitar usando en su lugar la función de interpolación *"interp",* pero se ha decidido no hacerlo.

Que la señal reconstruida tenga forma escalonada aporta grandes ventajas tanto para un procesador digital, como para una FPGA, frente a una señal de amplitud continua. Estas ventajas son:

1) Optimización de recursos: Al limitarse un conjunto finito de niveles, la amplitud puede codificarse con menos bits. Esto se traduce en un menor consumo de recursos computacionales tanto como para el procesador posterior, como para la integración y el trabajo futuro de una FPGA.
2) Facilidad de control y depuración: Trabajar con valores discretos, facilita y agiliza multitud de operaciones aritméticas. Además, los saltos y posibles anomalías se detectan inmediatamente, cosa que en una señal continua quedarían enmascaradas entre muchísimos valores. Y, por último, los tramos planos generan secuencias repetidas que se comprimen con ganancia alta, reduciendo el ancho de banda que debe transportar o almacenar el sistema.
3) Compatibilidad con punto fijo fi (Q1.15): Este tipo de datos, ofrecen mayor compatibilidad con bloques posteriores, ya que gracias a las ventajas del Q1.15 se pueden controlar efectos como el overflow, explicados anteriormente.

Los tres aspectos son claramente positivos para cumplir el requisito **R.5** del catálogo de requisitos establecido en la Tabla 5 en el apartado 3.2 Catálogo de requisitos del proyecto.

Como efecto secundario negativo, cabe destacar que trabajar con este tipo de señales sacrifica la fidelidad de la señal con la original, pero es un coste asumible cuando el objetivo principal del preprocesador es detectar eventos relevantes de la manera más rápida y eficiente posible, manteniendo la compatibilidad con FPGA.

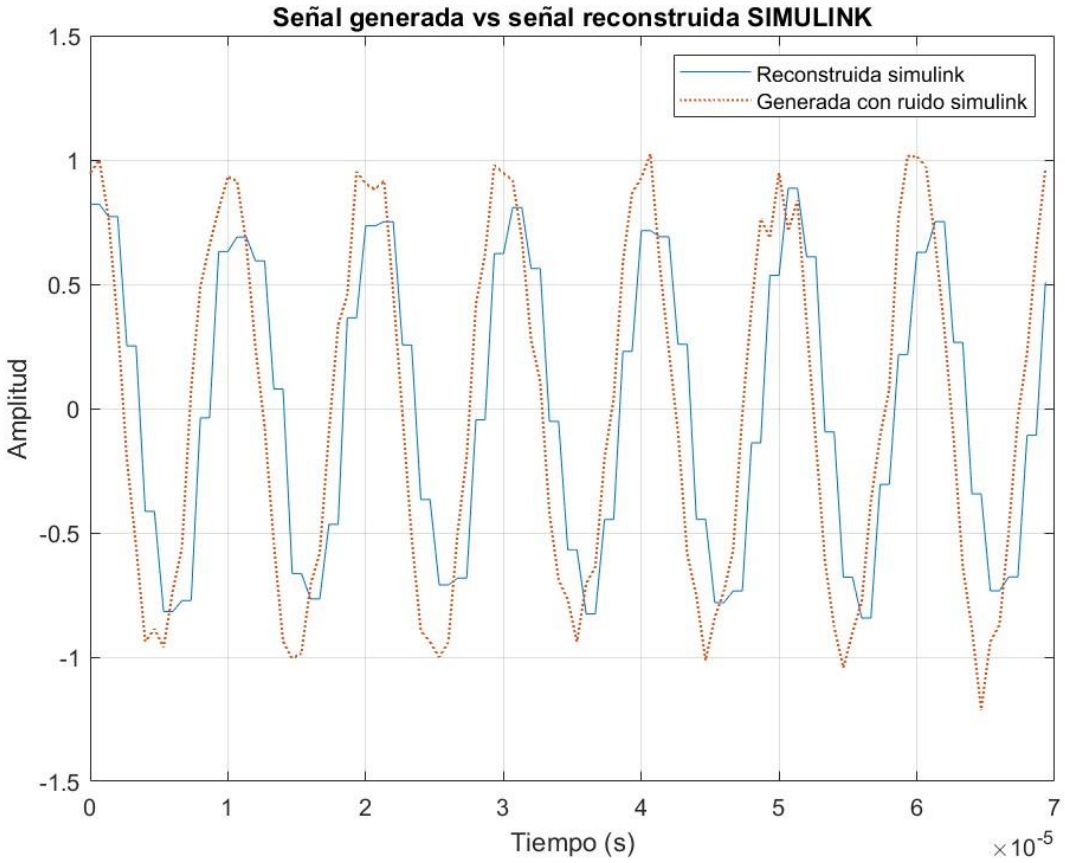


Figura 14: Comparación señal reconstruida VS generada SIN delay – SIMULINK

Se demuestra que el resultado obtenido es muy satisfactorio, y cumple con las expectativas establecidas. Se ha eliminado casi por completo el retraso o delay introducido, en términos generales se mantiene la geometría de la señal original, y se ha atenuado lo suficiente la amplitud para reducir la cantidad de información y agilizar su interpretación.

4.7.2 Conversión a HDL

Como se ha explicado a lo largo del manuscrito, uno de los objetivos del proyecto es que el sistema desarrollado, y en particular cada uno de los módulos que conforman su arquitectura, estén diseñados de manera que su conversión a código HDL sea sencilla. Esto permitirá una futura integración del preprocesador en una FPGA. Con este fin, Matlab ofrece dos opciones de trabajo.

En primer lugar, hacer directamente la conversión de código Matlab a HDL con una herramienta llamada HDL Coder, mencionada durante el estudio de herramientas. Esta herramienta traduce algoritmos escritos en Matlab Code (extensión .m) a HDL sintetizable.

En segundo lugar, generar código HDL a partir de un modelo jerárquico de bloques en Simulink, aprovechando el diseño digital ya implementado y estructurado en bloques. Posteriormente, usando HDL Workflow Advisor se genera código HDL.

Aunque el código y modelo digital implementados en Matlab y en Simulink, se han estructurado y pensado para facilitar la conversión a HDL, hay operaciones aritméticas y procesos, que se requieren para conseguir llevar a cabo la reducción de datos establecida de manera eficiente, cuya conversión exige un desarrollo nuevo de carácter impredecible.

Por este motivo, y con el objetivo de orientar al lector de cara a un posible desarrollo futuro, en la sección titulada 8.2 Trabajos futuros, se incluye una descripción general del proceso de conversión a código HDL, así como de los pasos necesarios para la integración del sistema en una FPGA.

5. Resultados

En este punto, se van a representar los resultados obtenidos en el proceso. Primero, se han realizado las pruebas unitarias. Estas pruebas consisten en, pasar a la señal generada con ruido por cada etapa independiente, para comprobar y validar su correcto funcionamiento. Después, se hará algo parecido, pero esta vez siguiendo y representando la señal durante todo el proceso hasta el final, demostrando el cambio que poco a poco va sufriendo hasta el final, lo que se llaman las pruebas de integración.

Para la facilidad del lector en las dos pruebas, se van a representar las comprobaciones visuales siempre en los primeros 7 períodos de la señal cuando la representación lo permita. Por ejemplo, cuando la señal esté dividida en bloques durante el procesado espectral no se podrá.

Además, para que sea más fácil identificar la señal en cada punto del proceso, siempre se va a representar la señal con un color específico dependiendo de en qué parte del proceso esté. Por ejemplo, la señal después de pasar por la etapa de calibración, siempre se va a representar verde. Las señales con sus respectivos colores se muestran en la Tabla 6.

Tabla 6: Colores representativos de las señales en las pruebas

Señal	Color representativo	
Generada	Azul Oscuro	
Generada con ruido	Rojo	
Calibrada	Verde	
Filtrada y decimada	Amarillo	
Enventanada	Magenta	
Post FFT + esparsificación + IFFT	Gris	
Reconstruida final	Cyan	

5.1 Pruebas unitarias

Durante el desarrollo del preprocesador, y en la fase de elección de las etapas que finalmente se han implementado, se llevaron a cabo las pruebas unitarias. Su función es asegurar y comprobar que cada proceso elegido e implementado por el que pasaría la señal, desempeña su trabajo correctamente. Sin embargo, para no representar demasiadas imágenes parecidas a las que se van a representar durante las pruebas de integración, se ha decido representar solo las de la función "*procesadoEspectralSenal*", que es la función más crítica y en la que más cambio se debe ver. Además, también se van a comentar las aportaciones obtenidas durante estas pruebas, que fueron muy satisfactorias y reveladoras.

5.1.1 Aportaciones de las Pruebas Unitarias

Gracias a estas pruebas, se detectó que se estaban cometiendo errores graves en el orden de algunos procesos. En primer lugar, se había implementado un umbral adaptativo justo antes del procesado espectral, que consiste en analizar y seleccionar las frecuencias más relevantes de la señal, y en función de esas frecuencias definir un umbral que posteriormente haría de filtro. Tras comparar los resultados después de este umbral, con los resultados de la esparsificación, resultaron sumamente parecidos, y resultó que básicamente se estaba haciendo el mismo proceso dos veces. Este error habría supuesto un doble procesamiento completamente innecesario, añadiendo complejidad computacional que ralentizaría el preprocesador.

Del mismo modo, se estaba canalizando, es decir, dividiendo en varios canales la señal, al mismo tiempo que aplicaba la decimación, sin realizar un filtrado previo. Como se ha explicado en el punto 4.3 Filtrado y Decimación, esto habría provocado que las frecuencias superiores a la de Nyquist ($f_s/2$) [40] se plieguen distorsionando gravemente la señal.

Gracias a estas pruebas, se corrigió el proceso y se ha podido llevar a cabo una buena implementación del preprocesador y obtener buenos resultados.

5.1.2 Pruebas unitarias de la función Procesado Espectral

A continuación, se van a representar los resultados que ofrece la función "*procesadoEspectralSenal*" que se ha implementado en el preprocesador. Como se ha comentado anteriormente, el objetivo de las pruebas unitarias es verificar que el funcionamiento de cada componente o etapa del proceso funciona correctamente.

Para hacerlo, se ha implementado un script llamado *"pruebasUnitarias.m"* en el que se llama a la función "*procesadoEspectralSenal*" y se pasa como parámetro directamente la señal generada con ruido en la función *"generarSenal",* así se comprobará directamente si de verdad la señal se esparsifica en cualquier condición, que es su objetivo. Esta prueba va a tener las siguientes características o particularidades.

- Se va a representar la señal después de pasar por la función, tanto en el dominio del tiempo como en el de la frecuencia. Esta representación aporta información muy importante ya que, durante la función, la señal pasa y sufre cambios en ambos dominios.
- La prueba se va a realizar con una señal triangular. De esta forma, al tener este tipo de señal picos más marcados, se va a apreciar mejor la reducción de información, es decir, la atenuación de amplitud.
- Para que el resultado sea más visible, se ha bajado el factor de energía retenida de 0.9999 a 0.95. Esto se debe a que con el factor 0.9999, sin que la señal pase antes por algún proceso de reducción de datos, el sistema no atenúa prácticamente nada. Para probar la eficacia de la esparsificación, se tiene que forzar al sistema a descartar más coeficientes. Además, al bajar dicho factor, la reconstrucción puede perder fidelidad y pueden aparecer diferencias visuales en pendientes o esquinas, que también es lo que se quiere evaluar.

En primer lugar, se representa en la Figura 15 la comparación en el tiempo de la señal antes y después de la función "*procesadoEspectralSenal*". Como se ha explicado anteriormente, se va a respetar el código de colores establecido.

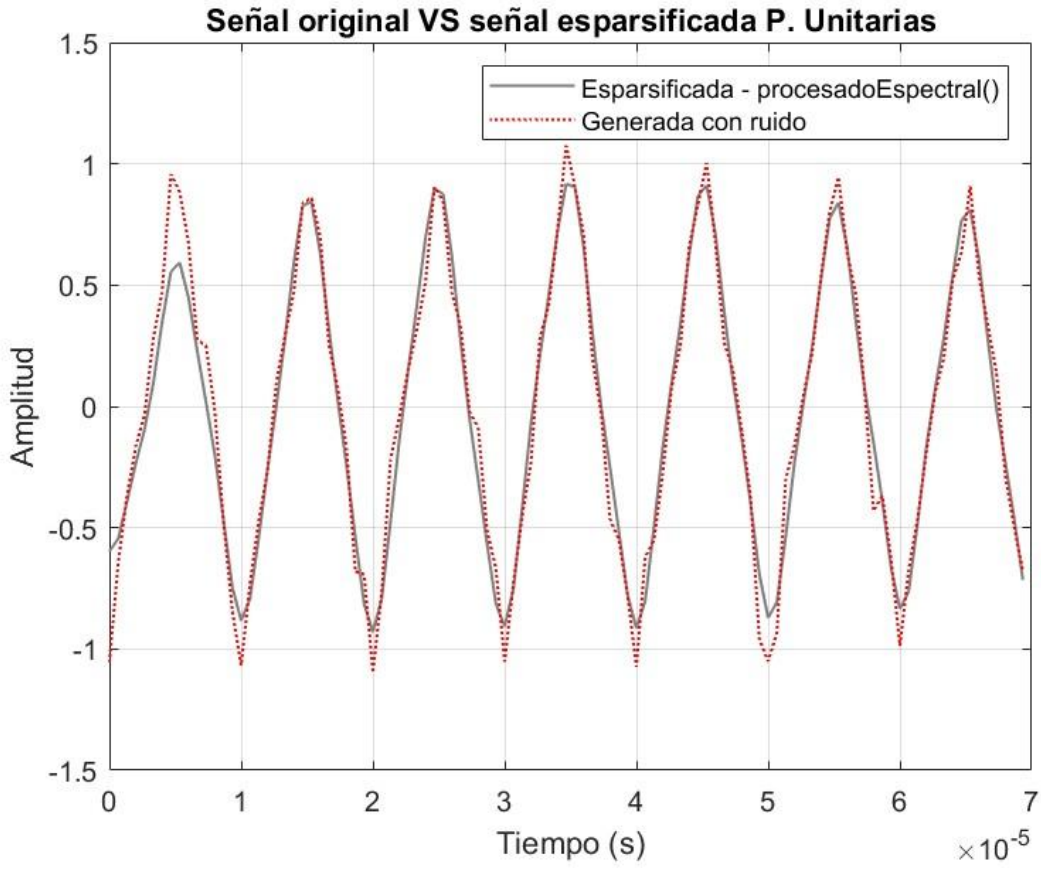


Figura 15: Comparación señal generada VS esparsificada [s] - P. Unitarias - TRIANGULAR

Como se puede apreciar, la señal esparsificada (gris) ha sufrido una evidente atenuación de la amplitud. Además, se mantiene la forma triangular de la señal y no se ha deformado la señal, que es justo el que se esperaba ver. Es decir, la señal atenúa, no deforma, y mantiene su forma

y frecuencia fundamental original, lo que es un resultado muy satisfactorio y confirma la eficacia de la compresión.

Además, se observa claramente el efecto de la ventana Hann, que como se ve en la **Figura 33**, deben sufrir más cambio las bajas y altas frecuencias. Aunque al representar solamente los primeros 7 periodos, solo se muestra el efecto en las bajas.

En segundo lugar, se representa en la Figura 16 la comparación de las señales en el dominio de la frecuencia. Lo que se espera ver, según la naturaleza de la señal triangular, es un espectro con el armónico a la frecuencia fundamental (100 kHz) muy marcado, y más armónicos consecutivos decrecientes en las frecuencias impares, es decir: 300 kHz, 500 kHz…. Además, se ha quitado la réplica de la señal acortando el eje X para tener una representación más directa y limpia.

Estos efectos se explican más en profundidad en los espectros de la sección 5.2.2.2 Comprobaciones visuales de las pruebas de integración de la calibración de la señal.

También se debe observar que no aparezca ningún armónico indeseado entre los impares, una reducción del ruido espectral gris, y una atenuación de la amplitud de los armónicos grises (señal esparsificada) respecto a los rojos (señal original) de manera exponencial. Es decir, retiene más en los primeros picos y pierde más en los siguientes.

Observando la Figura 16 se confirma que el resultado es óptimo. La esparsificación ha tenido un efecto directo, visible y muy positivo en el espectro.

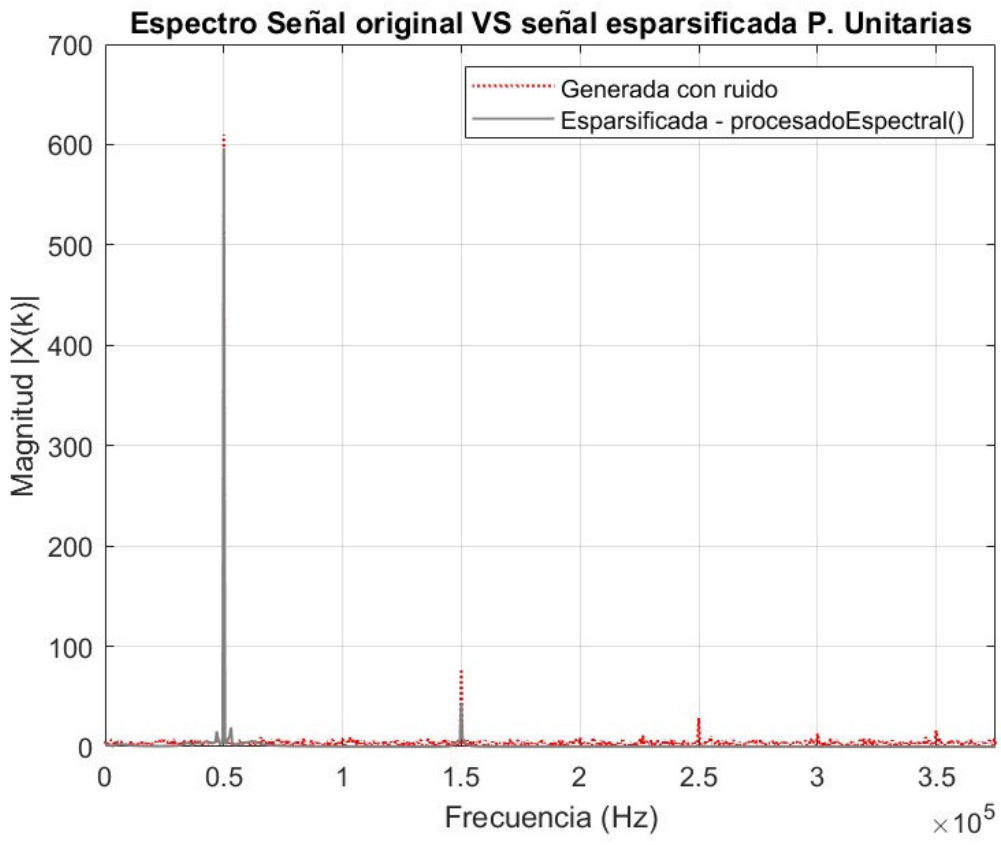


Figura 16: Comparación señal generada VS esparsificada [Hz] - P. Unitarias – TRIANGULAR

5.2 Pruebas de integración

Estas pruebas se van a dividir en 6 (I) generar señal (II) calibración de la señal (III) filtrado y decimación de la señal (IV) ventana Hann y FFT (V) cuantización, esparsificación e IFFT y (VI) Overlap-Add y reconstrucción. Se ha decido esta estructura intentando separar lo máximo posible las fases, pero al mismo tiempo valorando si va a ser notable el cambio entre una fase y otra. Esto quiere decir que, si parte de una fase, un proceso concreto, se realiza para preparar el siguiente, el cálculo numérico y efecto visual en la señal sería nulo, como ocurre con la ventana Hann y la FFT que se han juntado.

De cada etapa en las que se han separado las pruebas, se van a representar y explicar los resultados numéricos, en los que se calculan datos relevantes de la señal que demuestran el funcionamiento del proceso que ha sufrido la señal en cada caso, y además, resultados visuales para poder comprobarlo analizando como va cambiando la forma de la señal.

Todas las pruebas se van a hacer con los tres tipos posibles de señales que se han implementado: senoidal, cuadrada y triangular, con el fin de poner a prueba lo máximo posible al procesado de la señal y comprobar su buen funcionamiento.

Aunque se van a representar las gráficas más relevantes de los tres tipos de señales, con el motivo de no sobrecargar al lector con los mismos cálculos y razonamientos, de las señales cuadrada y triangular no se van a representar los cálculos numéricos, excepto alguno que se considere muy importante. De esta manera, se va a tomar la señal senoidal como la referente, explicando y comentando todos los resultados sobre ella, al ser su geometría la más parecida a lo que sería una señal de radiofrecuencia real.

Estas pruebas se han implementado en un script aparte llamado *"pruebasDeIntegracion.m"* que se adjunta, junto con el código principal del preprocesador llamado "*PreprocesadorEW.m"*. En la propia implementación de "*pruebasDeIntegracion.m"* se llama al código principal, con la intención de tener guardadas en el Workspace de Matlab todas las variables necesarias para realizar las comparaciones, pruebas y representaciones.

También, es importante explicar la creación en la función de "*procesadoEspectralSenal*" de una nueva estructura llamada "debug". La razón de esta nueva estructura es guardar todas las variables necesarias durante el procesado espectral. De esta manera, se pueden utilizar desde el script de pruebas de integración sin necesidad de incluirlas en la cabecera de la función, y así poder representar y calcular datos antes y después de cada etapa que implementa esta función.

5.2.1 Pruebas de integración de la generación de la señal

En esta fase, se muestra en la Figura 17 la señal senoidal generada limpia, es decir, sin ruido ni ninguna imperfección, junto con la señal con ruido, que simula la señal que se recibiría a través de una antena en el avión de combate. Aunque no es una variación muy agresiva, es lo suficiente como para complicar la interpretación de la señal, por lo que al final del proceso se espera una señal más sencilla y limpia.

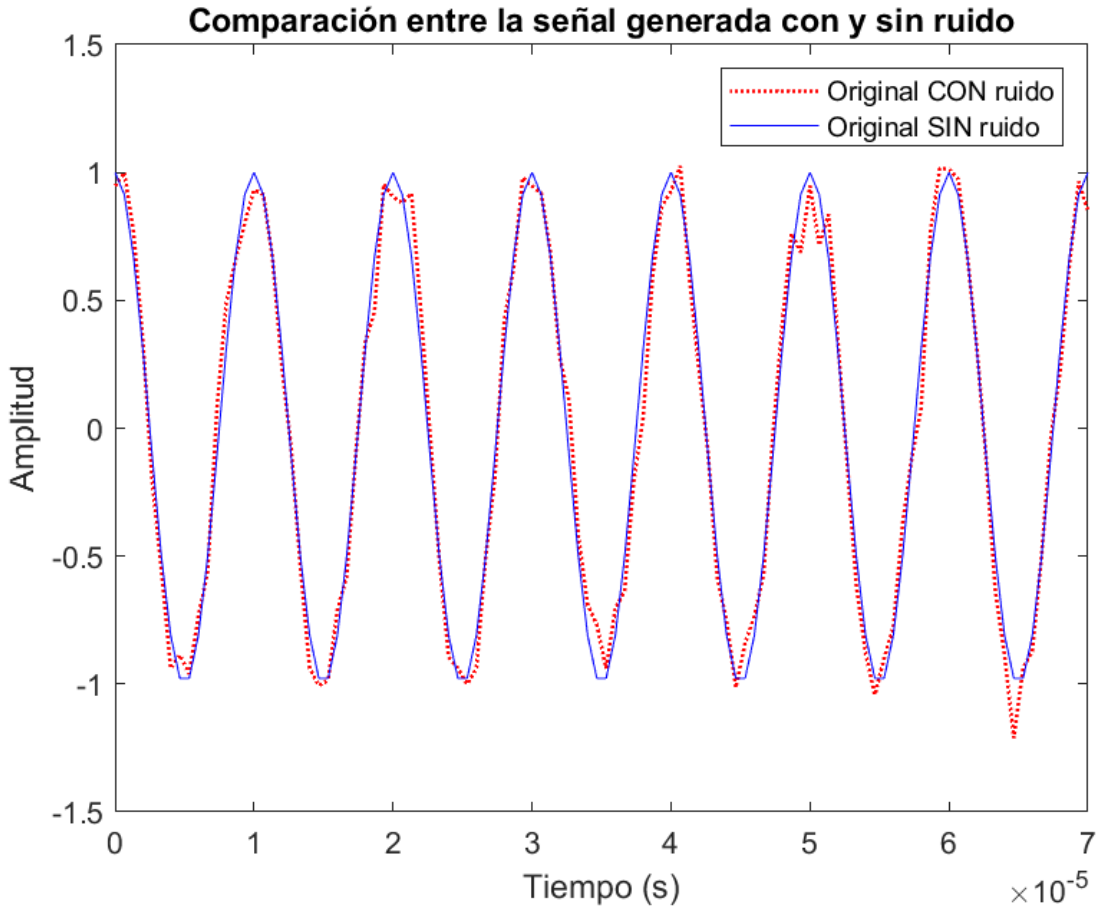


Figura 17: Comparación señal original VS señal con ruido – SENOIDAL

Del mismo modo, se muestran en la Figura 18 la señal cuadrada generada, limpia y con ruido; y en la Figura 19 la señal triangular generada y con ruido. Las tres señales tienen los mismos parámetros de configuración, solo cambia la forma de onda.

A priori, comparando las tres gráficas, se puede ver que, aunque el nivel de ruido establecido es el mismo para las tres, el de la señal cuadrada (Figura 18) es el más abrupto, debido a su naturaleza como señal que tiene cambios de energía mucho más agresivos que el resto.

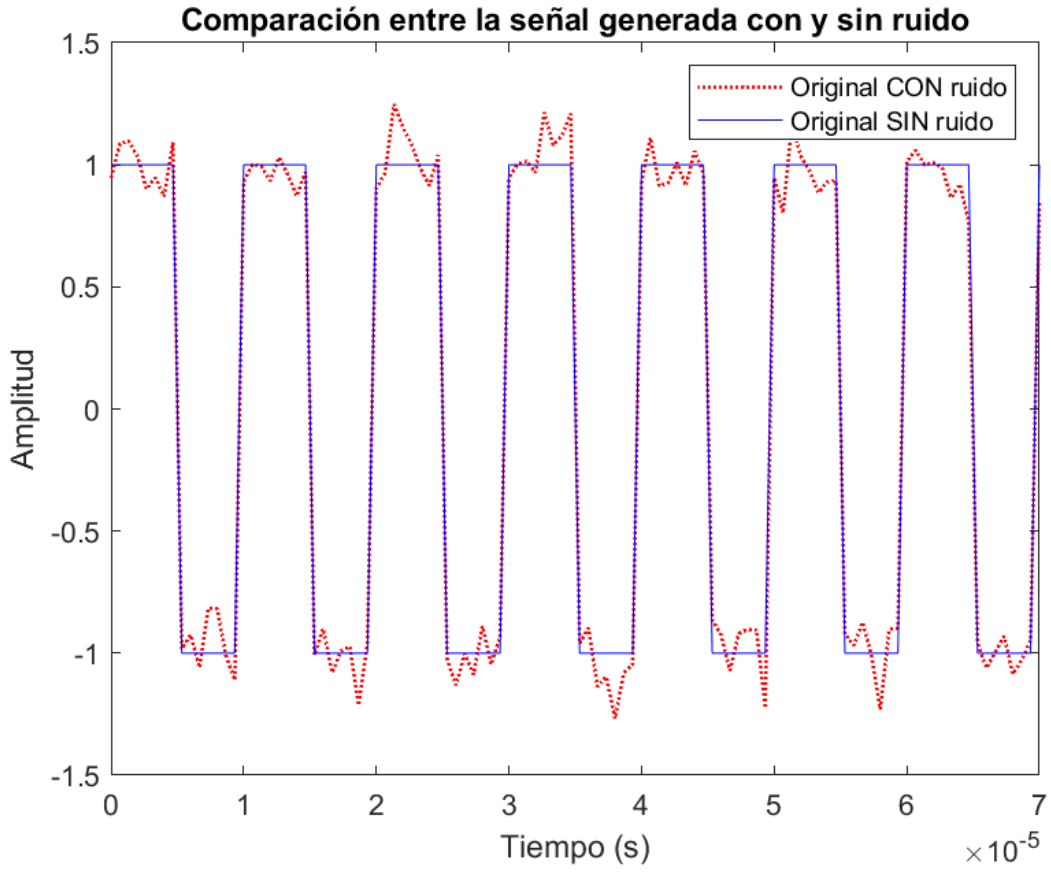


Figura 18: Comparación señal original VS señal con ruido – CUADRADA

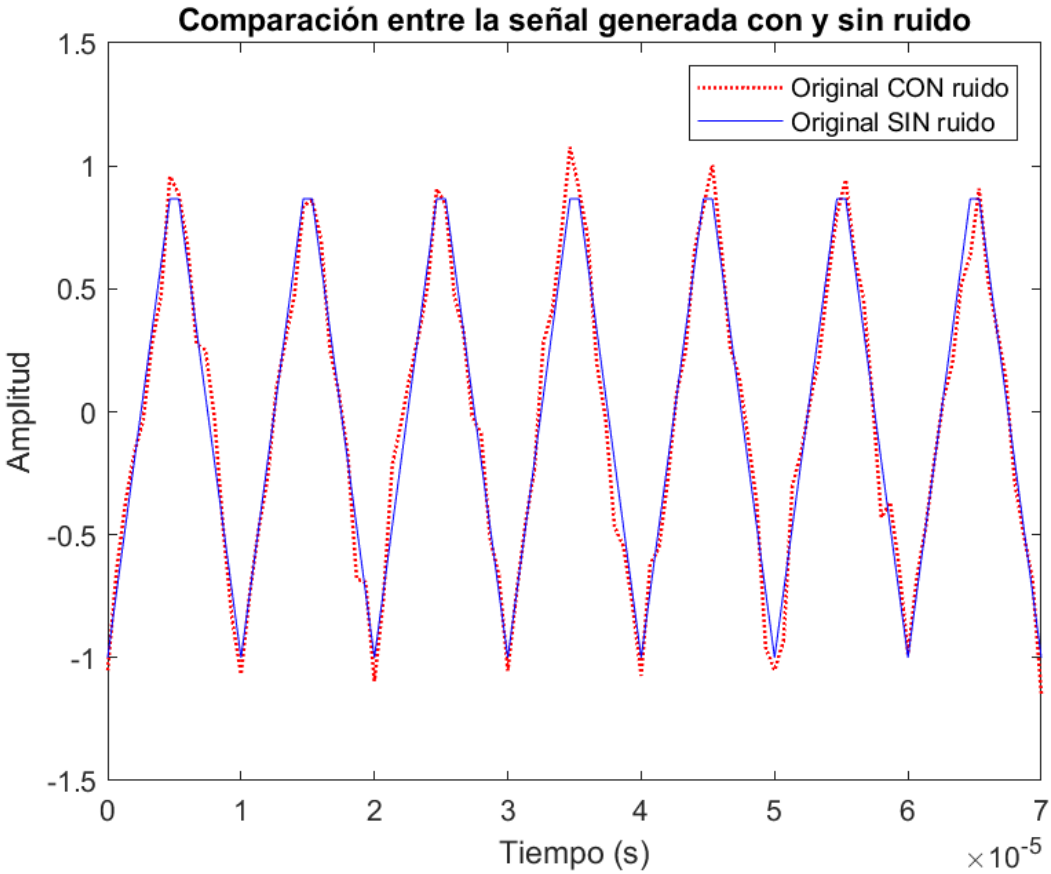


Figura 19: Comparación señal original VS señal con ruido – TRIANGULAR

5.2.2 Pruebas de integración de calibración de la señal.

En este apartado se va a validar el correcto funcionamiento de la función "calibrarSenal". Para ello, nos vamos a centrar en comprobar dos objetivos clave: eliminar el offset y normalizar el pico a 1, sin que se introduzca distorsión visible ni componentes espectrales en la señal modificada.

Para tener una referencia con lo que comparar los resultados, se ha fijado un umbral de 0.001% sobre el pico unitario, lo que da una tolerancia absoluta de 10^{-4}, que refleja un equilibrio entre eficiencia y robustez frente al ruido introducido a la señal.

5.2.2.1 Comprobaciones numéricas

Se ha impreso por pantalla el resultado de la media del error entre la señal original y la calibrada, y la desviación típica que se calcula con la función propia de Matlab "*std()*" de los tres tipos de señales. Los resultados se muestran en la Tabla 7.

Tabla 7: Tabla comparativa Media de Error y Desviación Típica de las tres señales

	Senoidal	***Cuadrada***	***Triangular***
Media del error	1.221×10^{-7}	4.069×10^{-8}	1.424×10^{-7}
Desviación típica	8.712×10^{-6}	8.810×10^{-6}	8.828×10^{-6}

Todos los resultados están muy cerca de 0. Las medias del error son prácticamente nulas en los tres casos, y las desviaciones están por debajo del umbral establecido, lo que confirma un offset virtualmente nulo, y un pico normalizado fiel a la tolerancia.

5.2.2.2 Comprobaciones visuales

Se han generado gráficas representativas de varias señales que ayudan a comprobar el resultado de la calibración. En primer lugar, en la Figura 20 se representa en el dominio del tiempo la señal original con ruido antes de calibrar, es decir, con offset, y superpuesta esa misma señal después de calibrar. La señal después de la calibración reduce levemente su amplitud, lo que reduce su carga computacional y se convierte en una señal más fácil de manejar para las siguientes etapas. Además, sus geometrías son idénticas por lo que se demuestra que no se ha alterado la forma de la onda.

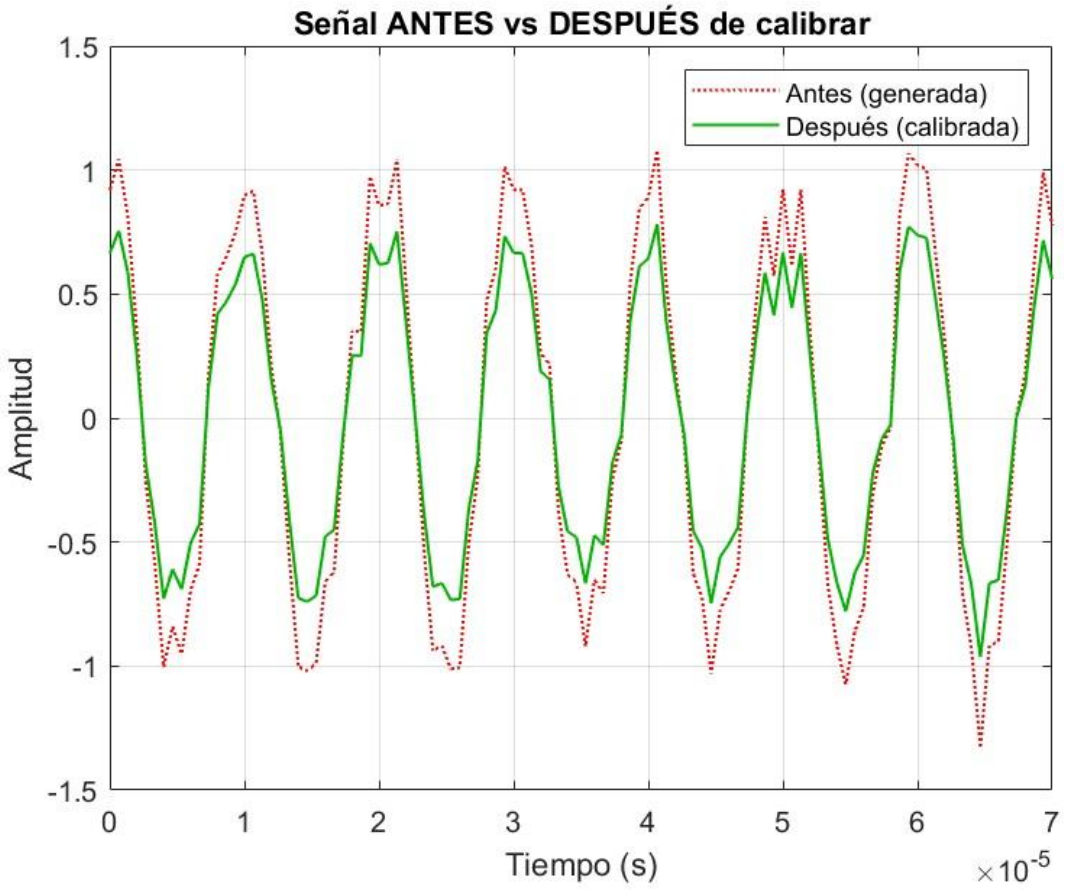


Figura 20: Comparación señal sin calibrar VS señal calibrada [s] - SENOIDAL

Ahora, se representa la misma gráfica, pero de la señal cuadrada (Figura 21) y triangular (Figura 22). De nuevo, ocurre que visualmente la que más ruido y más imperfecciones presenta es la señal cuadrada, aunque solo llevamos una etapa del procesado. En cambio, la que más "limpia" aparenta, es la triangular, presenta picos y caídas muy simétricas.

Una cosa importante es que, aunque se ha normalizado la señal a ±1, en ninguna de las tres señales la amplitud llega a ese valor. Este resultado era de esperar, debido a que al calibrar, se divide toda la señal por el máximo absoluto que aparece en todo el vector, y esto incluye el ruido, que pude provocar que el factor por el que se multiplica la señal para normalizarla sea menor que 1. Además, se representan únicamente los primeros 7 periodos, lo que necesariamente no quiere decir que en ningún momento llegue al valor de ±1.

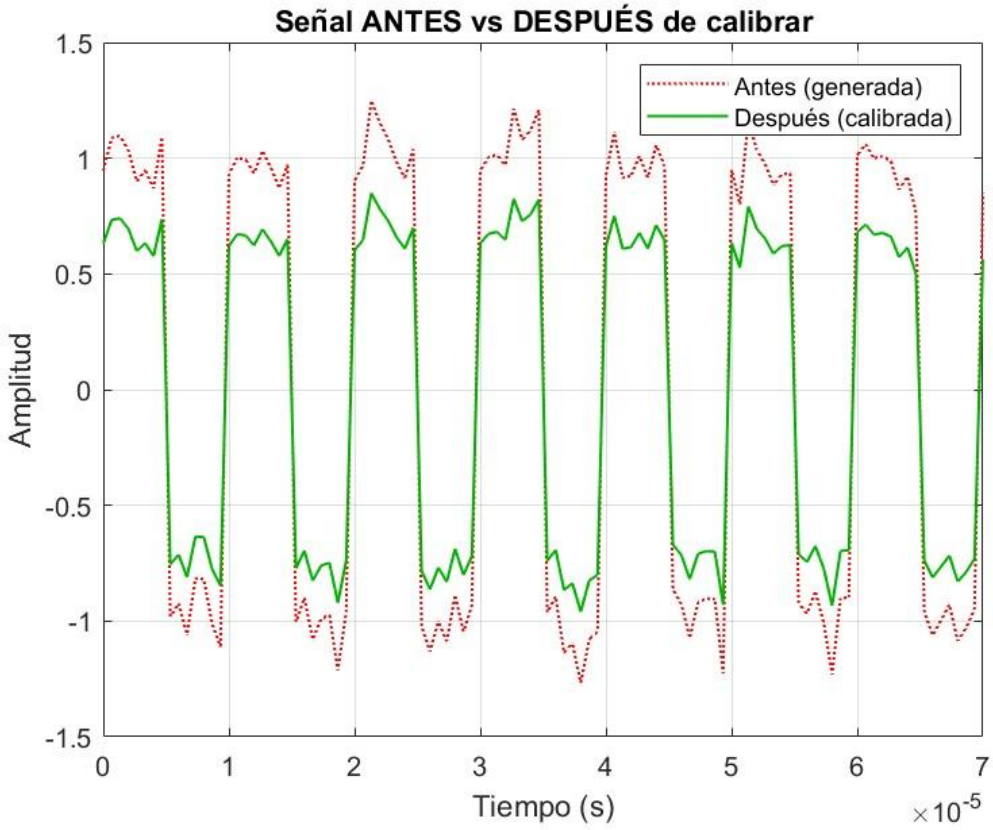


Figura 21: Comparación señal sin calibrar VS señal calibrada [s] - CUADRADA

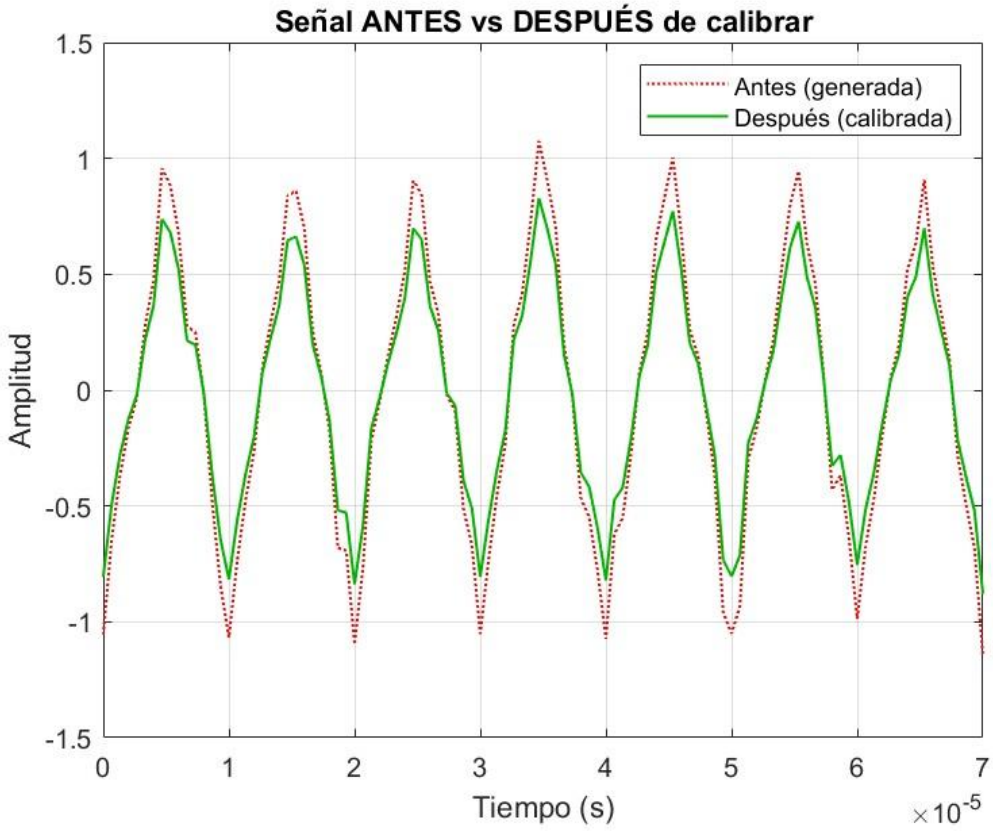


Figura 22: Comparación señal sin calibrar VS señal calibrada [s] – TRIANGULAR

En segundo lugar, la Figura 23 compara en el dominio de la frecuencia el módulo de la FFT antes y después de calibrar, donde únicamente se aprecia una reducción en la componente de 100 kHz, y el lóbulo principal y el nivel de fondo de ruido no han sido alterados. Esto vuelve a demostrar que la calibración no introduce armónicos ni irregularidades.

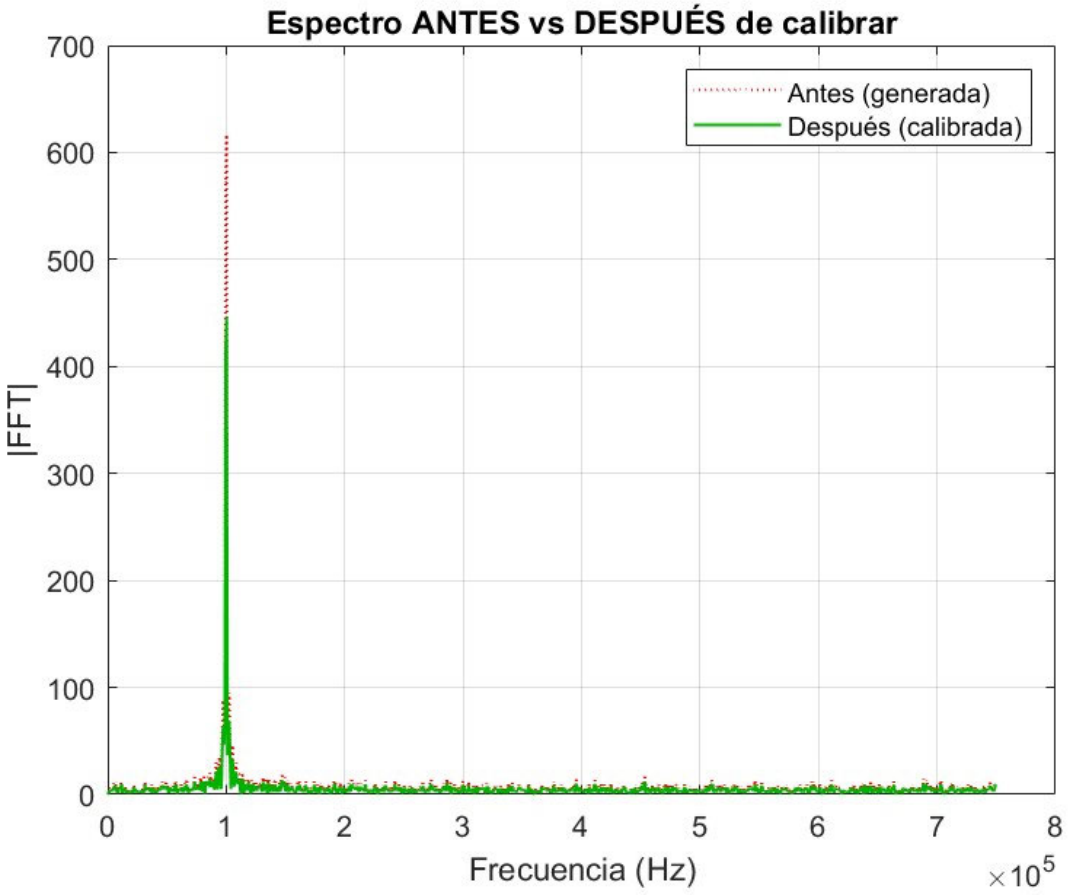


Figura 23 Comparación señal sin calibrar VS señal calibrada [Hz] - SENOIDAL

Se representan ahora el mismo espectro de la señal cuadrada en la Figura 24 y de la señal triangular en la Figura 25. Lo primero que salta a la vista comparando los espectros con el de la señal senoidal, es la cantidad de armónicos que tienen. Esto es debido a que, en la señal cuadrada al tener saltos instantáneos en el tiempo, genera una FFT compuesta por armónicos impares ($3f_o$, $5f_o$, $7f_o$, …) siendo la fundamental f_o = 100 kHz. Por eso, se crea una hilera de picos espaciados en múltiplos impares de 100 kHz decreciendo en 1/n (siendo n el armónico), es decir, el tercer armónico está a 1/3 del fundamental.

Aunque esto es la teoría, en la práctica aparecen armónicos en las f_o pares también, aunque leves. Esto se debe a varios factores que alteran la señal, el ruido blanco añadido, la conversión a Q1.15 y que realizar la FFT sin haber aplicado una ventana todavía esparce energía alrededor del fundamental, pueden provocar que algún armónico par sobreviva.

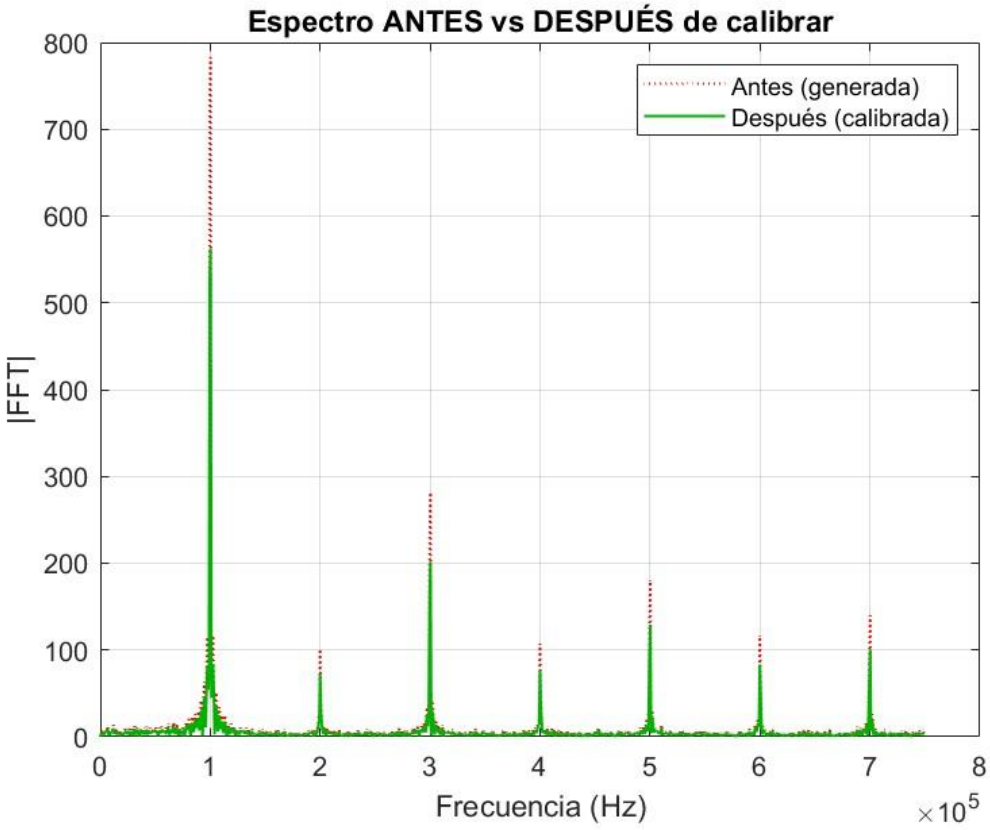


Figura 24: Comparación señal sin calibrar VS señal calibrada [Hz] – CUADRADA

Por otro lado, en el espectro de la señal triangular (Figura 25) ocurre algo similar, solo que al no haber saltos bruscos de amplitud si no que sus pendientes son progresivas, solo aparecen armónicos impares también que, esta vez, decaen en $1/n^2$, debido a que en los picos o vértices las pendientes sí cambian instantáneamente, es decir, por ejemplo, el quinto armónico vale 1/25 del fundamental. En la práctica, se traduce a que solo el tercer armónico sobrevive.

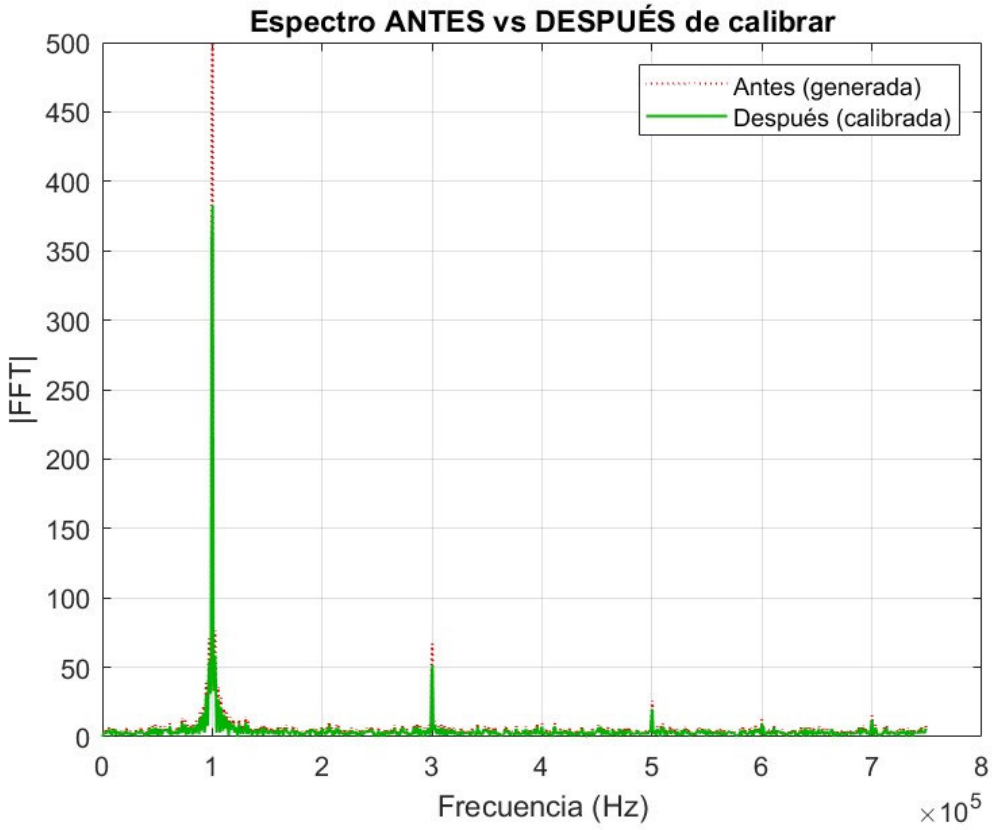


Figura 25: Comparación señal sin calibrar VS señal calibrada [Hz] – TRIANGULAR

5.2.3 Pruebas de integración de filtrado + decimación

Ahora, se van a realizar las pruebas de integración de la etapa de filtrar y decimar la señal, correspondiente a la función "filtrarYDecimarSenal". Igual que en el caso anterior, se han realizado tanto pruebas numéricas como gráficas.

5.2.3.1 Comprobaciones numéricas

Como se ha comentado anteriormente en la descripción de la propuesta, concretamente en 4.3 Filtrado y decimación, en este caso filtro FIR de orden 128/2 = 64, sirve para atenuar las frecuencias demasiado altas que podrán generar interferencias o "aliasing" y eliminar las frecuencias que no sean de utilidad debido al ruido o a información redundante. Por otro lado, al decimar lo que conseguimos reducir el número de muestras en función del factor de decimación elegido, en este caso D=2, agilizando así los procesos futuros, y manteniendo la misma información, siempre y cuando se haya filtrado la señal antes para no crear aliasing. La única pega del proceso es que suele introducir un retraso en la señal después de ser procesada, aunque tan pequeño que en la vida real será prácticamente indetectable.

Por este motivo, las comprobaciones que se van a realizar son revisar si se han eliminado algunas frecuencias, ver que el filtro no amplifica ni atenúa la señal a través de su ganancia (G) que idealmente debe ser 1, y confirmar que el número de muestras antes y después de la decimación con D=2 son correctas.

Después de ejecutar el script "*pruebasDeIntegracion.m*" los mensajes que se imprimen por pantalla informan de que:

$$Pico\ posterior: 0.8841\ <\ pico\ inicial: 1$$

$$Ganancia\ DC\ medida\ del\ filtro: 1.00$$

$$Long.Inicial: 1500\ muestras. Long. tras\ decimar: 750\ muestras$$

Viendo los resultados corroboramos que la etapa de filtrado y decimación funciona correctamente al no amplificar la señal, y aunque tampoco debería atenuarse al ser G=1, es una atenuación muy leve que se da por válida. Por otro lado, la longitud después de la decimación es justo la mitad lo que es correcto por ser D=2.

5.2.3.2 **Comprobaciones visuales**

Ahora se va a representar la señal antes y después del filtrado y la decimación, en el dominio de la frecuencia y del tiempo para corroborar los cambios que debe haber sufrido la señal, aunque, antes de nada, en la Figura 26 se muestra la respuesta en frecuencia del filtro FIR implementado, tanto su magnitud como su fase. La representación se hace automática con la función de Matlab "*freqz()*".

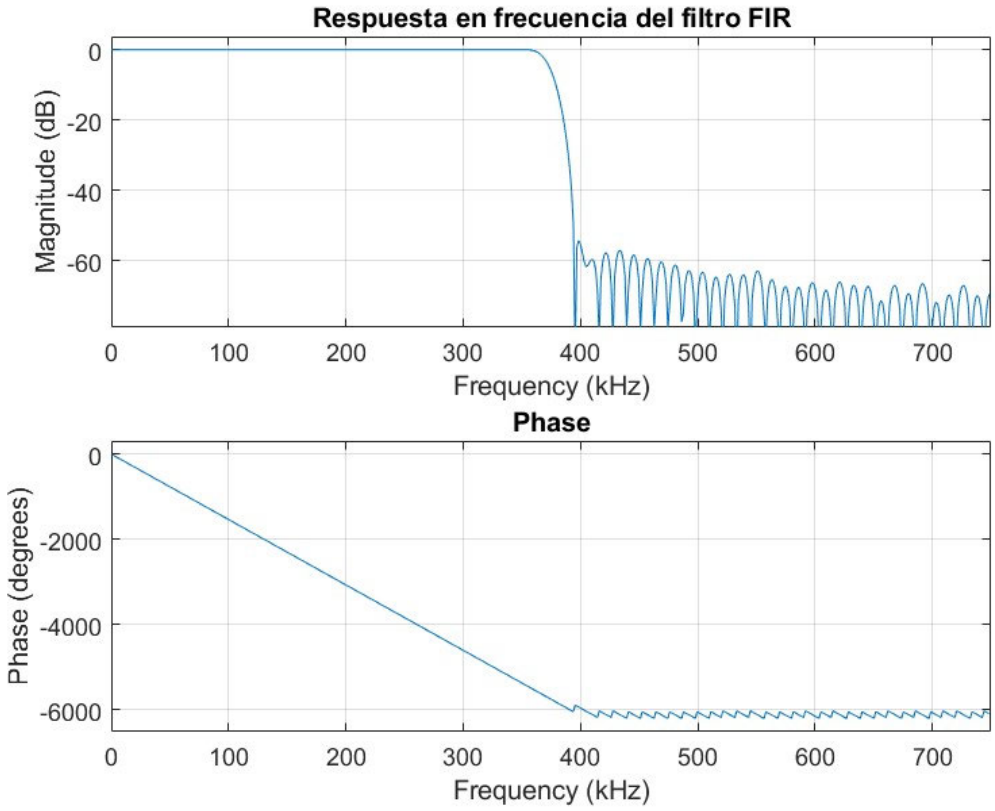


Figura 26: Filtro FIR implementado

En la Figura 27 se pueden apreciar varias cosas. En primer lugar, la eliminación de las primeras muestras de la señal hasta los 20 µs aproximadamente, que generalmente suelen ser ruido e interferencias. Además, también ha introducido un retraso de 1.3 µs entre los picos de las señales, lo que se ha explicado que podría suceder y es algo normal. Este retraso se ha reducido más gracias a dividir el orden del filtro entre 2, cuando se ha implementado. Por último, respecto a la amplitud, se ve que se ha reducido levemente gracias a la decimación al reducir el número de muestras, pero al conservar la misma geometría aproximada se confirma que no se pierde información, y casi lo más importante, se mantiene la frecuencia y el periodo de la señal original por lo que no introduce aliasing.

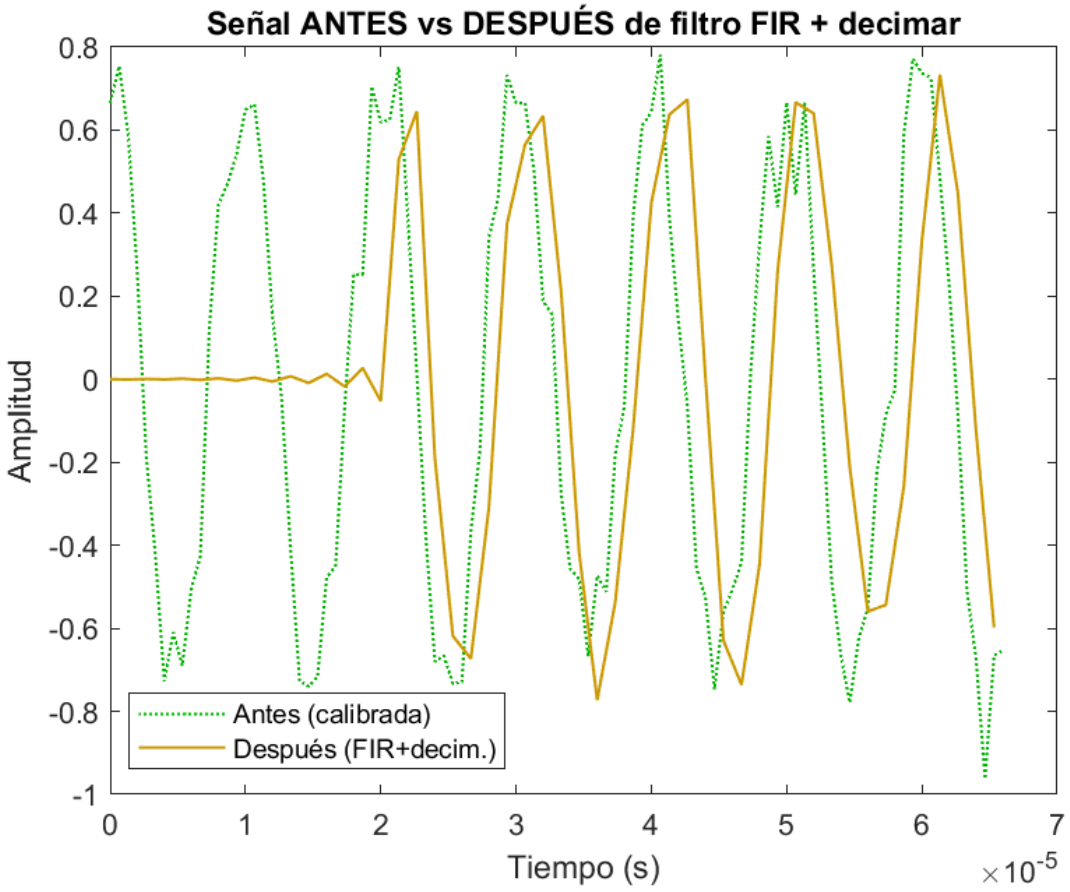


Figura 27:Comparación señal sin filtrar ni decimar VS señal filtrada y decimada [s] - SENOIDAL

Se representan ahora en la Figura 28 y la Figura 29, en el dominio del tiempo, la señal cuadrada y señal triangular respectivamente, después del filtrado y la decimación. Se aprecia que todas sufren aproximadamente el mismo retardo de 20 µs, y en líneas generales se mantiene la forma de onda, y también se mantiene la frecuencia y el periodo de la señal original por lo que tampoco introduce aliasing.

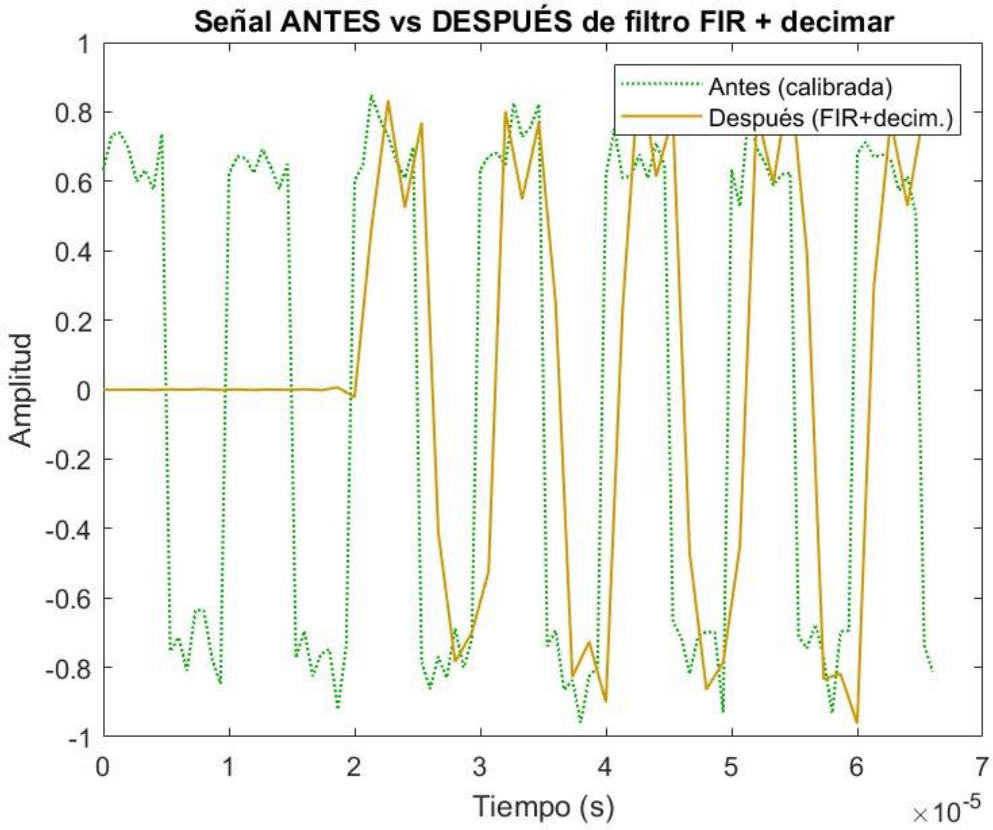


Figura 28: Comparación señal sin filtrar ni decimar VS señal filtrada y decimada [s] - CUADRADA

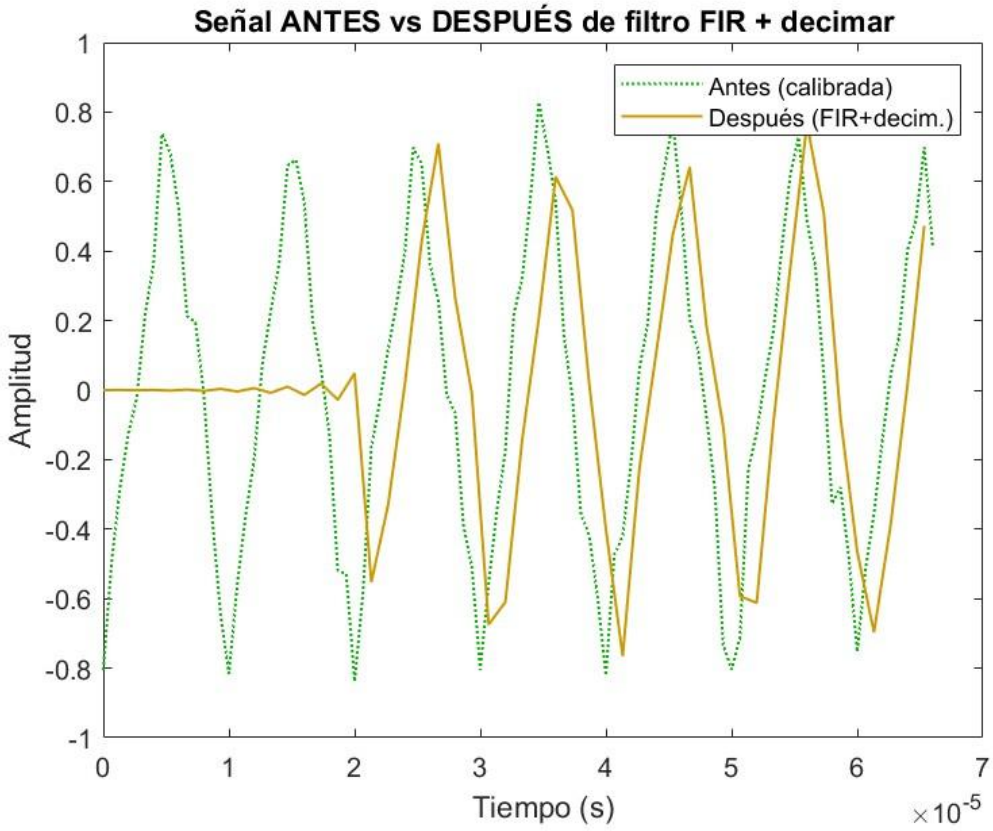


Figura 29: Comparación señal sin filtrar ni decimar VS señal filtrada y decimada [s] - TRIANGULAR

Después, en la Figura 30 en el dominio de la frecuencia de la señal senoidal, lo primero nuevo que se aprecia es que a los 650 kHz aproximadamente, aparece un armónico que no estaba anteriormente. Esto se debe a que cuando se baja la tasa de muestreo D veces, la transformada FFT de la señal se duplica D veces, en los intervalos que se muestran en la ecuación 3.

$$f_0, f_s' - f_0, f_s' + f_0 \ldots \tag{3}$$

En nuestro caso concreto, siendo $f_0 = 100\ kHz$ la frecuencia del armónico fundamental, y f_s' lo calculado en la ecuación 4, la réplica del armónico fundamental debe aparecer según la ecuación 5 en $650\ kHz$, que como vemos en la Figura 6 se cumple.

$$f_s' = \frac{f_s}{D} = \frac{1.5\ MHz}{2} = 750\ kHz \tag{4}$$

$$f_s' - f_0 = 750\ kHz - 100\ kHz = 650\ kHz \tag{5}$$

Además, el espectro de la señal senoidal demuestra más cosas. (I) Los picos principales (los armónicos de la señal) aparecen ambos en la misma frecuencia, y con la misma energía. (II) Las líneas magentas bajan de nivel, es decir, que el nivel de ruido espectral entre los picos se reduce ligeramente. (III) No aparecen armónicos extraños por lo que no hay aliasing. (IV) El armónico fundamental aparece en la misma frecuencia que en la original (Figura 23). Por estos motivos, se puede confirmar que la etapa de filtrado + decimación funciona correctamente.

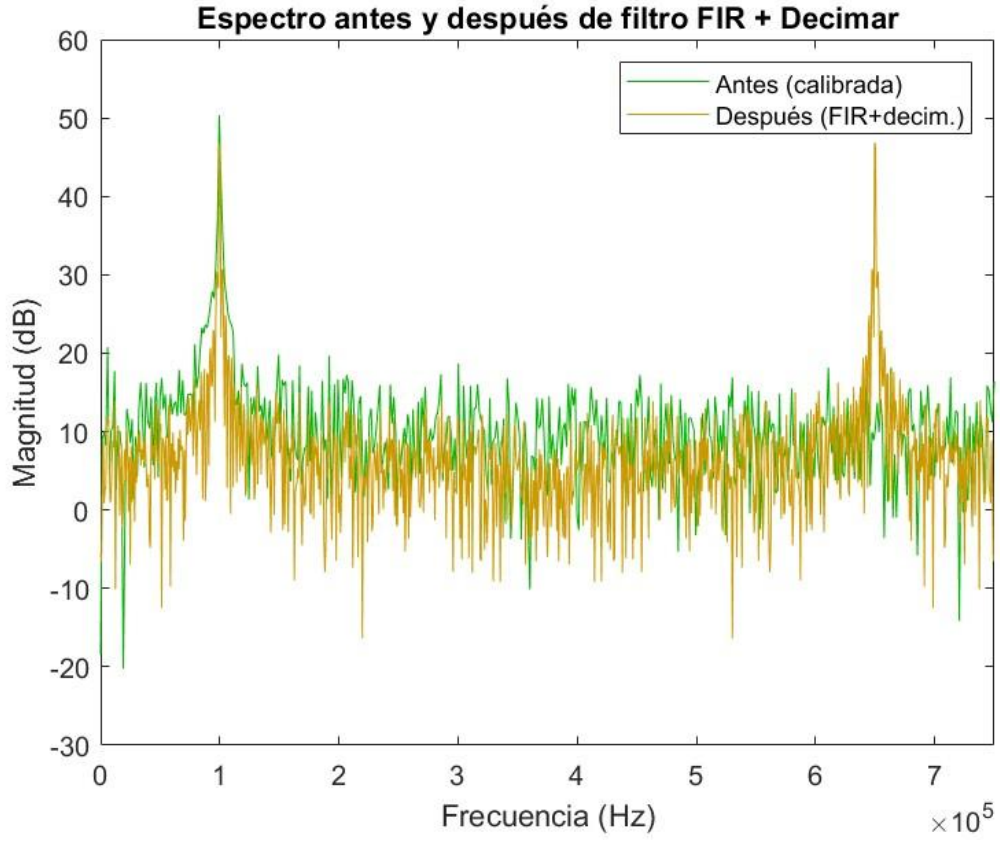


Figura 30: Comparación señal sin filtrar ni decimar VS señal filtrada y decimada [Hz] - SENOIDAL

En este punto, el espectro de la señal cuadrada (Figura 31) y señal triangular (Figura 32), ocurre lo mismo que en el espectro de la señal senoidal. La FFT se duplica D veces, y se refleja como un espejo después de los armónicos principales, y los armónicos a altas frecuencias por el efecto del filtro FIR quedan eliminados, actuando como "anti-aliasing".

Por ejemplo, en el espectro de la señal triangular se ve claro: los armónicos principales que son el fundamental a 100 kHz y el tercero (como se ha dicho antes quedan los impares) a 300 kHz, se reflejan dos veces y el resto quedan atenuados. Se refleja como un espejo, no es que se repita, quedando el reflejo del armónico fundamental igual que en la señal senoidal a 650 kHz y el tercer armónico como se muestra a continuación en la ecuación 6:

$$Tercer\ armónic.señal\ triangular = 750\ kHz - 300\ khz = 450\ khz \qquad (6)$$

El resultado se puede extrapolar al espectro de la señal cuadrada, y se confirma que es el resultado esperado.

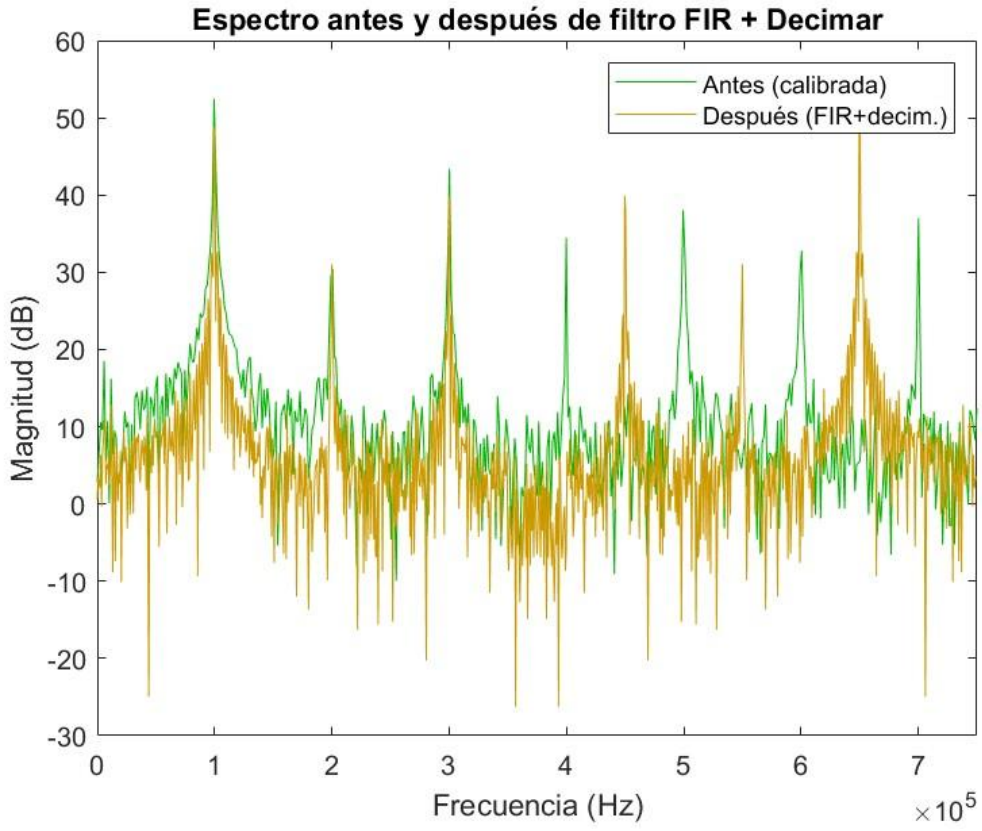


Figura 31: Comparación señal sin filtrar ni decimar VS señal filtrada y decimada [Hz] - CUADRADA

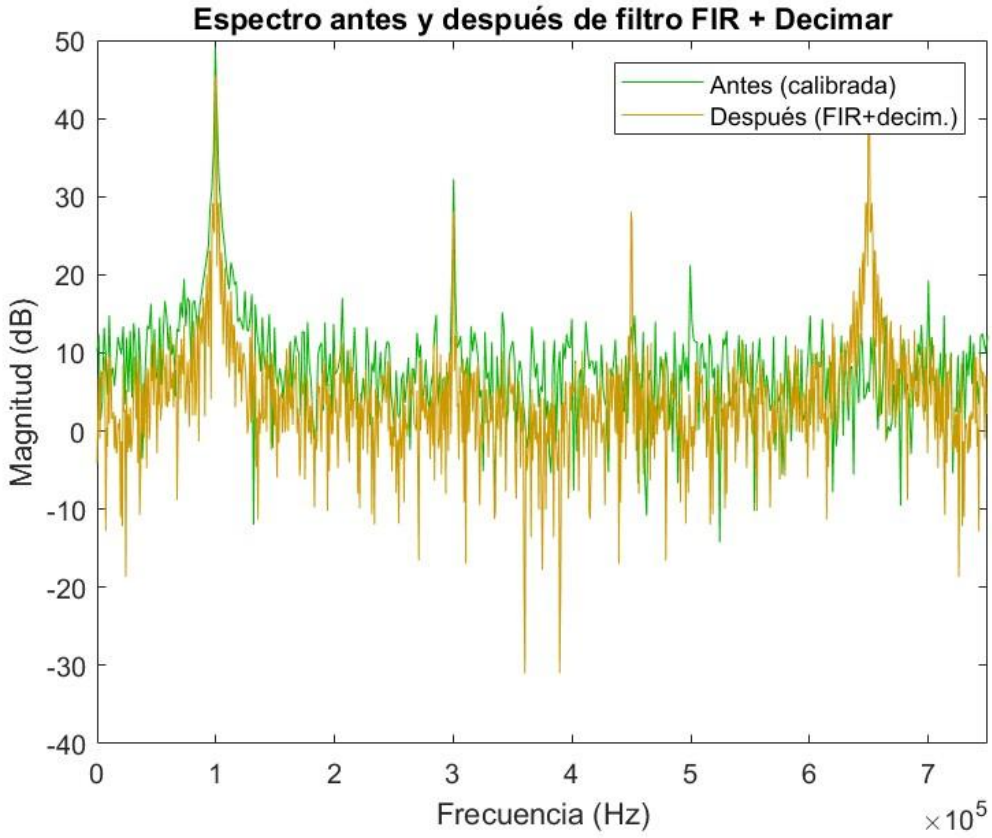


Figura 32: Comparación señal sin filtrar ni decimar VS señal filtrada y decimada [Hz] – TRIANGULAR

5.2.4 Pruebas de integración de ventana Hann + FFT

Este es el primer bloque de la función "procesadoEspectral", que se va a centrar en las comprobaciones del conjunto de las etapas de usar la ventana Hann y la FFT, para comprobar lo que ocurre antes de la FFT. Como se ha explicado, esta etapa va a aportar al proceso una reconstrucción muy fiable, y una FFT más limpia que ayuda a elegir los mejores coeficientes. En estas etapas, la señal ya está dividida en bloques por lo que todas las representaciones serán del primer bloque (k=1).

5.2.4.1 **Comprobaciones numéricas y visuales**

En esta sección se ha decidido juntar las comprobaciones numéricas y visuales para facilitar al lector su corroboración. Para confirmar el buen funcionamiento, se van a realizar primero comprobaciones antes de aplicar la ventana, sobre su longitud, simetría y valor de pico comparándolos con los valores teóricos que deben salir. En segundo lugar, las comprobaciones se van a centrar en verificar que el bloque de datos tras aplicar la ventana Hann coincide exactamente con la referencia teórica calculada. Y en tercer lugar la diferencia máxima entre la FFT creada y su valor de referencia teórico, juntando en los tres casos las comprobaciones numéricas y visuales.

5.2.4.1.1 Antes de aplicar la ventana Hann

Aplicando la función “length()” que te devuelve la longitud, de la ventana creada durante el proceso, y la creada de referencia teórica, en ambas sale que su longitud es de 512, ya que la longitud de la ventana debe coincidir con el tamaño del bloque de datos para que la multiplicación elemento a elemento sea válida.

$$[NFFT, numBloques] = [512, 5]$$

La raíz de Hann, cuando no está cuantizada, alcanza el valor 1 en el centro. Después de convertirla a Q1.15 el valor máximo debe de estar muy cera de 1, aunque con una pequeña diferencia introducida por la cuantización. Comprobarlo asegura que la implementación de la ventana no va a ser achatada ni hinchada. Que sean iguales es muy complicado ya que este tipo de operaciones siempre introduce error, pero sí que deben ser parecidas.

$$Pico\ de\ ventana: creada = 0.99997\ vs\ teórica = 1$$

Ahora, es importante corroborar que la ventana es simétrica, ya que en caso contrario se podrían generar armónicos o distorsiones en la FFT. Para hacerlo se mide la discrepancia entre el primer y el último punto, el segundo con el penúltimo, y así sucesivamente. El resultado visual se muestra en la Figura 33, que es la misma para las tres señales (senoidal, cuadrada y triangular). Su simetría y pico unitario validan visualmente las comprobaciones numéricas. La diferencia de simetrías resulta de 6.13×10^{-3} que es prácticamente imperceptible, por lo que se da por válido.

Después de estas comprobaciones, se garantiza que la ventana implementada en la función “procesadoEspectral” no va a introducir distorsiones al filtrar cada bloque antes de la FFT.

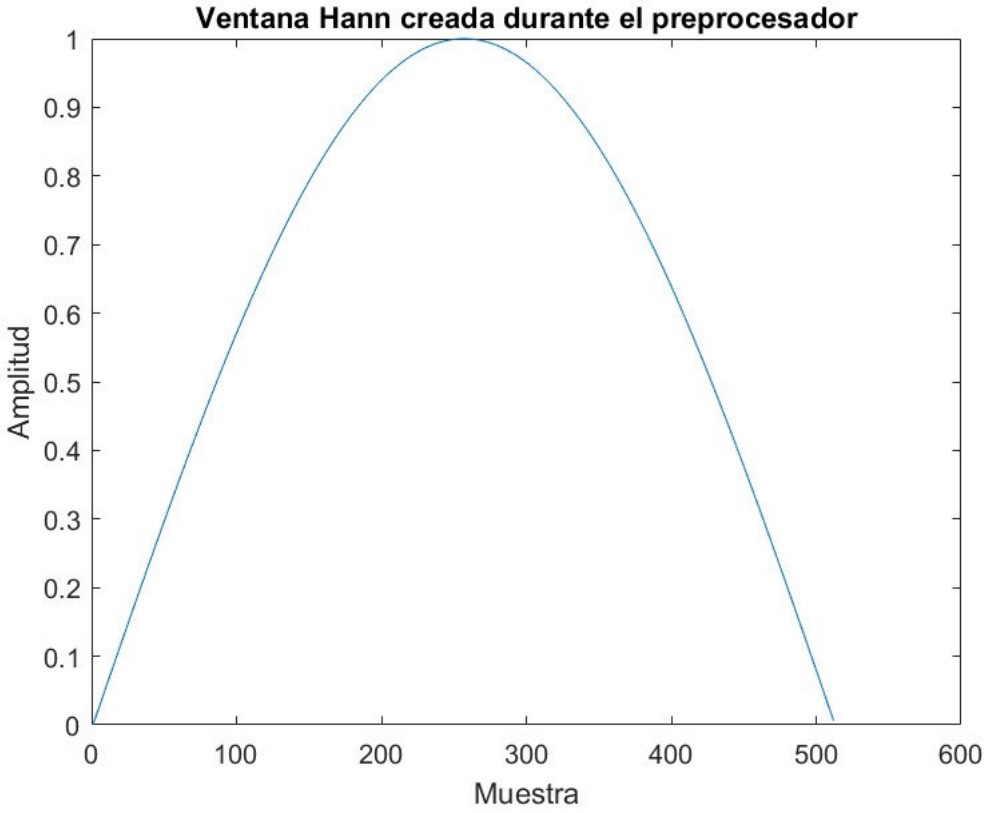


Figura 33: Representación prueba unitaria Ventana Hann

5.2.4.1.2 Después de aplicar la ventana Hann

Ahora, se va a comprobar que el bloque de datos, habiendo aplicado la ventana coincida con el de referencia calculado. Para calcularlo, se va a comparar elemento a elemento el de referencia calculado teóricamente con lo que devuelve el código implementado, y se extrae el error. Este paso es necesario porque si hay alguna discrepancia puede modificar drásticamente la energía del bloque, es por esto por lo que en este caso el error debe ser 0.

Tras los cálculos, efectivamente el error sale 0, como se ve en el mensaje impreso por pantalla en la ventana de comandos de Matlab:

$$CORRECTO: Ventanado\ bloque\ 1, error\ máximo\ =\ 0.00e+00$$

En la Figura 34 se comparan el primer bloque de la señal creada en la implementación del preprocesador, antes y después de aplicarle la ventana. Se aprecia que una vez aplicada, reproduce las mismas oscilaciones, pero atenuadas progresivamente hacia los extremos apreciando una forma de "campana", tal y como debería pasar según se ha visto en la representación de la ventana Hann, por lo tanto, se valida que está funcionando correctamente.

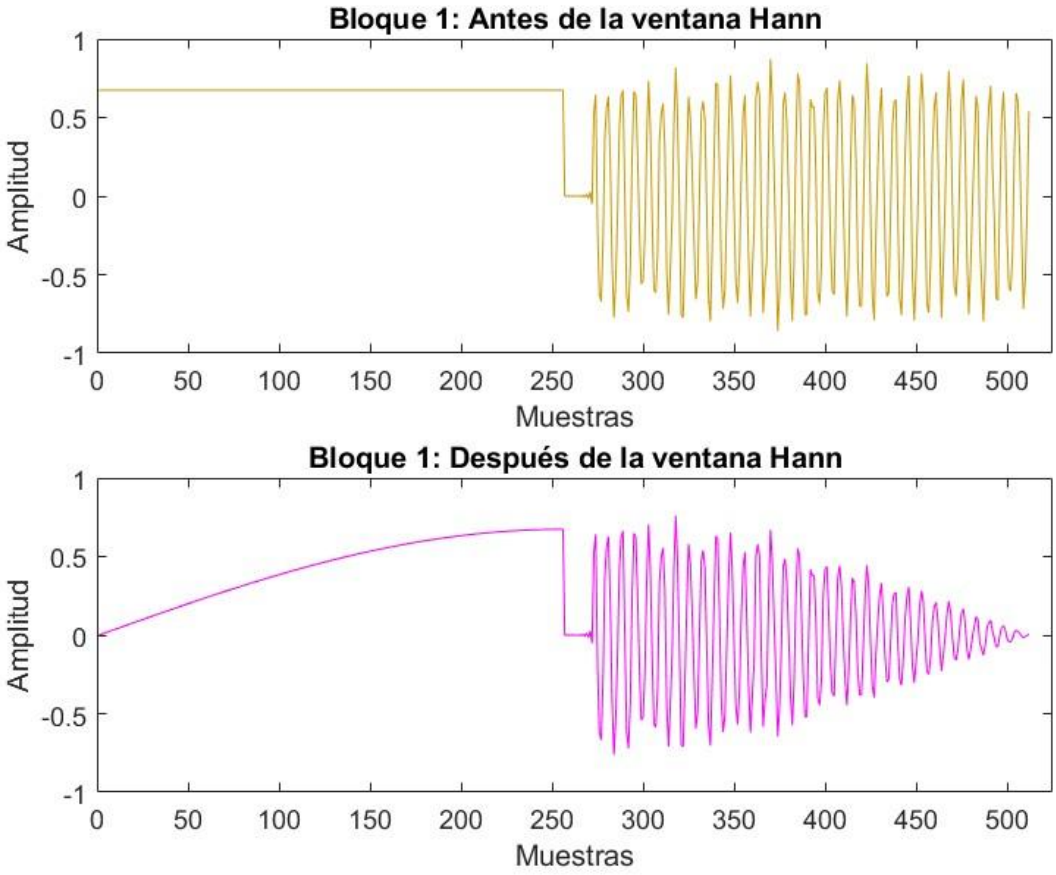


Figura 34: Bloque 1 antes VS después de aplicar la ventana Hann - SENOIDAL

Ocurre lo mismo en la representación del primer bloque de la señal cuadrada y triangular que se muestran en la Figura 35 y Figura 36 respectivamente.

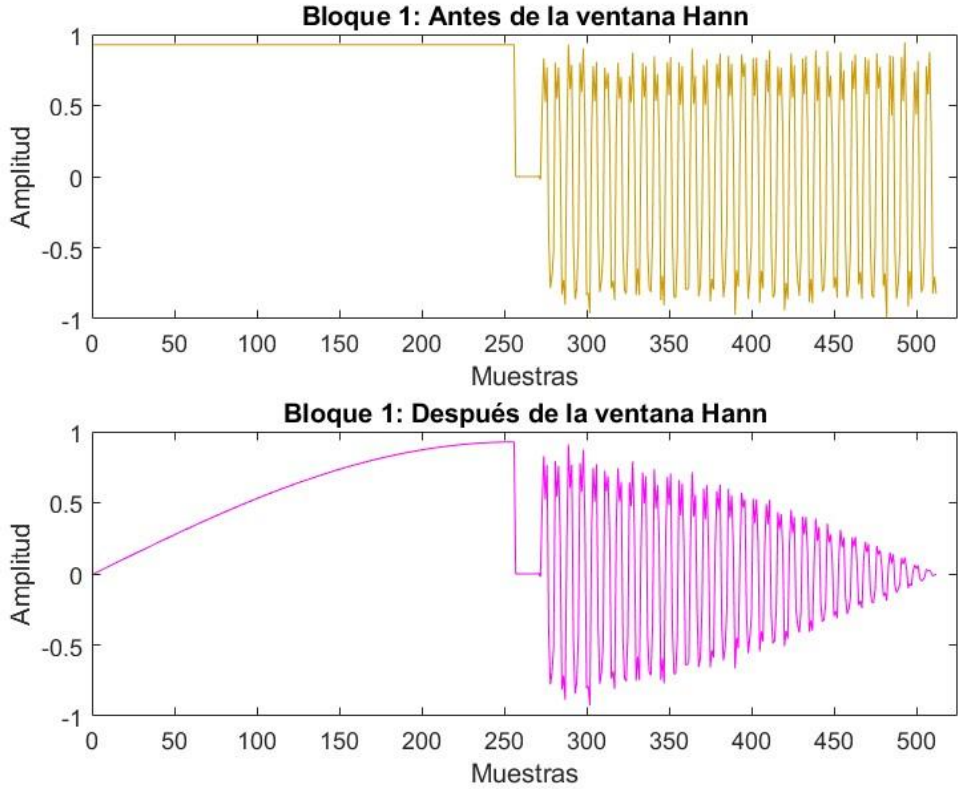


Figura 35: Bloque 1 antes VS después de aplicar la ventana Hann - CUADRADA

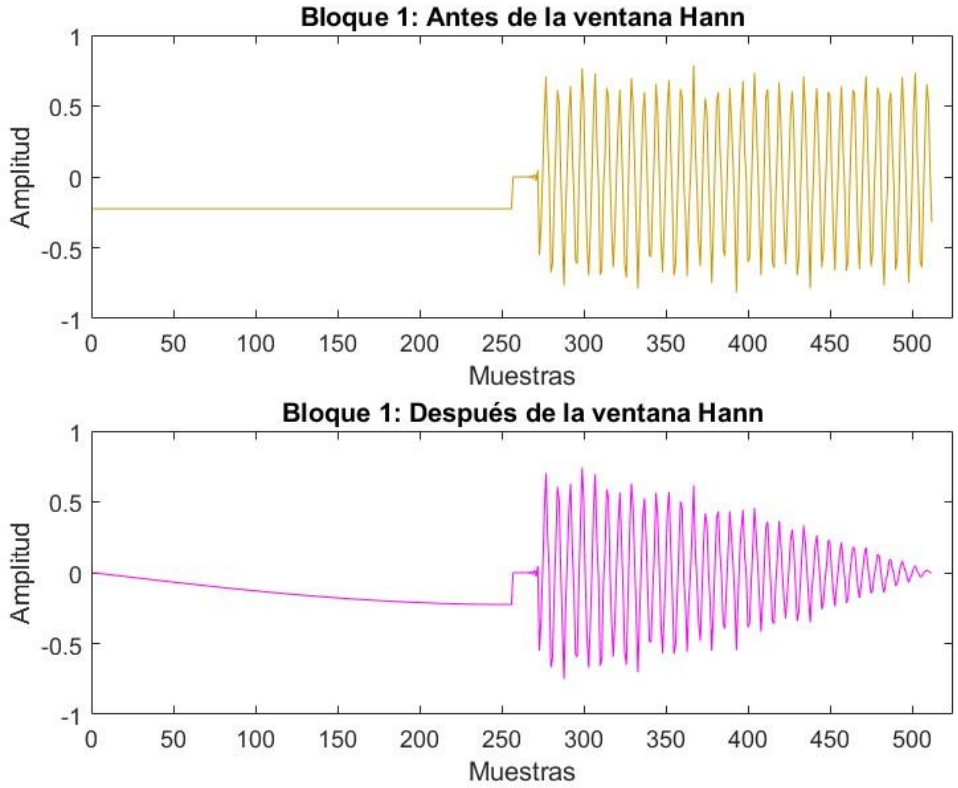


Figura 36: Bloque 1 antes VS después de aplicar la ventana Hann – TRIANGULAR

5.2.4.1.3 Después de aplicar FFT

Tras aplicar la ventana, el paso crítico, o importante, es calcular la FFT normalizada. Cualquier error aquí se traduce en un desplazamiento de energía o distorsiones espectrales que arruinarían el proceso.

Para llevarlo a cabo, como se ha hecho en los casos anteriores, se va a comparar la FFT cuantizada generada en la implementación del preprocesador, con otra teórica creada, midiendo el error máximo absoluto de sus coeficientes, de manera que la diferencia máxima entre las dos FFT, esté por debajo de la tolerancia definida. Para definir esta tolerancia (ecuación 7), se ha tenido en cuenta que en Q1.15 (punto fijo fi) cada bit equivale a $^{1}/_{(2^{15})}$ y la cuantización introduce un error de ±0.5, de forma que al normalizar la FFT este error se suma en la parte real y parte imaginaria, es decir:

$$Tolerancia\ cuantización = \ 2 \ \times \left(\frac{1}{2^{15}}\right) \ = 6.01 \times 10^{-5} \qquad (7)$$

Finalmente, el error calculado es de $2.12 \ \times \ 10^{-5}$, que como es menor que la tolerancia establecida, se confirma que la cuantización no altera demasiado la FFT, y se da por válido. Por otro lado, en la Figura 37 se representa en valor absoluto la FFT teórica con la FFT implementada, y como era de esperar por el error tan indetectable que hay, se solapan totalmente mostrando picos en las mismas componentes de frecuencia. Esto confirma que la FFT está detectando correctamente las frecuencias.

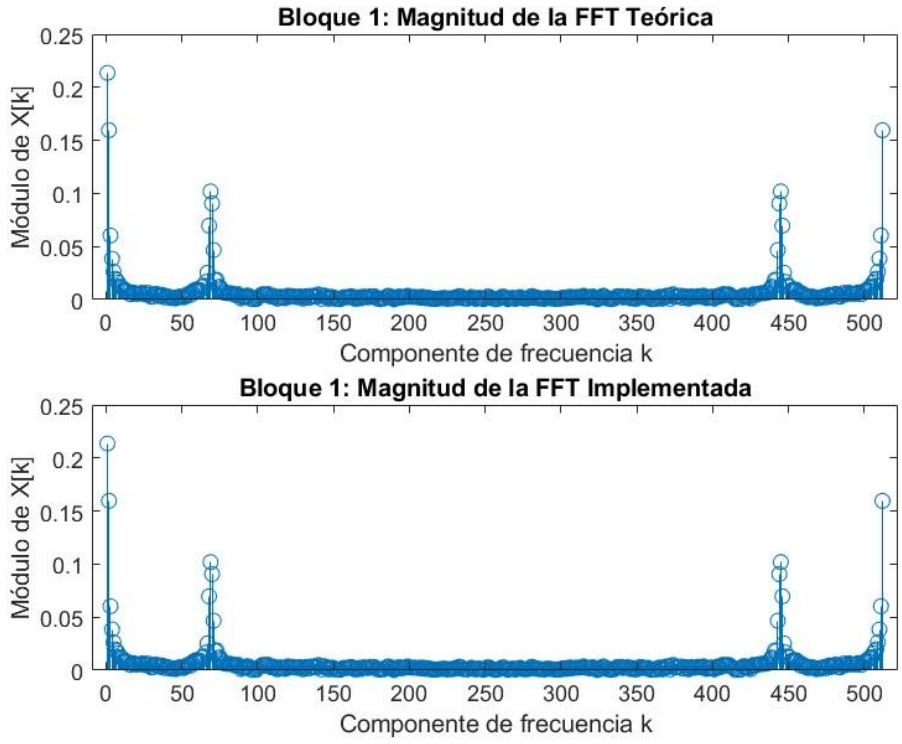


Figura 37: Comparación FFT teórica VS implementada – SENOIDAL

En cuanto a la FFT de la señal cuadrada (Figura 38) y la FFT de la señal triangular (Figura 39), corrobora lo mismo que la de la senoidal. Localiza las frecuencias correctamente, y además estas dos gráficas nos confirman otra cosa. Los armónicos pares que veíamos con más claridad en los espectros anteriores (ejemplos Figura 24 y Figura 25) han desaparecido casi por completo, confirmando que las etapas que se han hecho hasta el momento están funcionando correctamente.

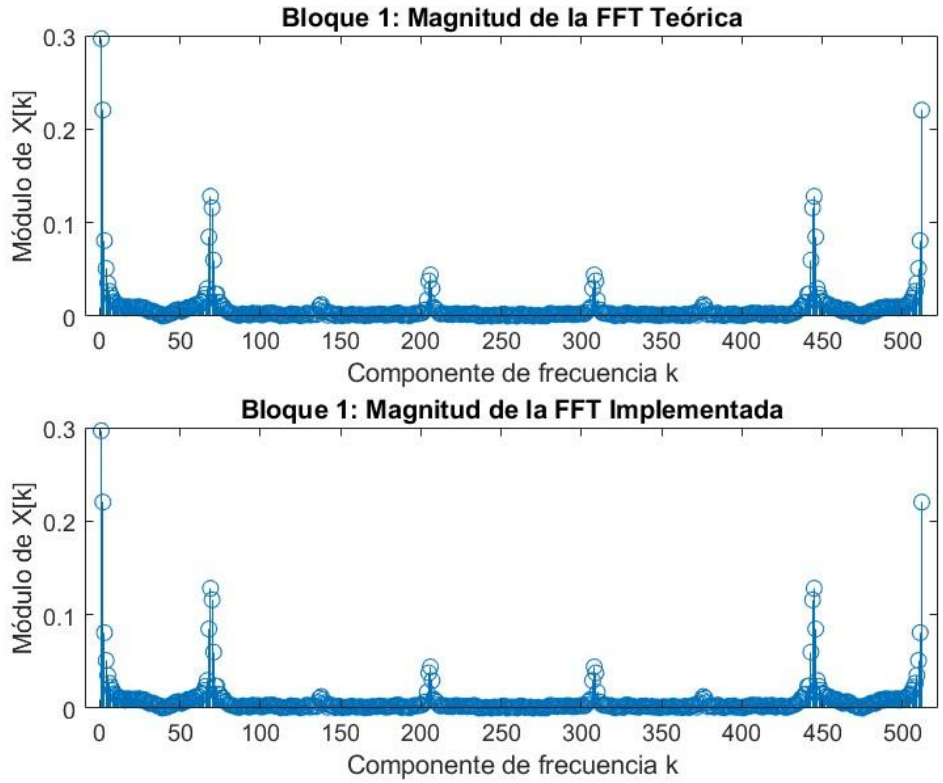


Figura 38: Comparación FFT teórica VS implementada - CUADRADA

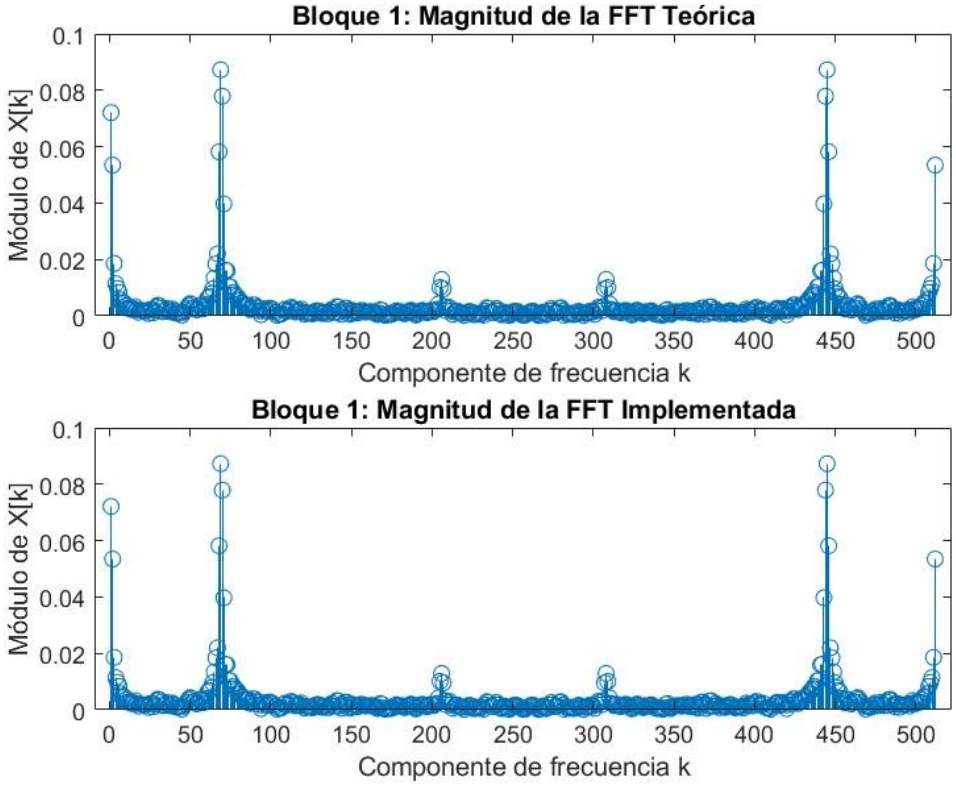


Figura 39: Comparación FFT teórica VS implementada – TRIANGULAR

5.2.5 Pruebas de integración del umbral + esparsificación + cuantización + IFFT

Ahora, este segundo bloque de las pruebas de integración de la función "procesadoEspectral", se va a centrar en verificar la etapa posiblemente más delicada de toda la implementación: la reducción de información espectral mediante la esparsificación, y la posterior cuantización. Es decir, en estas etapas se decide qué información o parte de la señal se descarta y qué parte se queda, que afecta directamente a la reconstrucción de la señal.

5.2.5.1 **Comprobaciones numéricas**

Las comprobaciones numéricas van a ser primero en relación con el umbral, la máscara (índices o coeficientes conservados) y la energía retenida, a ver si los resultados son los esperados entre los valores teóricos y los valores sacados de la implementación del preprocesador. Si se calcula mal el umbral o la máscara, se perdería demasiada información o no se reduciría nada.

Los resultados se imprimen por pantalla, y son muy satisfactorios. El umbral teórico y el implementado son exactamente el mismo, no hay error. Por otro lado, se comparará la energía que se ha conservado tras la esparsificación, con la energía establecida en los parámetros de configuración. Este cálculo se ha realizado con las fórmulas que se muestran en las ecuaciones 8 y 9, que muestran respectivamente cómo se calcula la energía de cada coeficiente y la energía total.

$$E_i = |X[i]|^2 \quad (8)$$

$$E_{total} = \sum_i E_i \quad (9)$$

Una vez calculada la energía total, se ordenan de mayor a menor y acumulándolas, se obtiene el umbral que la energía retenida alcanza. De nuevo los resultados son buenos y coinciden, los mensajes impresos por pantalla son:

$$Umbral\ y\ Energía\ retenida:$$

$$CORRECTO: Umbral\ implementado\ =\ 6.0536e-07, teórico\ =\ 6.0536e-07, error\ =\ 0.00e+00$$

$$CORRECTO: Energía\ retenida\ =\ 0.9999\ >=\ 0.9999\ Valor\ configurado$$

Después, se comprobará el error de cuantización, pero esta vez después de la esparsificación, no como en las pruebas de integración anteriores que fueron tras la FFT, aunque la tolerancia será la misma que la calculada en la ecuación 7. Se comprobará lo mismo, es decir, que el error de cuantización no supere lo permitido por el formato Q1.15. El resultado es muy bueno, aunque hay algo de error que es normal, es prácticamente nulo:

$$CORRECTO: Error\ de\ cuantización\ =\ 4.34e-19\ <\ tolerencia\ establecida\ =\ 6.10e-05$$

Por último, se verificará que la longitud del bloque tras la IFFT es la establecida en el parámetro NFFT, además de que debe ser real (no debe tener parte imaginaria), que con la función de Matlab "*isreal()*" se comprueba fácilmente. Si no se cumplen estas condiciones, no habrá coherencia en la reconstrucción. De nuevo coinciden y sí es real. Por lo tanto, se da por buena esta fase, y se garantiza una buena esparsificación y reconstrucción.

$$CORRECTO: La\ IFFT\ es\ real\ y\ su\ longitud\ =\ 512, con\ NFFT\ =\ 512$$

5.2.5.2 **Comprobaciones visuales**

A continuación, se va a representar la señal, o en este caso el primer bloque, antes y después de la esparsificación. Se muestra en la Figura 40. Además de corroborar que la esparsificación ha hecho efecto al reducirse la amplitud en algunos puntos sobre todo en los dos extremos, también confirma que la IFFT restaura la forma de onda original, ya que la superposición es prácticamente perfecta, y se sigue manteniendo la forma de "campana" comentada anteriormente que ayudará a tener una mejor reconstrucción. Se representa el primer bloque esparsificado de la señal cuadrada en la Figura 41 y de la señal triangular en la Figura 42, que también cumplen con las expectativas.

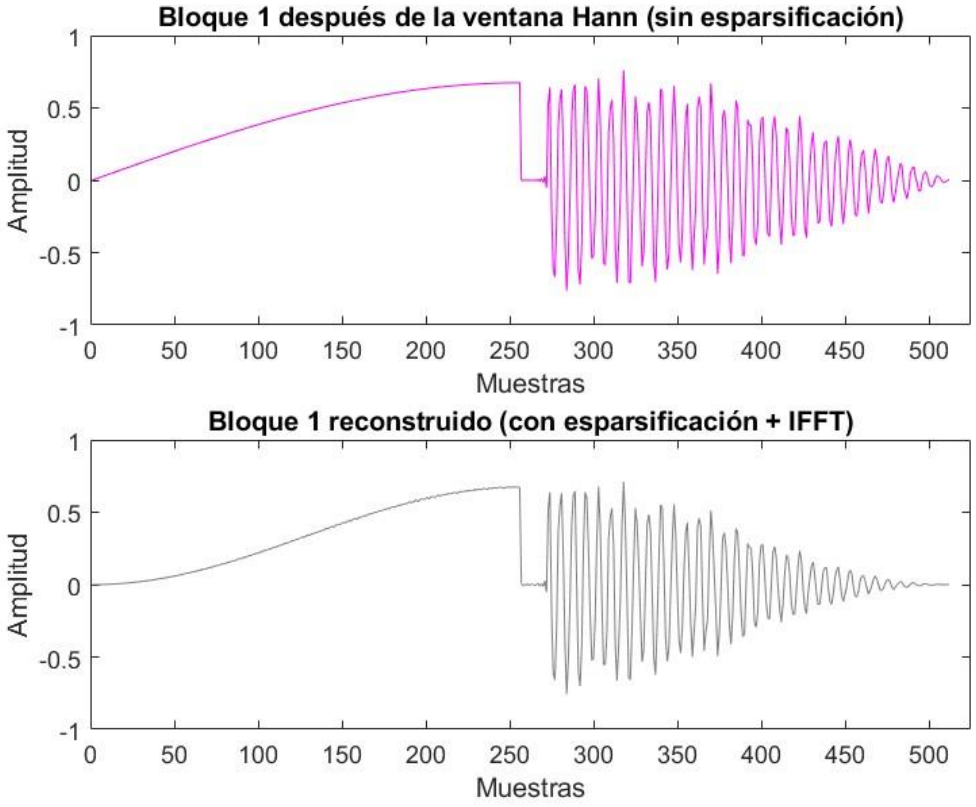


Figura 40: Bloque 1 antes VS después de la esparsificación + IFFT - SENOIDAL

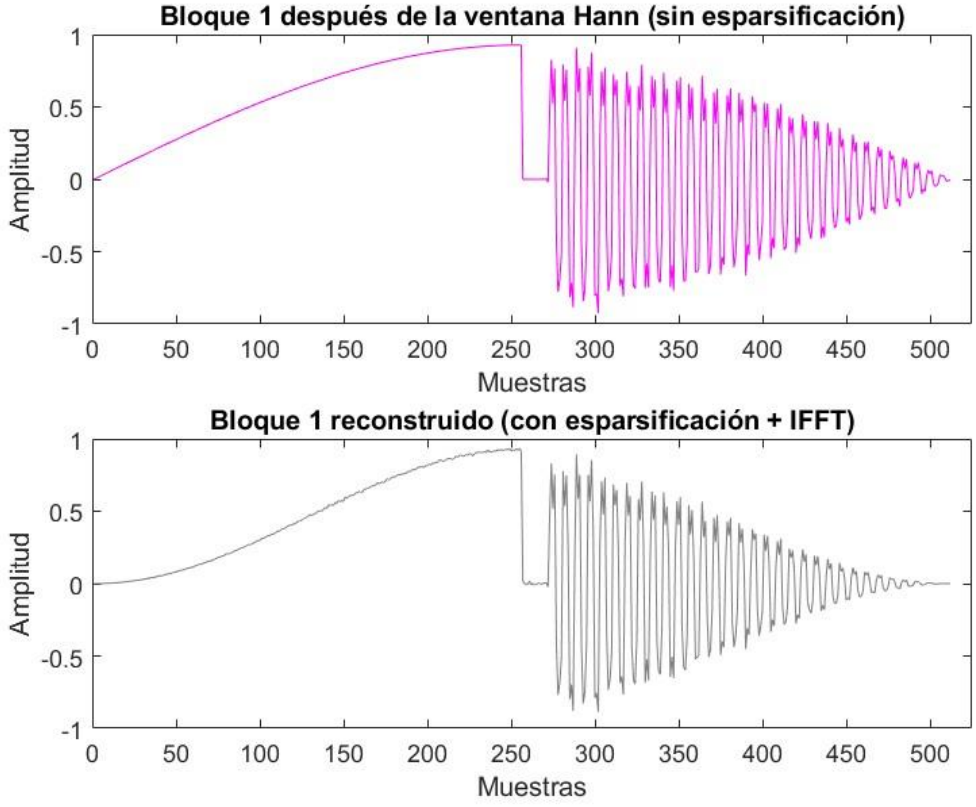


Figura 41: Bloque 1 antes VS después de la esparsificación + IFFT - CUADRADA

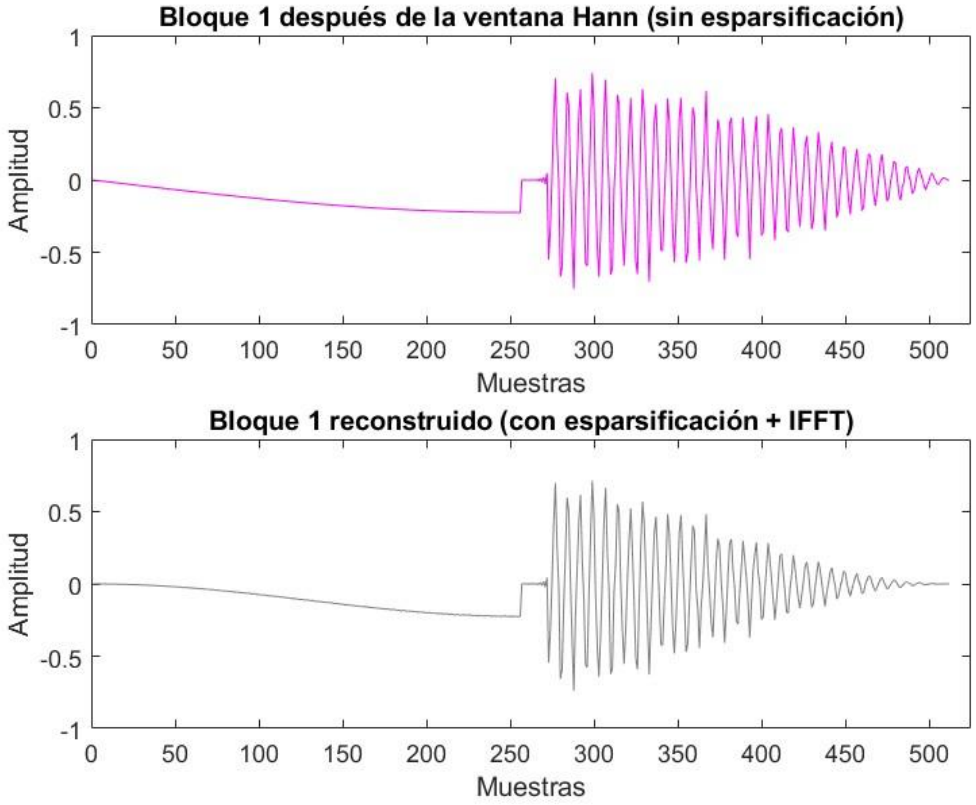


Figura 42: Bloque 1 antes VS después de la esparsificación + IFFT – TRIANGULAR

5.2.6 Pruebas de integración de Overlap-Add + reconstrucción

En esta etapa se van a realizar las últimas pruebas sobre el resultado tras la reconstrucción de la señal temporal aplicando la técnica Overlap-Add, y posteriormente recortando el exceso de muestras generado por el padding. Esta es una etapa difícil, ya que, si en alguna parte del proceso se ha producido un error o distorsión en la señal y no se ha corregido, se reflejará en la reconstrucción final.

5.2.6.1 **Comprobaciones numéricas**

Para comprobar la fidelidad de la reconstrucción, como comprobación numérica se va a calcular la correlación entre la señal decimada y la reconstruida. Este dato refleja si en esta función completa de procesado espectral, se ha mantenido la forma de onda o no. Se calcula con el coeficiente de correlación de Pearson (ecuación 10), donde X e Y son respectivamente las señales decimada y reconstruida. Esta correlación informa sobre cuanto comparten las señales la forma de onda, y el resultado debe de ser como mínimo 0.98 o lo más cercano a 1 para que se considere favorable.

$$\rho = \frac{cov(x,y)}{\sigma_x \, \sigma_y} \tag{10}$$

Siendo las variables:

$$cov(x,y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y}) \quad (11)$$

$$\sigma_x = \sqrt{var(x)} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 \quad (12)$$

La ecuación 12 sería igual con la señal Y. Después de realizar el cálculo, el resultado que se imprime por pantalla es que la correlación es de 1.00 lo que es un resultado altamente satisfactorio.

En segundo lugar, se ha calculado la fracción de energía reconstruida respecto a la señal decimada, es decir, que no se haya perdido información relevante por errores acumulados durante todo el procesado espectral. Se hace con la ecuación 13.

$$frac.energía = \frac{\sum y^2_{reconstruida}}{\sum y^2_{decimada}} \quad (13)$$

El valor esperado de esta operación debe ser cercano a uno, normalmente entre 0.95 y 1. En este caso, después de realizar el cálculo, el resultado es de 0.998, es decir, muy cercano al valor establecido de 0.9999 lo que indica que el proceso ha funcionado correctamente, y sobre todo indica una buena reconstrucción.

Por último, se ha comparado un dato básico pero obligatorio, que es la longitud de la señal reconstruida y la señal original con ruido. La longitud es un dato que únicamente se modifica en la decimación, por lo que si a la señal original se le aplica el valor de decimación D=2, las dos longitudes deben coincidir. Después del cálculo efectivamente las longitudes coinciden.

Los mensajes impresos por pantalla en estas comprobaciones son los siguientes:

$CORRECTO: Longitud\ señal\ generada\ original: 750, reconstruida: 750$

$CORECTO: Energía\ reconstruida\ = 0.998\ (fracción\ respecto\ a\ decimada)$

$CORRECTO: Correlación\ entre\ señal\ decimada\ y\ reconstruida\ = 1.00$

5.2.6.2 **Comprobaciones visuales**

Como últimas representaciones visuales, se van a mostrar dos gráficas, las dos con el objetivo de confirmar que la forma general de la señal se mantiene, tanto en el dominio del tiempo como en el de la frecuencia, de cada tipo de señal.

Como se ha comentado más veces, se mantiene el color representativo de las señales anteriores para facilitar al lector identificar las señales al momento. En la Figura 43 se muestra el espectro en escala logarítmica de las señales decimada y reconstruida senoidal, en la Figura 44 de la señal cuadrada y en la Figura 45 de la señal triangular. En todas las gráficas se verifica que la distribución espectral original de la señal se conserva, lo que indica que el preprocesado y la reconstrucción son correctas, y cumplen el requisito **R.3** establecido en la Tabla 5 del catálogo de requisitos.

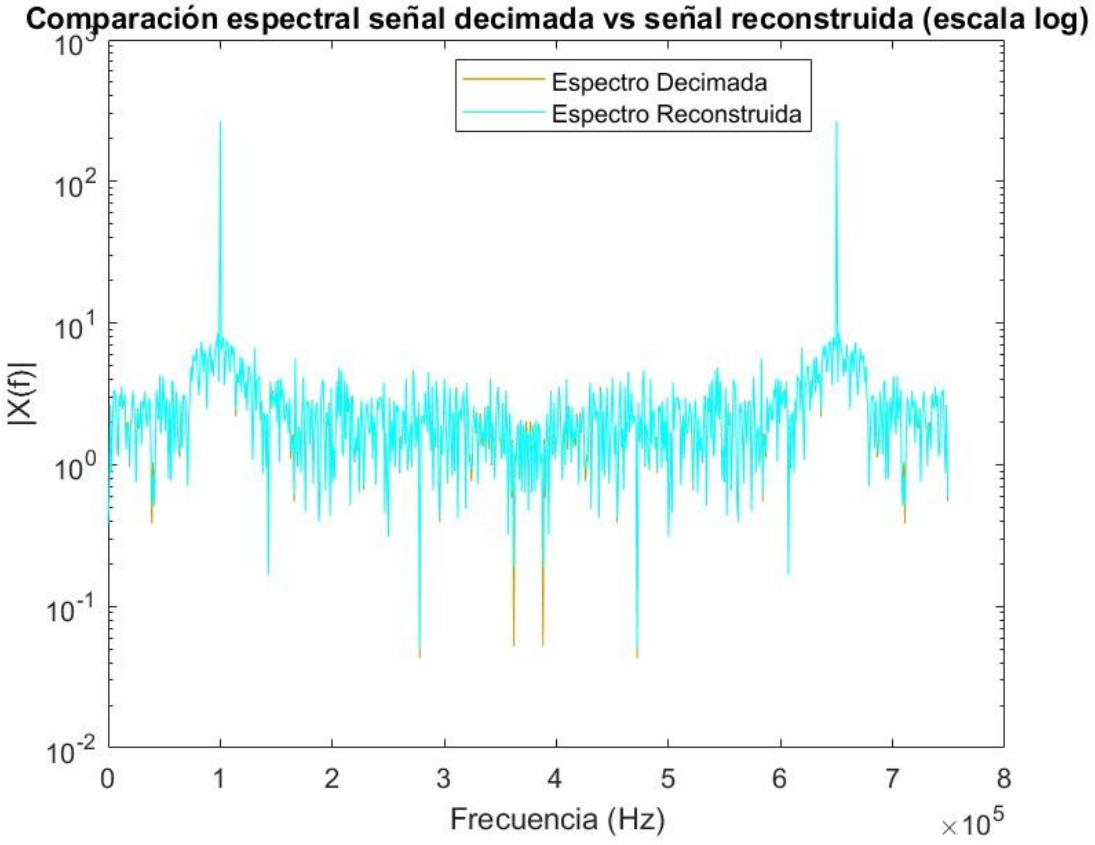


Figura 43: Espectro señal decimada VS reconstruida - SENOIDAL

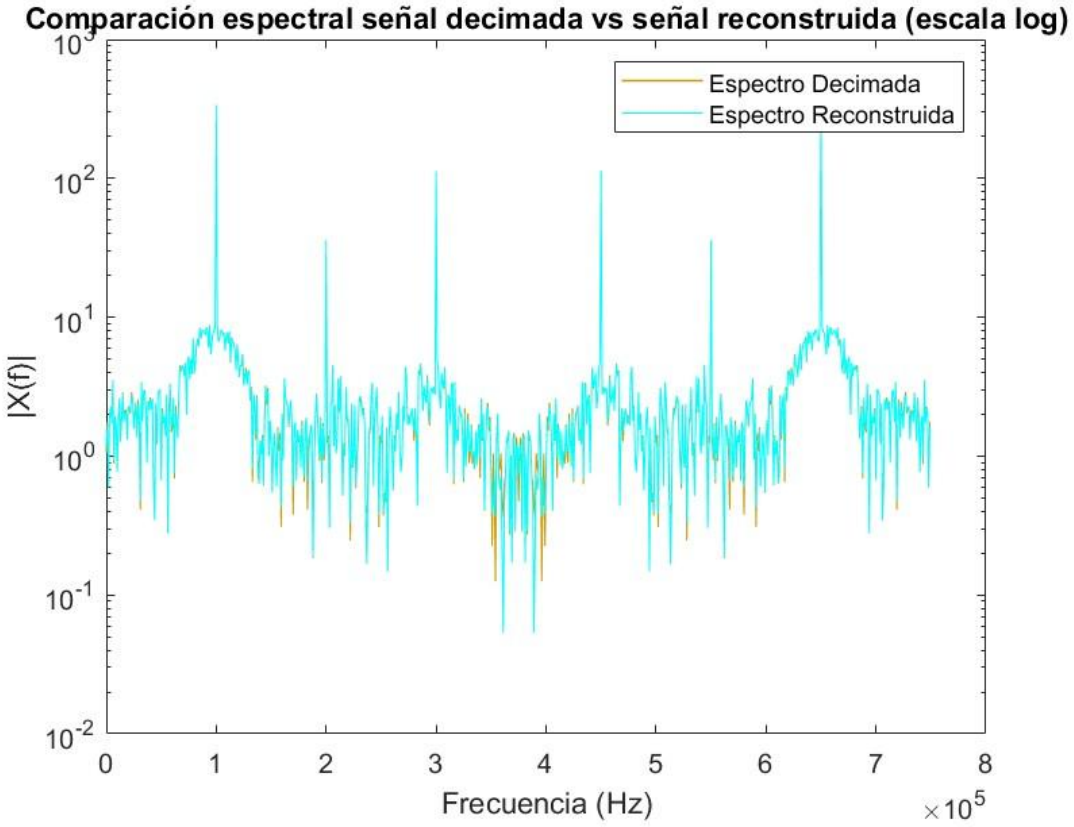


Figura 44: Espectro señal decimada VS reconstruida - CUADRADA

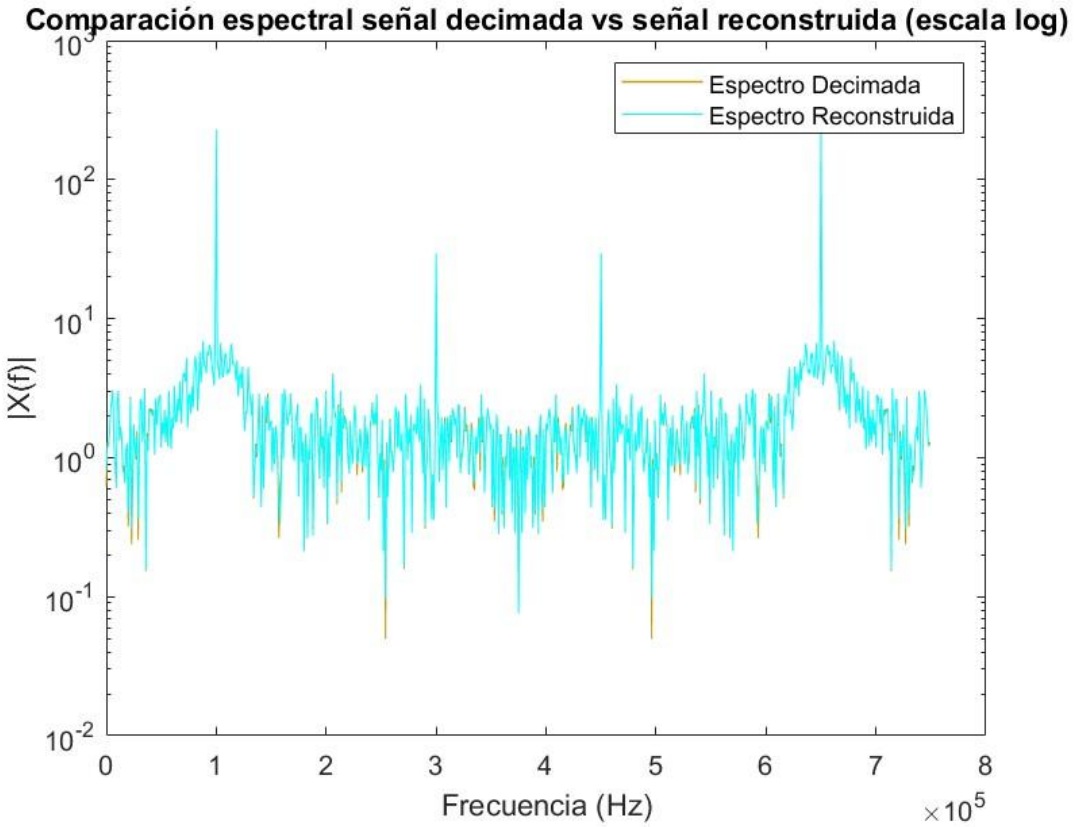


Figura 45: Espectro señal decimada VS reconstruida - TRIANGULAR

Por último, la gráfica prácticamente más relevante, en la que se muestra el resultado final de todo el proceso que se ha implementado, se representa en la Figura 46. Esta gráfica representa la señal original, la generada en Matlab con ruido, respecto a la reconstruida tras esta última función procesado espectral. Como se ha dicho en varias ocasiones, algunas etapas que ha sufrido la señal introducen cierto retardo, por lo tanto, es normal que se aprecie algo de retardo, pero usando el mismo método que se explicó en la sección 4.7.1 Propuesta del diseño digital, se ha conseguido casi eliminar por completo el retardo. Como se ve en el eje X, la escala está en $10^{-5}s$, por lo que la diferencia entre dos picos respectivos de las señales es aproximadamente de $1.36\ \mu s$, que es tan pequeño que en tiempo real sería imperceptible.

La señal reconstruida además presenta escalones, que también se explicó durante la sección 4.7.1 Propuesta del diseño digital, este efecto es totalmente normal al tratarse de una señal digital que ha sido procesada.

Por otro lado, se observa una clara diferencia de amplitud, una señal sin ruido y prácticamente idéntica en cuanto a geometría con la original. Esto se traduce en una gran reducción en la carga computacional del procesador que, para el posible trabajo futuro a la hora de integrarlo en una FPGA, será una gran ventaja. Estos resultados cumplen con los requisitos **R.1, R.2** y **R.4** del catálogo de requisitos establecido en la Tabla 5 en el apartado 3.2 Catálogo de requisitos del proyecto.

Ocurre lo mismo con las señales cuadrada y triangular, que se representan en la Figura 47 y Figura 48 respectivamente. También se ha logrado una reconstrucción y una reducción de datos extraordinaria, y de igual manera que en la senoidal, presenta los escalones propios de una señal digital.

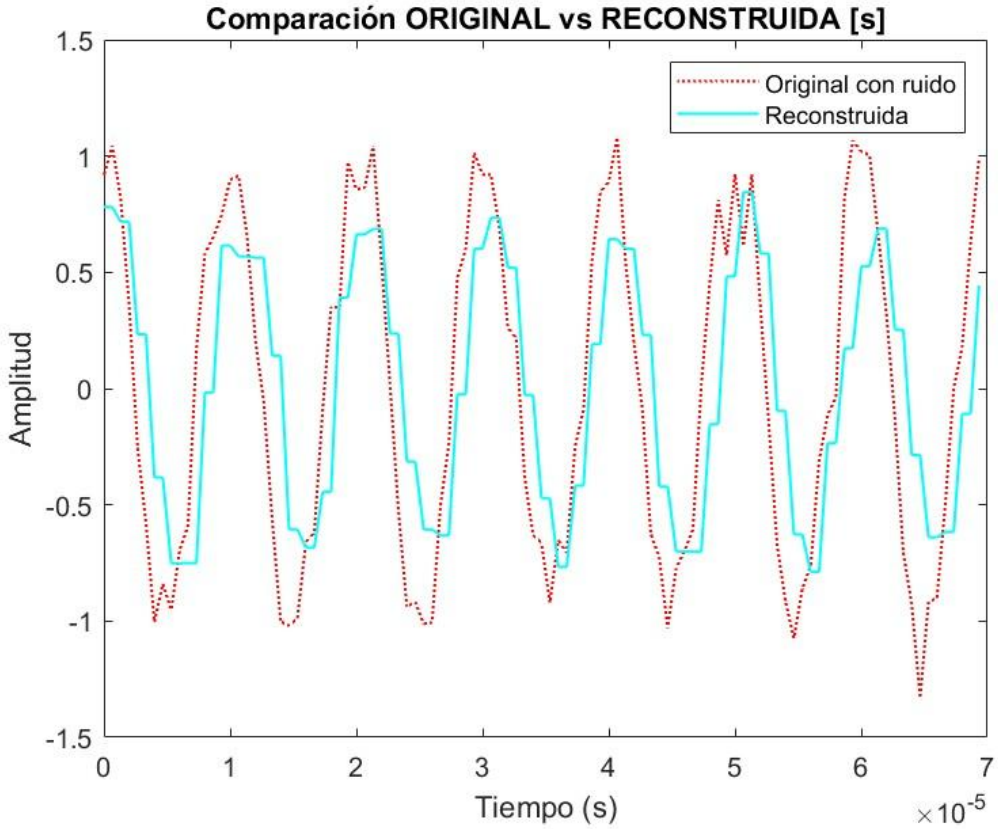


Figura 46: Representación FINAL: Original con ruido VS reconstruida – SENOIDAL

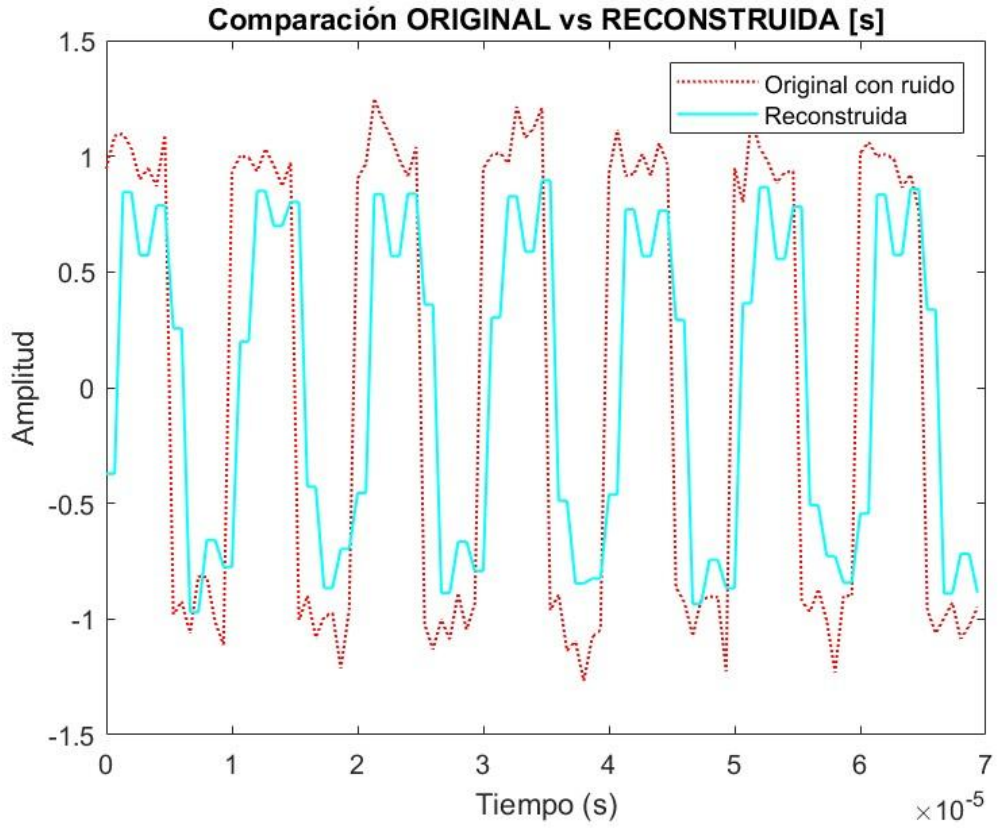


Figura 47: Representación FINAL: Original con ruido VS reconstruida – CUADRADA

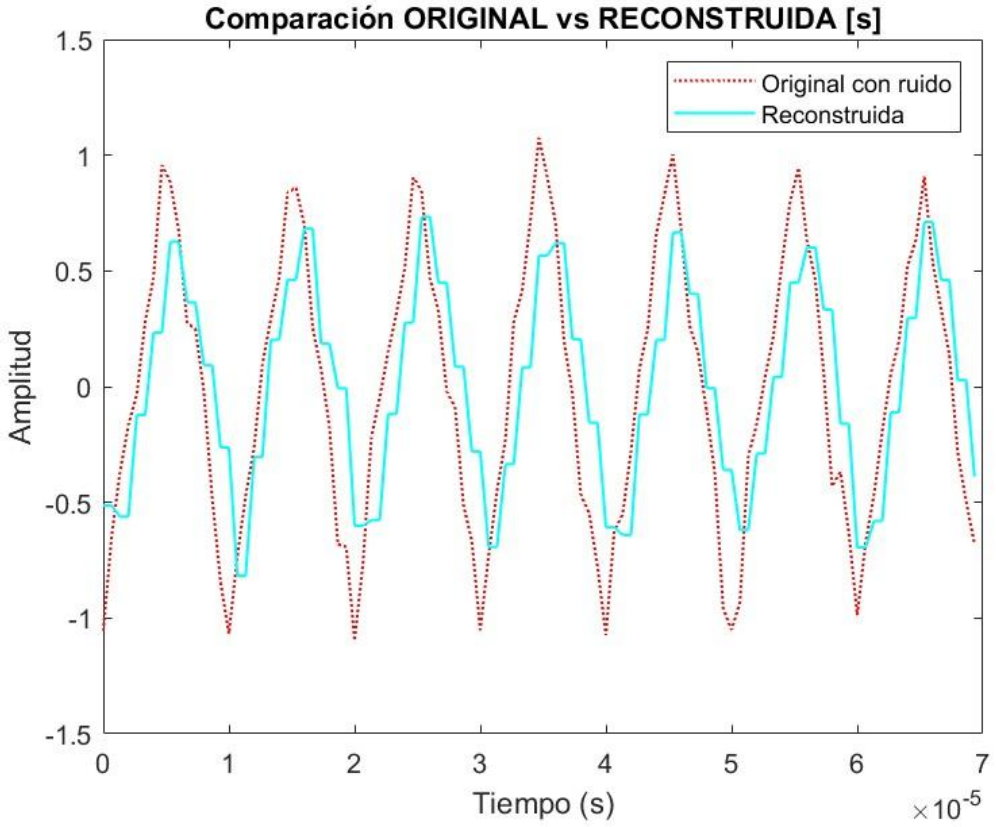


Figura 48: Representación FINAL: Original con ruido VS reconstruida – TRIANGULAR

5.2.7 Resultados de las métricas de calidad

En este último apartado, se van a mostrar y comentar los resultados obtenidos en las métricas de calidad explicadas en la descripción de la propuesta. La MSE y SNR, que se calculan con las ecuaciones 1 y 2. Los resultados de las tres señales se muestran en la Tabla 8.

Tabla 8: Resultados de las métricas de calidad

	Senoidal	***Cuadrada***	***Triangular***
MSE	$\mathbf{1.2402 \times 10^{-5}}$	$\mathbf{2.3102 \times 10^{-5}}$	$\mathbf{6.2262 \times 10^{-6}}$
SNR (dB)	**43.27**	**43.06**	**44.11**

Como se explicó anteriormente, lo ideal es que la SNR sea lo más alta posible, lo ideal es que sea mayor de 40dB, y la MSE lo más baja posible, lo más cercano a 0.

Viendo los resultados está claro que, aunque todos son buenos, como se venía diciendo durante las pruebas de integración los mejores son los de la señal triangular. Tiene el mejor

compromiso entre fidelidad y procesamiento con la señal original. Esto es debido a que además de eliminar eficazmente el ruido, y deja pasar la banda que realmente interesa, provocando un error minúsculo (MSE) y una SNR de casi 44 dB, lo cual habla reconstrucción muy fiel.

La señal senoidal, pese a ser la más limpia de la serie, solo tiene una frecuencia, por lo que su precisión queda limitada por el nivel de ruido antes del filtrado, y consigue una reconstrucción también muy buena.

Por último, la señal cuadrada, aunque los resultados son muy buenos, es la peor de las tres. Sufre más que las otras debido a su naturaleza, los cambios abruptos generan mucha energía. Al aplicar el filtro FIR gran parte de esa energía desaparece, que pare recuperarla se deberían de cuantizar muchísimos coeficientes que añadiría aún más ruido. Así, la reconstrucción tiene menos detalle que el resto, aunque sigue siendo buena, y un MSE más alto que el resto.

5.3 Cumplimiento de requisitos

Se exponen en la Tabla 9 en este capítulo, los requisitos establecidos en la sección 3.2 Catálogo de requisitos del proyecto que finalmente cumple el sistema de preprocesado implementado. Su verificación se ha ido confirmando durante el desarrollo del manuscrito.

Tabla 9: Cumplimiento de requisitos del proyecto

Requisito	**Identificador**	**Cumplimiento**
Reducción de datos	R.1	Sí (sección 5.2.6)
Mantener la geometría de la señal original	R.2	Sí (sección 5.2.6)
Mantener la frecuencia de la señal original	R.3	Sí (sección 5.2.6)
Reconstrucción sin imperfecciones	R.4	Sí (sección 5.2.6)
Escalabilidad a FPGA	R.5	Sí (sección 4.1 y 4.7.1)

6. Presupuesto

Se expone a continuación en la Tabla 10: Presupuesto total proyecto el presupuesto estimado del proyecto. El cálculo se ha basado en los siguientes factores:

- Un ingeniero cobra por hora de mano de obra una media de 21€ [41]. Si se multiplica por el total de horas que abarca el proyecto:

$$21 €/_h \times 360h = 7560 \text{ €} \tag{14}$$

- La licencia del software Matlab cuesta 700 €/año, y aunque el proyecto abarque un tiempo de 4 meses, la modalidad del pago es anual. En cambio, la licencia de Microsoft 365, que aporta herramientas como Word que es la que se ha usado para redactar el proyecto, es de 10 €/mes, que multiplicado por los 4 meses de proyecto hacen 40€.

- El hardware del PC como teclado, ratón, teclado, pantalla, y ordenador portátil. Para este conjunto de elementos se estima un precio aproximado de 1500 €.

Tabla 10: Presupuesto total proyecto

	Gastos (€)	***Fuente de financiación***
Mano de obra de Ingeniero	7.560	Inetum
Licencias de software		
Matlab	700	UPM
Microsoft 365	40	UPM
Hardware PC	1500	Inetum
Gastos totales	9800	

7. Impacto del proyecto

El desarrollo y la implementación de un preprocesador de EW tiene un impacto claro y transversal. Este capítulo presenta una reflexión sobre las implicaciones que tiene dicha implementación y su potencial contribución a los ODS (Objetivos de Desarrollo Sostenible).

7.1 Implicaciones sociales y de seguridad

Como se mencionó en el aparato de 2.1 Necesidad Social y Tecnológica, la implementación de este preprocesador va a mejorar sobre todo el ámbito de la seguridad operacional, ya que contribuye a detectar y eliminar interferencias, ruido y emisiones que puedan resultar una amenaza, y de esta manera proteger infraestructuras como redes de comunicaciones o de transporte.

Además, al automatizar el preprocesado, se minimiza la exposición del personal a entornos o situaciones peligrosas, como entornos con gran radiación, protegiendo la salud de los trabajadores. Es decir, también tiene un impacto positivo en el ámbito social.

7.2 Implicaciones Ambientales

El proyecto también aporta ventajas en el ámbito ambiental. Gracias a que se eliminan datos redundantes y se reduce la cantidad de información, la carga computacional y el consumo energético disminuye considerablemente durante el procesamiento.

7.3 Implicaciones Económicas.

Lo comentado en el punto anterior sobre las implicaciones ambientales, trae como consecuencia directa un ahorro en el gasto de los sistemas integrados militares e industriales.

Por otro lado, gracias a las ventajas que aporta la compatibilidad con HDL, en un futuro se podría escalar su fabricación a nivel nacional, o en el mejor de los casos internacional, lo que supondría un claro aumento de las ganancias.

7.4 Implicaciones Tecnológicas e Industriales

El diseño desarrollado e implementado con compatibilidad HDL, genera una base técnica sólida para futuras mejoras, y para su posible integración en distintos sectores no solo en defensa, como las telecomunicaciones, ciberseguridad, etc, impulsando el I+D (Investigación y Desarrollo).

Un ejemplo de una mejora que aportaría un gran valor extra al preprocesador es la integración de la IA. La IA cada vez cobra más importancia en el mundo, en todos los ámbitos, y un preprocesador de EW con IA integrada, es un proyecto muy ambicioso que sin duda contribuiría a su expansión.

7.5 Contribución a los Objetivos de Desarrollo Sostenible

El proyecto puede alinearse con varios ODS [42] establecidos por la ONU (Organización de las Naciones Unidas).

- ODS 8: Trabajo decente y crecimiento económico.

El desarrollo, implementación y mantenimiento de sistemas como el preprocesador, contribuye a la creación de puestos de trabajo especializados y potenciando la competitividad del sector tecnológico.

- ODS 9: Industria, innovación e infraestructura

Como se ha comentado en las Implicaciones Tecnológicas e Industriales, el diseño implementado impulsa la creación de sistema inteligentes.

- ODS 12: Producción y consumo responsable

El proyecto optimiza los recursos computacionales, y promueve la eficiencia de datos y energía, como se ha comentado en las Implicaciones Ambientales.

8. Conclusiones y Trabajo Futuro

Este capítulo presenta las reflexiones finales sobre los resultados del proyecto, destacando los aspectos interesantes encontrados durante el desarrollo. Además, se proponen dos líneas futuras de trabajo, explicando el motivo y su funcionalidad.

8.1 Conclusiones

Este Trabajo de Fin de Grado ha permitido desarrollar un diseño completo y funcional de un preprocesador de señales, orientado a aplicaciones de EW. Se ha logrado una de las principales metas: ofrecer una propuesta independiente, bien estructurada y con potencial real para ser integrada en sistemas más amplios. No se trata simplemente de un proyecto académico, sino de una solución que, aunque en fase inicial, se ha concebido con criterio técnico, visión a futuro y posibilidad de crecimiento.

Entre las aportaciones destacables, se debe mencionar el hecho de haber seguido fielmente una arquitectura modular, que permite separar claramente las etapas de calibración, filtrado y compresión, facilitando tanto su análisis individual, como su implementación en hardware específico, como una FPGA. Esta independencia no solo mejora la compresión del sistema, sino que también facilita la escalabilidad del sistema a otros entornos de trabajo.

Cabe destacar que, durante el desarrollo, ha habido varios aspectos llamativos. Uno de ellos ha sido, sin duda, la enorme variedad de algoritmos disponibles para la reducción y compresión de datos. No solo por el número en sí, sino por la cantidad de variaciones, combinaciones, enfoques, y casos de uso, que se pueden hacer, abriendo las puertas a multitud de proyectos. Del mismo modo, la facilidad que ofrece Matlab en cuanto a compatibilidad y escalabilidad con otros sistemas, no pasa desapercibida. Transmite una sólida fiabilidad que refuerza su papel como una herramienta de referencia tanto para desarrollar prototipos, como proyectos más avanzados.

Este trabajo pone de manifiesto que los conocimientos adquiridos a lo largo del grado, resultan fundamentales para abordar problemas de ingeniería reales. Una buena base técnica y un enfoque estructurado, contribuyen a resolver los retos que se presentan día a día en las nuevas tecnologías.

8.2 Trabajos futuros

Este capítulo aborda las principales líneas de trabajo futuras de este proyecto, que son la adaptación del preprocesador para operar con señales de radio frecuencia reales y la conversión a código HDL.

8.2.1 Ampliación para trabajar con señales RF reales

Como se explicó en la sección de la Introducción 1.1.1 Proyecto real de referencia, para trabajar con señales reales y probar la eficacia del sistema de preprocesado de este proyecto, es necesario implementar dos etapas extra.

Por un lado, la integración de una etapa que amplifique la potencia, para ajustar el nivel de la señal recibida, y asegurar una amplitud óptima para su posterior digitalización. Esta etapa es fundamental cuando se trabaja con señales de baja potencia, típicas en escenarios de EW, donde la señal útil puede verse afectada por ruido o sufrir atenuaciones durante la transmisión.

Y por otro lado, la inclusión de un conversor analógico – digital (ADC) es imprescindible para convertir la señal a digital, y que sea apta para ser procesada por el preprocesador. Las características del ADC dependerán de la señal de interés, ya que deben estar alineadas.

La integración de estas etapas permitirá validar el funcionamiento del preprocesador en situaciones reales, abriendo la puerta a pruebas de campo.

8.2.2 Conversión a HDL e integración en FPGA

Como se ha explicado en la sección 4.7.2 Conversión a HDL, hay dos herramientas posibles para realizar la conversión de código a HDL: HDL Coder de Matlab o HDL Workflow Advisor de Simulink.

Aunque se han tomado decisiones para facilitar la conversión, como la estructura, y el acceso y tipo de las variables a punto fijo Q1.15, la conversión directa no siempre es trivial. Hay determinadas operaciones presentes en el algoritmo actual, como las divisiones aritméticas, o la creación de estructuras como “config” creada para simplificar el código, que no pueden ser directamente soportadas en este tipo de hardware. Por ello, sería necesario realizar una revisión y adaptación del código, sustituyendo aquellas partes que no sean compatibles, sin alterar el resultado final ni el algoritmo.

Para hacerlo con HDL Coder basta con escribir en la ventana de comandos “hdlcoder”, y se abrirá una ventana en la que se debe añadir: El script que se desea convertir, y un script nuevo creado en el que se implemente el test que se quiera realizar para comprobar si se ha realizado bien la conversión.

En cambio, en HDL Workflow Advisor, gracias a que Simulink ofrece bloques ya creados compatibles de código HDL, como por ejemplo el filtro FIR, sería algo más sencillo. Consistiría en elegir correctamente los bloques HDL (ya que hay varias opciones de cada uno en función de las necesidades) y configurar los parámetros necesarios. Después, implementar el código correspondiente a las partes del código que no dispongan de bloques ya creados, y probar el funcionamiento.

Por otra parte, una vez realizada la conversión y comprobado el correcto funcionamiento, se podría llevar a cabo la integración del preprocesador en una FPGA. Para ello, los pasos que se deberían realizar, a grandes rasgos, serían: En primer lugar, importar el código HDL generado al entorno de desarrollo de la FPGA elegido, por ejemplo, Quartus para Intel. En segundo lugar, programar la FPGA con el diseño sintetizado, configurar el entorno de desarrollo hardware (como los módulos, buffers, pines, etc). Por último, conectar la placa FPGA al ordenador, probar el funcionamiento y corregir los posibles errores que pueda haber.

9. Referencias

[1] O. Rodriguez, «OTAN Inteligencia Artificial: Palantir y el Futuro del Combate Moderno», Virtuabarcelona.com. Accedido: 30 de mayo de 2025. [En línea]. Disponible en: https://virtuabarcelona.com/2025/04/15/otan-inteligencia-artificial-palantir-y-el-futuro-del-combate-moderno/

[2] «ESPAÑA», Inetum. Accedido: 30 de mayo de 2025. [En línea]. Disponible en: https://www.inetum.com/es/espana

[3] R. D. InfoDefensa, «J. Torres (Inetum): "Apostamos por una digitalización «ad hoc» para las Fuerzas Armadas españolas"», Infodefensa - Noticias de defensa, industria, seguridad, armamento, ejércitos y tecnología de la defensa. Accedido: 30 de marzo de 2025. [En línea]. Disponible en: https://www.infodefensa.com/texto-diario/mostrar/3763146/-j-torres-inetum-apostamos-digitalizacion-ad-hoc-fuerzas-armadas-espanolas

[4] Á. M. Serrador, «▷ Tipos de Pruebas de Software: Unitarios vs. Integración vs. End-to-End», Yeeply. Accedido: 30 de abril de 2025. [En línea]. Disponible en: https://yeeply.com/blog/digitalizacion/tipos-pruebas-software-test-unitarios-vs-integration-test-vs-test-end-to-end-e2e/

[5] «Guerra electrónica: el campo de batalla silencioso del futuro», Grupo Oesía. Accedido: 19 de marzo de 2025. [En línea]. Disponible en: https://grupooesia.com/insight/guerra-electronica-el-campo-de-batalla-silencioso-del-futuro/

[6] R. D. InfoDefensa, «El Ejército diseña un plan para renovar sus medios de guerra electrónica tras la guerra en Ucrania», Infodefensa - Noticias de defensa, industria, seguridad, armamento, ejércitos y tecnología de la defensa. Accedido: 19 de marzo de 2025. [En línea]. Disponible en: https://www.infodefensa.com/texto-diario/mostrar/4157819/ejercito-disena-plan-renovar-medios-guerra-electronica-guerra-ucrania

[7] A. Lacasa, «Del dial-up al 5G: la evolución histórica de la banda ancha - Blog de Informática», Grado de Inge Informática. Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://blogs.udima.es/ingenieria-informatica/del-dial-up-al-5g-la-evolucion-historica-de-la-banda-ancha/

[8] M. G. Sanchez, «▷ Avances en Tecnología de Guerra Electrónica [2025]», Militar y Arsenal. Accedido: 19 de marzo de 2025. [En línea]. Disponible en: https://dudasytextos.com/militar/blog/tecnologia-en-la-guerra-electronica/

[9] V. Fernandes, G. Carvalho, V. Pereira, y J. Bernardino, «Analyzing Data Reduction Techniques: An Experimental Perspective», *Applied Sciences*, vol. 14, n.º 8, Art. n.º 8, ene. 2024, doi: 10.3390/app14083436.

[10] Defensa.com, «Thales España celebra los 25 años de la llegada de las radios digitales a las Fuerzas armadas españolas-noticia defensa.com - Noticias Defensa España», Defensa.com. Accedido: 19 de marzo de 2025. [En línea]. Disponible en: https://www.defensa.com/espana/thales-espana-celebra-25-anos-llegada-radios-digitales-fuerzas

[11] «Radioteléfono PR4G - Ejército de tierra». Accedido: 19 de marzo de 2025. [En línea]. Disponible en: https://ejercito.defensa.gob.es/materiales/transmisiones/Radiotelefono.html

[12] Defensa.com, «La nueva radio del Mando de Operaciones Especiales del Ejército de Tierra-noticia defensa.com - Noticias Defensa España», Defensa.com. Accedido: 19 de marzo de 2025. [En línea]. Disponible en: https://www.defensa.com/espana/nueva-radio-mando-operaciones-especiales-ejercito-tierra

[13] «Falcon III® RF-7850S SPR™ Advanced Wideband Secure Personal Radio | L3Harris® Fast. Forward.» Accedido: 19 de marzo de 2025. [En línea]. Disponible en: https://www.l3harris.com/all-capabilities/falcon-iii-rf-7850s-spr-advanced-wideband-secure-personal-radio

[14] «¿Qué es inteligencia electrónica? - Términos y definiciones de ciberseguridad». Accedido: 29 de mayo de 2025. [En línea]. Disponible en: https://www.vpnunlimited.com/es/help/cybersecurity/electronic-intelligence?srsltid=AfmBOor60Go23Z0i-ib7WELkNi18jdt2fKM637xfPPI9805i2FrQT6TG

[15] «https://www.defenceconnect.com.au/pdfs/DC_Multi-Domain_SE_V11_LR_compressed.pdf». Accedido: 29 de junio de 2025. [En línea]. Disponible en: https://www.defenceconnect.com.au/pdfs/DC_Multi-Domain_SE_V11_LR_compressed.pdf

[16] R. & S. G. & C. KG, «Rohde & Schwarz launches high-performance ELINT receiver». Accedido: 29 de abril de 2025. [En línea]. Disponible en: https://www.rohde-schwarz.com/us/about/news-press/all-news/rohde-schwarz-launches-high-performance-elint-receiver-press-release-detailpage_229356-918848.html

[17] «Datasheet-5950-3U-VPX-fpga-ad-da-board.pdf». Accedido: 29 de junio de 2025. [En línea]. Disponible en: https://www.mrcy.com/application/files/9017/0291/0242/Datasheet-5950-3U-VPX-fpga-ad-da-board.pdf

[18] «deliver.pdf». Accedido: 29 de junio de 2025. [En línea]. Disponible en: https://www.pentek.com/deliver/deliver.cfm?DI=3&FN=5950_062218.pdf

[19] «COTS - Commercial Off The Shelf Software - Dazzet». Accedido: 19 de mayo de 2025. [En línea]. Disponible en: https://dazzet.co/que-es/commercial-off-the-shelf-software/

[20] «Apache Flink vs Apache Spark: A detailed comparison for data processing», Mage. Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://www.mage.ai/blog/apache-flink-vs-apache-spark-detailed-comparison-for-data-processing

[21] «Centro de ayuda». Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://es.mathworks.com/help/index.html

[22] «Digital Signal Processing Design for FPGAs and ASICs - MATLAB & Simulink». Accedido: 19 de mayo de 2025. [En línea]. Disponible en: https://es.mathworks.com/help/dsphdl/gs/dsphdl-design-for-fpgas-and-asics.html

[23] «Matlab versus Octave | Matemata». Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://blogs.upm.es/matemata/2020/11/02/matlab-versus-octave/

[24] jacabrera93, «LabVIEW: ¿Qué es, cómo funciona y para qué sirve?», iElectel - Electrónica y Telecomunicaciones. Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://ielectel.com/labview-que-es-como-funciona-y-para-que-sirve/

[25] «Simulación y diseño basado en modelos con Simulink». Accedido: 26 de mayo de 2025. [En línea]. Disponible en: https://es.mathworks.com/products/simulink.html

[26] L. B. Castillo, M. F. Marín, G. J. O. Jouvin, y I. S. C. Chica, «Optimización del uso de ancho de banda en los enlaces de transmisión de datos por medio de algoritmos de compresión.», *REVISTA CIENTÍFICA ECOCIENCIA*, vol. 6, n.º 1, Art. n.º 1, feb. 2019, doi: 10.21855/ecociencia.61.181.

[27] O. Media, «Low-latency processing, open architectures key for smarter radar/EW systems - Military Embedded Systems». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://militaryembedded.com/radar-ew/rugged-computing/low-latency-key-smarter-radarew-systems

[28] S. Chen *et al.*, «Efficient sequencing data compression and FPGA acceleration based on a two-step framework», *Front. Genet.*, vol. 14, sep. 2023, doi: 10.3389/fgene.2023.1260531.

[29] G. Martins Dias, «Prediction-based strategies for reducing data transmissions in the IoT», http://purl.org/dc/dcmitype/Text, Universitat Pompeu Fabra, 2016. Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://recerca.uoc.edu/documentos/6560f414d280522600935114

[30] M. J. Basgall, M. Naiouf, y A. Fernández, «FDR2-BD: A Fast Data Reduction Recommendation Tool for Tabular Big Data Classification Problems», *Electronics*, vol. 10, n.º 15, Art. n.º 15, ene. 2021, doi: 10.3390/electronics10151757.

[31] B. L. Pradeep, R. Anand, P. Vadakattu, S. Azeemudin, y A. Ahmed, «Design and Implementation of a Real-time Parallel FFT for a Direction-Finding System on an FPGA», en *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, sep. 2022, pp. 1-7. doi: 10.1109/HPEC55821.2022.9926310.

[32] N. M. Papadakis, I. Aroni, y G. E. Stavroulakis, «Effectiveness of MP3 Coding Depends on the Music Genre: Evaluation Using Semantic Differential Scales», *Acoustics*, vol. 4, n.º 3, Art. n.º 3, sep. 2022, doi: 10.3390/acoustics4030042.

[33] «Cómo Cumplir con ITAR para Exportadores de Defensa», Kiteworks | Your Private Data Network. Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://www.kiteworks.com/es/glosario-riesgo-cumplimiento/itar/

[34] A. Ortiz, «Introducción A La Construcción Modular: Qué Es Y Cómo Funciona - Arkiteckts». Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://arkiteckts.com/construccion/construccion-modular/introduccion-a-la-construccion-modular-que-es-y-como-funciona/

[35] «¿Qué es un gemelo digital? | IBM». Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://www.ibm.com/es-es/think/topics/what-is-a-digital-twin

[36] «fi - Crear un objeto numérico de punto fijo - MATLAB». Accedido: 19 de abril de 2025. [En línea]. Disponible en: https://es.mathworks.com/help/fixedpoint/ref/embedded.fi.html

[37] P. W. Chen, N. Wary, L. Wang, Q. Wang, y A. C. Carusone, «All-Digital Calibration Algorithms to Correct for Static Non-Linearities in ADCs», en *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, oct. 2020, pp. 1-5. doi: 10.1109/ISCAS45731.2020.9180833.

[38] S. Sudharman, A. D. Rajan, y T. S. Bindiya, «Design of a Power-Efficient Low-Complexity Reconfigurable Non-maximally Decimated Filter Bank for High-Resolution Wideband Channels», *Circuits Syst Signal Process*, vol. 38, n.º 6, pp. 2703-2735, jun. 2019, doi: 10.1007/s00034-018-0988-0.

[39] S. W. Smith, *The scientist and engineer's guide to digital signal processing*, 2nd edition. San Diego (Calif.): California Technical Pub., 1999.

[40] «Adquirir una señal analógica: ancho de banda, teorema de muestreo de Nyquist y aliasing». Accedido: 27 de abril de 2025. [En línea]. Disponible en: https://www.ni.com/es/shop/data-acquisition/measurement-fundamentals/analog-fundamentals/acquiring-an-analog-signal--bandwidth--nyquist-sampling-theorem-.html

[41] «Coste laboral por hora efectiva por divisiones de la CNAE-09(6037)», INE. Accedido: 21 de junio de 2025. [En línea]. Disponible en: https://www.ine.es/jaxiT3/Tabla.htm?t=6037&L=0

[42] M. J. Gamez, «Objetivos y metas de desarrollo sostenible», Desarrollo Sostenible. Accedido: 21 de mayo de 2025. [En línea]. Disponible en: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[43] «Descargar e instalar MATLAB - MATLAB & Simulink». Accedido: 21 de mayo de 2025. [En línea]. Disponible en: https://es.mathworks.com/help/install/ug/install-products-with-internet-connection.html

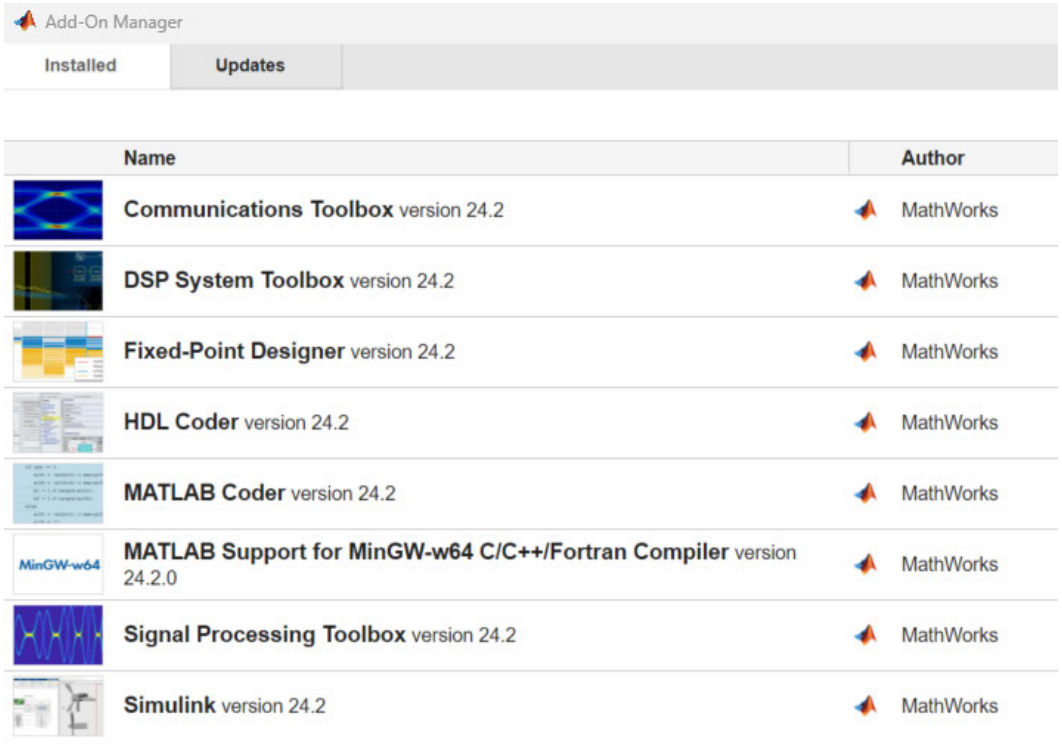
ANEXO I: Manual de Usuario

En este anexo, se va a explicar cómo ejecutar el código proporcionado, con imágenes explicativas y descripciones. Se va a dividir en dos secciones, una sobre como ejecutar los scripts.m que se han proporcionado, y otra sobre como ejecutar en Matlab Simulink el diseño digital implementado.

A.1 Manual de script.m – Matlab

En primer lugar, se va a explicar cómo ejecutar los scripts "*PreprocesadorEW.m", "pruebasDeIntegracion.m"* y *"pruebasUnitarias.m".* Los tres scripts se ejecutan de la misma manera, por lo que esta explicación sirve para todos. Aunque, en el caso de *"pruebasDeIntegracion.m"* hay que tener en cuenta, que es necesario situar en la misma ruta o carpeta donde se encuentra, el archivo de "*PreprocesadorEW.m"*, ya que el archivo de las pruebas de integración llama al del preprocesador para tener acceso a todas las variables creadas desde el "*Workspace"* de Matlab. Del mismo modo, para ejecutar las pruebas unitarias, se debe tener la función *"procesadoEspectralSenal.m"* en la misma ruta.

En primer lugar se deberá instalar Matlab, con la respectiva licencia de uso, a través de la página oficial de Mathworks [43]. Durante la instalación se preguntará que "toolboxes" o *"Add_Ons*" se quieren instalar, y debemos seleccionar las que aparecen en la Figura A.1.1, a excepción del *"Matlab Support for MinGW-w64 C/C++/Fortran Compiler*" y el *"HDL Coder"*, que se instalaron para realizar pruebas sobre HDL, pero para ejecutar este script no son estrictamente necesarios. La instalación de estos "*toolboxes"* no tiene ningún coste adicional.

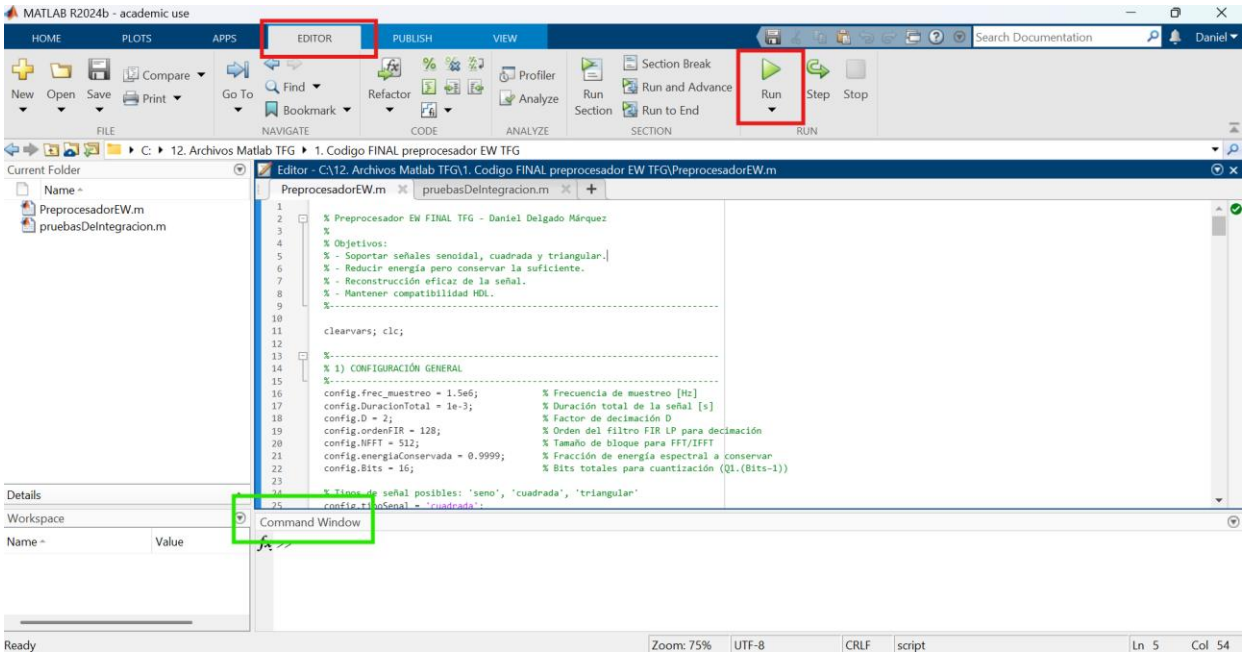


A.1. 1: Toolboxes de Matlab que se deben instalar

Una vez tenemos instalado Matlab y las toolboxes, se podrá ejecutar el código. Para hacerlo, simplemente se debe navegar a través del panel derecho que se ve en la Figura A.1.2 llamado *"Current Folder"* hasta la ruta donde se hayan guardado los archivos.m y abrirlos.

Una vez abiertos, hay dos formas de ejecutarlo. La primera más sencilla, es dar al botón "Run" que aparece en el menú Editor, como se ve en la Figura A.1.3 enmarcado con un rectángulo rojo. La segunda manera, consiste en escribir el nombre del archivo en la ventana de comandos enmarcada con un rectángulo verde en la parte inferior y pulsar Enter. (IMPORTANTE: Si se decide usar este método el nombre debe coincidir exactamente, mayúsculas y minúsculas).

Al principio del código de "*PreprocesadorEW.m"* en la línea 25, se puede elegir el tipo de señal generada escribiendo entre comillas simples la señal deseada, por ejemplo: 'triangular'.



A.1. 2: Ventana principal Matlab

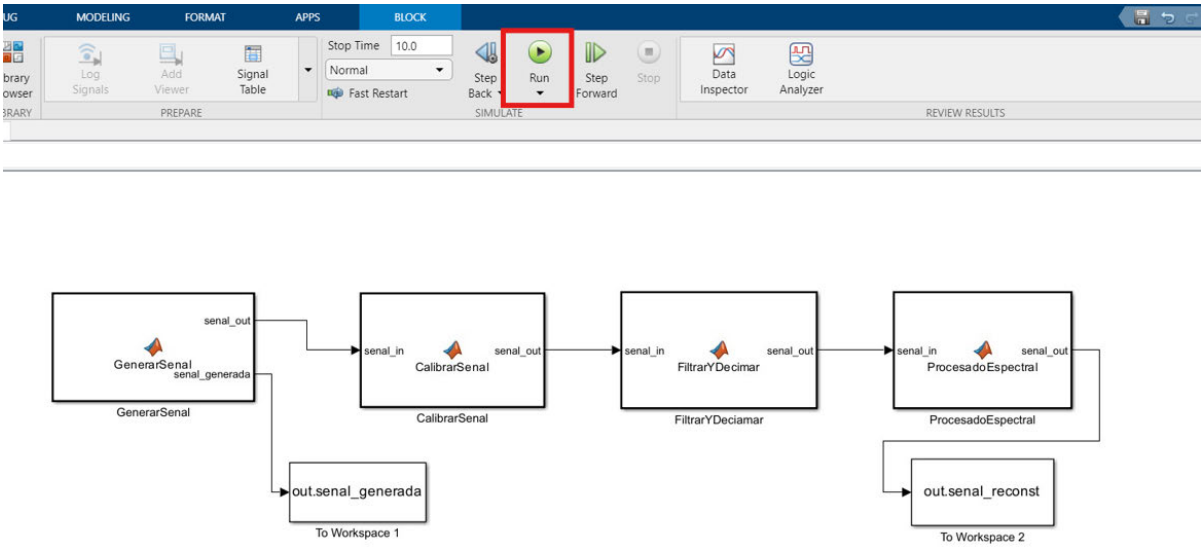
Una vez ejecutado el código se representarán las figuras correspondientes a la señal generada con y sin ruido, y la comparación entre la señal generada con ruido y la reconstruida.

Si en vez de ejecutar este código ejecutamos "*pruebasDeIntegracion.m*", nos saldrán todas las figuras representadas durante el apartado 5.2 Pruebas de Integración, y los respectivos mensajes por pantalla que se ven en la *"Command Window*", también explicados durante las pruebas de integración.

A.2 Manual de Matlab Simulink

En este anexo, se va a explicar cómo ejecutar el archivo del modelo digital "*PreprocesadorEW_Simulink.slx"* adjuntado con los archivos de este proyecto. Es importante mencionar, que al haber implementado este diseño en la aplicación de Matlab Simulink 2025, es necesario tener instalada la versión de "MATLAB R2025" para poder ejecutarlo.

Una vez abierto Matlab, en el panel de la izquierda mencionado anteriormente *"Current Folder"* se debe dar doble click al archivo del modelo digital "*PreprocesadorEW_Simulink.slx",* y se abrirá automáticamente Simulink y el proyecto del archivo, como se muestra en la Figura A.2.1. Una vez abierto, para ejecutarlo, de forma análoga a como se ha explicado en el Anexo A.1, se debe dar al botón de "Run" enmarcado en rojo en la Figura A.2.1



A.2. 1: Ventana principal proyecto "PreprocesadorEW_Simulink.slx"

Una vez finalizada la ejecución se añadirán al "Workspace" las variables necesarias para visualizar las gráficas obtenidas de Simulink. Para ejecutar la representación, solamente se deberá ejecutar el script "graficasSenalesSimulink.m" de la misma manera que antes, dando al botón "Run", y se mostrarán por pantalla.