
Hardware Design for Medical Hyperspectral Image Classification on
FPGA using VHDL

Diseño Hardware para la Clasificación de Imágenes Hiperespectrales
Médicas en FPGA usando VHDL



TRABAJO FIN DE GRADO

Iván Nieto Peralvo

Director:
Carlos Gonzalez Calvo

Calificación: **9.0**

Facultad de Informática
Universidad Complutense de Madrid

Junio 2026

”La verdadera victoria se logra cuando contribuimos al desarrollo de los demás, tanto dentro como fuera del tatami.”— Jigoro Kano

Agradecimientos

En primer lugar, me gustaría expresar mi más profundo agradecimiento a mi tutor Carlos, has sido una persona excepcional de la cual he aprendido muchísimo en todos los sentidos, tanto a nivel personal como académico.

También, me gustaría darle las gracias a mis padres y a mi hermana, por apoyarme desde el principio, y sobre todo motivarme en mis momentos de mayor duda, os quiero mucho.

Por último, a todos los profesores, profesoras y amigos que he conocido durante estos cinco años, y han hecho de mi etapa universitaria un periodo increíble del que siempre guardaré un muy buen recuerdo.

Resumen

La medicina contemporánea ha estado reinventándose constantemente durante las últimas décadas, con el objetivo de mejorar los métodos de diagnóstico, así como el tratamiento y seguimiento de los pacientes. Gracias al avance tecnológico reciente, se ha conseguido el desarrollo de herramientas cada vez más precisas, como la cirugía asistida por robot o las técnicas de análisis basadas en inteligencia artificial.

Entre estas, destacamos un tipo de imágenes conocidas como “hiperespectrales”, que inicialmente se crearon para estudiar materiales de la superficie terrestre desde satélites con una elevada precisión. Dado que contienen información espectral detallada para cada uno de sus píxeles, permiten una caracterización más específica de los tejidos biológicos humanos, lo que las convierte en una herramienta emergente en el ámbito del diagnóstico médico, especialmente a la hora de detectar cáncer de piel o tumores cerebrales.

El presente Trabajo de Fin de Grado se centrará en el diseño e implementación de un sistema de clasificación de estas imágenes sobre una plataforma FPGA, utilizando para ello el lenguaje de descripción hardware VHDL. Se llevará a cabo el desarrollo de un clasificador basado en el algoritmo de aprendizaje supervisado K-NN, adaptado desde C++ a una arquitectura hardware reconfigurable, con el fin de acelerar el procesamiento y hacerlo viable en tiempo real, utilizando para ello una plataforma FPGA de gama baja.

Palabras clave: Imagen hiperespectral, hardware reconfigurable, FPGA, VHDL, C++, clasificador, K-NN.

Abstract

For decades, contemporary medicine has been constantly reinventing itself, with the aim of improving diagnostic, treatment, and patient monitoring methods. Technological progress has enabled the development of increasingly precise tools, such as robot-assisted surgery and analysis techniques based on artificial intelligence.

Among these advances, we highlight a type of images known as hyperspectral, which were initially created to study materials on the Earth's surface from satellites with high precision. Since they contain detailed spectral information for each of their pixels, they allow a more specific characterization of human biological tissues, making them an emerging tool in the field of medical diagnosis, especially for detecting skin cancer or brain tumors.

This Bachelor's Thesis will focus on the design and implementation of a classification system for these images on an FPGA platform, using the VHDL hardware description language. A classifier based on the supervised learning algorithm known as K-NN, will be developed and adapted from C++ to a reconfigurable hardware architecture in order to accelerate processing and make it feasible in real time, using for this purpose a low-range FPGA platform.

Keywords: Hyperspectral imaging, reconfigurable hardware, FPGA, VHDL, C++, classifier, K-NN.

Índice general

Agradecimientos	I
Resumen	II
Abstract	III
Índice de figuras	VI
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Plan de trabajo	2
1.4. Estructura del documento	3
2. Introduction	5
2.1. Motivation	5
2.2. Objectives	6
2.3. Work plan	6
2.4. Document structure	7
3. Estado del arte	9
3.1. Imágenes hiperespectrales y sus aplicaciones	9
3.2. El proyecto HELICoiD como motivación	11
3.3. Placas programables FPGA	13
3.4. Lenguajes de descripción de hardware	14
4. Previo al algoritmo K-NN	16
4.1. Reducción de Dimensionalidad (PCA)	16
4.2. Clasificación Supervisada (SVM)	17
5. ¿Qué es el K-NN y cómo lo implementaremos?	18
6. Implementación en la FPGA	21
6.1. Paso 1 (Memoria):	23
6.2. Paso 2 (Distancia Euclídea):	24
6.3. Paso 3 (Ordenación):	27
6.4. Paso 4 (Asignación de Clase):	29
6.5. Unidad de Control como Máquina de Estados	31
7. Resultados experimentales	34
7.1. Verificación Funcional y Simulación	35
7.2. Análisis de Rendimiento y Tiempo de Procesamiento	37
7.3. Análisis Temporal del Circuito	38
7.4. Implementación y Utilización de Recursos	39

7.5. Validación Física en la FPGA	41
8. Conclusiones y trabajo futuro	44
8.1. Conclusiones	44
8.2. Trabajo futuro	45
9. Conclusions and future work	46
9.1. Conclusions	46
9.2. Future work	47
Bibliografía	48

Índice de figuras

3.1. Hipercubo de datos mostrando las dimensiones espaciales (X, Y) y la dimensión espectral (λ) [1].	10
3.2. Operación cerebral usando la tecnología de HELICoiD para la visualización de tumores en tiempo real [13].	11
3.3. Cadena de procesamiento de imágenes HSI. Se observa la ejecución paralela del algoritmo de extracción de características (PCA) y el clasificador supervisado (SVM), cuyos resultados convergen en el filtro espacial-espectral (K-NN) [10]. . .	12
3.4. Evolución de la arquitectura de una FPGA: a la izquierda, el modelo clásico y a la derecha el modelo moderno (con DSPs, Block RAMs y controladores) [22]. . .	14
6.1. <i>Cadena de montaje</i> del algoritmo K-NN para la clasificación de píxeles.	22
6.2. Módulo que calcula la distancia euclídea entre p y cada píxel del <i>dataset</i>	27
6.3. Módulo de comparación <code>celda_minimo.vhd</code>	29
6.4. Módulo que calcula la asignación final de la clase al píxel p	31
6.5. Diagrama de la Máquina de Estados Finita implementada en la Unidad de Control, desde la inicialización (IDLE) hasta la clasificación de todos los píxeles de la imagen (DONE).	33
7.1. Captura del analizador de ondas mostrando las transiciones de la máquina de estados al inicio del procesado de una ventana espacial (<code>read_center</code> , <code>store_center</code>).	34
7.2. Simulación funcional correcta del módulo K-NN, mostrando la transición al estado <code>next_pixel</code> tras emitir la clase predicha (2) y las probabilidades finales.	35
7.3. Vista del entorno de simulación de Vivado tras finalizar el procesado de la imagen. En la consola Tcl inferior se registra un tiempo total de simulación de 780.7 ms.	38
7.4. Reporte de tiempos tras la implementación del diseño. Se observa un <i>Worst Negative Slack</i> de 1.480 ns y cero errores de <i>Setup/Hold</i>	39
7.5. Reporte de utilización de recursos post-síntesis extraído de Vivado.	40
7.6. Vista de dispositivo mostrando el mapa de ocupación del chip tras el proceso de enrutamiento. Las regiones coloreadas corresponden a los distintos módulos instanciados en el clasificador K-NN.	41

Capítulo 1

Introducción

Actualmente, la práctica clínica está en constante búsqueda de la precisión. La evolución de las nuevas tecnologías y su introducción en el ámbito médico ha transformado el panorama por completo, mejorando desde la observación tradicional a nivel macroscópico hacia sistemas más complejos y específicos, que permiten una comprensión profunda de la fisiología humana a niveles antes inalcanzables. En este contexto, el procesamiento de información espectral se ha convertido en uno de los métodos más prometedores para la asistencia quirúrgica en tiempo real y la detección de patologías críticas [3].

Las imágenes hiperespectrales (HSI), cuyo origen se remonta al estudio de los materiales de la superficie terrestre desde satélites orbitando alrededor de la Tierra, ofrecen nuevas expectativas en la oncología moderna, pues permiten detectar con elevada precisión la materia sana de la tumoral en el tejido humano, superando los límites de la visión convencional.

Esta memoria documenta el diseño y la implementación de una solución hardware específica, en la que utilizando lógica reprogramable y un lenguaje de descripción hardware, se propone un algoritmo de clasificación para el procesamiento de datos médicos en tiempo real.

1.1. Motivación

La detección temprana de patologías como el cáncer de piel, así como la precisión durante la extirpación de tumores cerebrales, son desafíos críticos en el ámbito médico actual. Las imágenes hiperespectrales han surgido como una herramienta muy útil capaz de ofrecer información detallada que el ojo humano no puede percibir. Mientras que en algunas enfermedades mejoran la capacidad de diagnóstico inmediato, en el caso de los tumores cerebrales su función principal es guiar al cirujano en tiempo real durante la operación. Sin embargo, estas imágenes generan una inmensa cantidad de datos, y requieren de una capacidad de cómputo elevada por sus largos tiempos de procesamiento, lo que dificulta su viabilidad en tiempo real dentro del quirófano [1].

Aquí es donde entran en juego las placas basadas en FPGAs (*Field-Programmable Gate Arrays*), dispositivos que permiten diseñar arquitecturas hardware a medida, ofreciendo un alto rendimiento junto con una eficiencia energética superior a los aceleradores convencionales basados en GPUs (*Graphics Processing Units*) y dependientes de un *host* [7]. Por tanto, la motivación de este trabajo nace de la necesidad de minimizar la distancia entre el procesamiento elevado de datos hiperespectrales y la aplicabilidad clínica real. Al implementar el algoritmo K-NN (*K-Nearest Neighbors*) directamente en hardware usando VHDL (*VHSIC Hardware Description Language*), buscamos demostrar que es posible obtener un sistema de clasificación eficiente, rápido y viable para asistir a los cirujanos en tiempo real durante las operaciones, sin depender de grandes servidores de cómputo.

1.2. Objetivos

El objetivo principal de este trabajo consiste en la implementación de un clasificador de imágenes hiperespectrales médicas, en una placa reprogramable FPGA en lenguaje VHDL. La lógica detrás de este clasificador vendrá dada por el algoritmo de aprendizaje supervisado K-NN; este estará desarrollado en C++, y trabajará con la información previamente obtenida por los algoritmos PCA y SVM.

La finalidad de este proyecto se basa en el desarrollo de dos implementaciones independientes del clasificador: una versión en software desarrollada en C++ y su correspondiente en hardware, descrita en VHDL mediante Vivado. Este enfoque permite utilizar la implementación en C++ como un modelo de referencia para validar la arquitectura hardware, pues gracias a la comparación de los resultados obtenidos en ambos entornos, se garantiza que el diseño en la FPGA mantiene la integridad y la precisión del algoritmo original.

Se cubren por tanto los siguientes aspectos:

- Analizar la viabilidad de implementar algoritmos de clasificación hiperespectral en hardware reconfigurable.
- Desarrollar una arquitectura eficiente en VHDL para la clasificación de datos preprocesados (PCA y SVM) que permita un procesamiento optimizado e independiente de la carga computacional del software.
- Evaluar el rendimiento del sistema en términos de ocupación de recursos lógicos (LUTs, registros, BRAMs), uso de bloques aritméticos (DSPs) y robustez temporal (*Slack*).
- Validar la exactitud del diseño hardware frente a un modelo de referencia en software (C++) mediante un entorno de simulación automatizado.
- Verificar el funcionamiento real de la arquitectura sobre una placa FPGA Basys 3, demostrando su capacidad para operar de forma física y fiable en condiciones reales de ejecución.

En esencia, se busca contribuir al desarrollo de soluciones hardware especializadas para el procesamiento de datos biomédicos, con aplicaciones potenciales en sistemas portátiles de diagnóstico o dispositivos médicos integrados.

1.3. Plan de trabajo

Para alcanzar los objetivos propuestos, se ha seguido una metodología de desarrollo incremental dividida en varias fases, que van desde el estudio teórico hasta la implementación física y el análisis de resultados. La planificación se ha estructurado de la siguiente manera:

1. **Investigación y Estado del Arte:** Estudio de los fundamentos de las imágenes hiperespectrales, comprensión del algoritmo K-NN y revisión de las arquitecturas FPGA.
2. **Análisis y Modelo de Referencia en Software:** Implementación y validación del algoritmo K-NN en C++ para establecer un modelo de referencia y verificar los resultados de clasificación esperados.
3. **Diseño de la Arquitectura Hardware:** Planteamiento de la jerarquía de bloques funcionales (lectura de memoria, distancia euclídea, ordenación y asignación de clase) y diseño de la Unidad de Control como una máquina de estados finitos.
4. **Codificación en VHDL:** Descripción del hardware utilizando lenguaje VHDL, integrando IP Cores para el manejo de aritmética en punto flotante y gestionando el flujo de datos mediante el protocolo AXI Stream.

5. **Simulación y Verificación Funcional:** Realización de *testbenches* automatizados con datos médicos reales para comprobar que los módulos hardware obtienen los mismos resultados que el modelo de software.
6. **Síntesis, Implementación y Validación Física:** Generación del *bitstream* y volcado del diseño en la FPGA. Se incluye la creación de un módulo *wrapper* para validar el funcionamiento en placa utilizando memorias BRAM y señalización mediante LEDs.
7. **Análisis de Resultados y Redacción:** Evaluación técnica del diseño (tiempos de procesamiento, ocupación de recursos, *slack* temporal) y escritura final de la presente memoria.

1.4. Estructura del documento

Para esquematizar y entender mejor el desarrollo del trabajo, la información estará organizada en diferentes capítulos, desde conceptos iniciales como la explicación de las imágenes hiperespectrales, las FPGAs, los lenguajes HDL, hasta la implementación y desarrollo de los algoritmos utilizados junto con los resultados obtenidos. Estos capítulos serán:

1. **Introducción:** En este capítulo, y donde forma parte esta misma clasificación de las diferentes secciones en las que se va a estructurar el Trabajo Fin de Grado, se resume en que va a consistir el mismo, contando con subapartados como la motivación que hay detrás, los objetivos, así como el plan de trabajo que se ha llevado a cabo.
2. **Estado del Arte:** En este capítulo se presenta una visión general de los fundamentos teóricos del trabajo. Por un lado, se explica el concepto de las imágenes hiperespectrales y sus aplicaciones, con especial énfasis en el ámbito médico e introduciendo el proyecto europeo HELICoiD como la principal fuente de inspiración para el desarrollo de este trabajo. Además, se describe el material empleado en el desarrollo, tanto las placas programables FPGA como los lenguajes de descripción de hardware (HDL), detallando cómo ambos se combinan para llevar a cabo la implementación del proyecto.
3. **Previo al algoritmo K-NN:** Este capítulo presenta un análisis en conjunto de los algoritmos PCA y SVM, los cuales se ejecutan de forma concurrente previamente a la etapa de clasificación realizada mediante K-NN. Aunque no se profundiza en los detalles de su implementación, ya que los datos con los que se trabaja han sido previamente generados tras la ejecución de estos, resulta fundamental comprender su origen y el proceso mediante el cual se obtienen.
4. **¿Qué es el K-NN y cómo lo implementaremos?:** En este capítulo se realiza un estudio detallado del algoritmo K-NN. Se aborda desde la descripción de su tipo y funcionamiento hasta su implementación en C++, explicando paso a paso las estructuras de datos empleadas para favorecer su comprensión y optimización. Finalmente, se analiza su ejecución y los resultados obtenidos para ser posteriormente comparados con los del hardware. En conjunto, el objetivo es ofrecer una visión completa de la funcionalidad y la utilidad de este algoritmo.
5. **Implementación en la FPGA:** En este capítulo, tras entender la parte del software donde se desarrolla el algoritmo en su totalidad, pasaremos a la implementación en hardware del mismo. Dividido en secciones, se irán explicando cada una de las funcionalidades del algoritmo, y cómo han sido implementadas en la placa FPGA, adjuntando diversas imágenes a modo de esquema para entender correctamente cada uno de los módulos empleados y así tener una visión general del algoritmo en todos los aspectos.

6. **Resultados experimentales:** En este capítulo se expone el análisis cuantitativo y cualitativo del sistema diseñado. Se detallan los recursos de la FPGA consumidos y se presentan las métricas de rendimiento temporal, incluyendo el análisis del *slack* y la frecuencia máxima teórica. Además de la comparativa con el modelo de software para validar el *speedup* y la precisión, se incluye un apartado sobre la validación física del diseño en la placa, donde se demuestra el funcionamiento real de la arquitectura.
7. **Conclusiones y trabajo futuro:** Finalmente, este capítulo cierra la memoria resumiendo los logros alcanzados en relación con los objetivos iniciales, destacando la validación física del diseño y la consecución del procesado en tiempo real. Se reflexiona sobre la utilidad de las FPGAs en el entorno clínico y las limitaciones de memoria interna encontradas durante el desarrollo. Asimismo, se proponen líneas de trabajo futuro orientadas a la integración de todas las etapas de preprocesado en la propia placa, el uso de sensores hiperspectrales en vivo, el aprovechamiento del margen de frecuencia disponible y el uso de memoria RAM externa para ampliar la capacidad de procesamiento del sistema.

Capítulo 2

Introduction

Currently, clinical practice is in constant pursuit of precision. The evolution of new technologies and their introduction into the medical field has completely transformed the landscape, shifting from traditional observation at a macroscopic level toward more complex and specific systems that allow for a deep understanding of human physiology at levels previously unattainable. In this context, spectral information processing has become one of the most promising methods for real-time surgical assistance and the detection of critical pathologies [3].

Although initially developed for environmental and remote sensing, specifically the study of Earth's surface materials from satellites, hyperspectral imaging (HSI) offers new expectations in modern oncology. It allows for the highly precise differentiation between healthy and tumor tissue in human organs, overcoming the limits of conventional human vision.

This memory documents the design and implementation of a specific hardware solution in which, utilizing reprogrammable logic and a hardware description language, a classification architecture is proposed for real-time medical data processing.

2.1. Motivation

The early detection of pathologies such as skin cancer, as well as precision during the excision of brain tumors, are critical challenges in the current medical field. Hyperspectral images have emerged as a highly useful tool capable of offering detailed information that the human eye cannot perceive. While in some diseases they improve immediate diagnostic capabilities, in the case of brain tumors their primary function is to guide the surgeon in real time during surgery. However, these images generate an immense volume of data and require high computational capacity due to their long processing times, which hinders their viability in real time within the operating room [1].

This is where boards based on FPGAs (*Field-Programmable Gate Arrays*) come into play, devices that allow the design of custom hardware architectures, offering high performance along with superior energy efficiency compared to conventional, host-dependent GPU-based (*Graphics Processing Units*) accelerators [7]. Therefore, the motivation of this work stems from the need to bridge the gap between high-performance processing of hyperspectral data and real clinical applicability. By implementing the K-NN (*K-Nearest Neighbors*) algorithm directly in hardware using VHDL (*VHSIC Hardware Description Language*), we seek to demonstrate that it is possible to achieve an efficient, fast, and viable classification system to assist surgeons in real time during operations, without relying on large computing servers.

2.2. Objectives

The main objective of this work consists of implementing a medical hyperspectral image classifier on a reprogrammable FPGA board using VHDL. The logic behind this classifier is provided by the K-NN supervised learning algorithm; this is developed in C++ and works with information previously obtained by the PCA and SVM algorithms.

The purpose of this project is based on the development of two independent implementations of the classifier: a software version developed in C++ and its hardware counterpart, described in VHDL using Vivado. This approach allows using the C++ implementation as a reference model to validate the hardware architecture, since comparing the results obtained in both environments ensures that the FPGA design maintains the integrity and precision of the original algorithm.

The following aspects are therefore covered:

- Analyze the liability of implementing hyperspectral classification algorithms on reconfigurable hardware.
- Develop an efficient VHDL architecture for the classification of preprocessed data (PCA and SVM) that allows for optimized processing independent of the software's computational load.
- Evaluate system performance in terms of logical resource utilization (LUTs, registers, BRAMs), use of arithmetic blocks (DSPs), and timing robustness (*Slack*).
- Validate the accuracy of the hardware design against a software reference model (C++) using an automated simulation environment.
- Verify the actual operation of the architecture on a Basys 3 FPGA board, demonstrating its capability to operate physically and reliably under real execution conditions.

In essence, the aim is to contribute to the development of specialized hardware solutions for biomedical data processing, with potential applications in portable diagnostic systems or integrated medical devices.

2.3. Work plan

To achieve the proposed objectives, an incremental development methodology divided into several phases has been followed, ranging from theoretical study to physical implementation and results analysis. The planning has been structured as follows:

1. **Research and State of the Art:** Study of the fundamentals of hyperspectral imaging, understanding of the K-NN algorithm, and review of FPGA architectures.
2. **Analysis and Software Reference Model:** Implementation and validation of the K-NN algorithm in C++ to establish a reference model and verify the expected classification results.
3. **Hardware Architecture Design:** Definition of the hierarchy of functional blocks (reading data from memory, Euclidean distance, sorting, and class assignment) and design of the Control Unit using a finite state machine.
4. **Coding in VHDL:** Description of the hardware using VHDL language, integrating IP Cores for floating-point arithmetic management and managing data flow via the AXI Stream protocol.

5. **Simulation and Functional Verification:** Execution of automated *testbenches* with real medical data to verify that the hardware modules obtain the same results as the software model.
6. **Synthesis, Implementation, and Physical Validation:** Bitstream generation and design download onto the FPGA. This includes creating a *wrapper* module to validate on-board operation using BRAM memories and LED signaling.
7. **Results Analysis and Writing:** Technical evaluation of the design (processing times, resource utilization, timing *slack*) and final writing of this manuscript.

2.4. Document structure

To outline and better understand the development of this work, the information is organized into different chapters, ranging from initial concepts such as the explanation of hyperspectral images, FPGAs, and HDL languages, to the implementation and development of the algorithms used along with the obtained results. These chapters are as follows:

1. **Introduction:** In this chapter, which includes this classification of the different sections structuring the Thesis, the work is summarized, featuring subsections such as the motivation, the objectives, as well as the work plan carried out.
2. **State of the Art:** In this chapter, a general overview of the theoretical foundations of the work is presented. On one hand, the concept of hyperspectral imaging and its applications are explained, with special emphasis on the medical field, introducing the European HELI-CoID project as the primary source of inspiration for this work. Furthermore, the materials employed in the development are described, including both programmable FPGA boards and hardware description languages (HDLs), detailing how both combine to carry out the project's implementation.
3. **Prior to the K-NN algorithm:** This chapter presents a brief analysis of the PCA and SVM algorithms, which run concurrently prior to the classification stage performed by K-NN. Although their implementation details are not explored deeply, since the data used has been previously generated after their execution, understanding their origin and the process through which they are obtained is fundamental.
4. **What is K-NN and how will we implement it?:** In this chapter, a detailed study of the K-NN algorithm is carried out. It addresses everything from its type and operation to its implementation in C++, explaining step-by-step the data structures used to facilitate its understanding and optimization. Finally, its execution and the obtained results are analyzed to be subsequently compared with those of the hardware. Overall, the objective is to offer a comprehensive view of the functionality and utility of this algorithm.
5. **Implementation on the FPGA:** In this chapter, after understanding the software part where the algorithm is fully developed, we move on to its hardware implementation. Divided into sections, each of the algorithm's functionalities will be explained along with how they have been implemented on the FPGA board, attaching various images as diagrams to correctly understand each of the modules employed and thus gain an overall view of the algorithm in all aspects.
6. **Experimental results:** In this chapter, the quantitative and qualitative analysis of the designed system is presented. The consumed FPGA resources are detailed, and timing performance metrics are shown, including the slack analysis and the theoretical maximum frequency. In addition to the comparison with the software model to validate the speedup

and accuracy, a section on the physical validation of the design on the board is included, where the actual operation of the architecture is demonstrated.

7. **Conclusions and future work:** Finally, this chapter closes the memory by summarizing the achievements reached in relation to the initial objectives, highlighting the physical validation of the design and the attainment of real-time processing. It reflects on the utility of FPGAs in the clinical environment and the internal memory limitations encountered during development. Likewise, future lines of work are proposed, oriented toward the integration of all preprocessing stages on the board itself, the use of live hyperspectral sensors, taking advantage of the available frequency margin, and the use of external RAM memory to expand the system's processing capacity.

Capítulo 3

Estado del arte

En este capítulo se realizará una introducción técnica sobre los materiales y recursos fundamentales que sirven de base al trabajo, comenzando con las imágenes hiperespectrales, sus características generales y principales aplicaciones en el mundo real. También, se hablará de la motivación y punto de partida de donde ha surgido la idea para llevar a cabo este diseño en placa. Por último, se describirán los dispositivos FPGA y sus propiedades más relevantes, junto con los lenguajes HDL, explicando cómo permiten implementar de manera eficiente una gran cantidad de código en ellas, estableciendo un puente entre el software y el hardware.

3.1. Imágenes hiperespectrales y sus aplicaciones

Las **imágenes hiperespectrales** o **HSI** (*hyperspectral imaging*) constituyen una evolución de la fotografía digital. Mientras que una cámara estándar captura la luz en tres bandas anchas (rojo, verde y azul), la tecnología hiperespectral recopila y procesa información a lo largo de un amplio rango del espectro electromagnético. En este tipo de imágenes, los píxeles dejan de ser representaciones de color para convertirse en vectores que contienen una gran densidad de valores espectrales [14].

A diferencia de las imágenes convencionales (las que obtenemos con cámaras comunes), que suelen capturar solo tres bandas de luz: rojo, verde y azul (RGB), son una técnica analítica basada en la espectroscopia (estudio de la interacción entre la radiación electromagnética y la materia) que recoge cientos de imágenes de la misma región espacial a diferentes longitudes de onda, captando cientos de bandas en distintos rangos de luz, incluidos el infrarrojo cercano (NIR) o el ultravioleta (UV). Esta capacidad permite obtener una gran cantidad de información sobre las propiedades físicas y químicas de los objetos en la imagen.

Estas bandas capturadas forman una estructura de datos conocida como **hipercubo**. Se trata de una matriz tridimensional donde las dos primeras dimensiones (ejes X e Y) corresponden a la resolución espacial de la imagen, indicando la posición física del píxel en la escena observada. La tercera dimensión (eje Z o λ) corresponde al dominio espectral, almacenando la información de la luz captada en decenas o cientos de longitudes de onda estrechas y contiguas [1].

Si aislamos un único píxel espacial (x, y) y observamos su profundidad a lo largo del eje Z, obtenemos un vector de datos espectrales en lugar de un simple valor de color. Este vector representa la **firma espectral** de ese punto específico. Físicamente, la firma espectral es una curva que grafica la cantidad de energía electromagnética que la materia refleja, absorbe o transmite en función de la longitud de onda incidente. [8, 14].

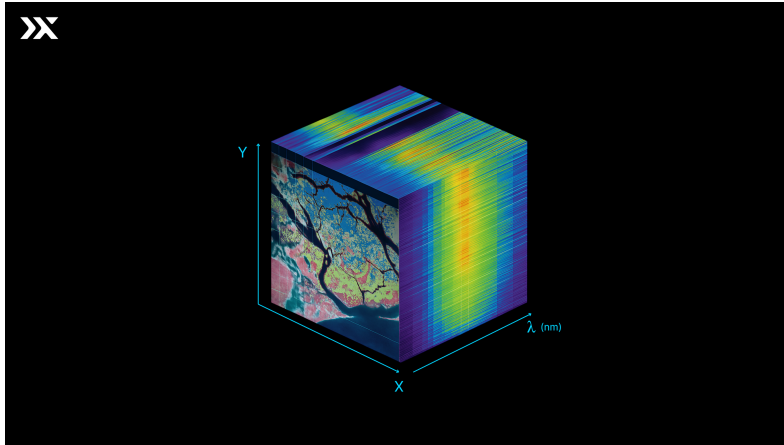


Figura 3.1: Hipercubo de datos mostrando las dimensiones espaciales (X, Y) y la dimensión espectral (λ) [1].

Dado que la interacción de la luz con la materia depende directamente de su composición molecular y de su estructura química, cada material presenta picos y valles de absorción característicos. En el contexto médico, las alteraciones patológicas (como la hipervascularización o el cambio en la densidad celular de un tumor cerebral) modifican esta interacción óptica. Por tanto, esta curva actúa como una “huella dactilar” única que el algoritmo K-NN utilizará como rasgo discriminatorio para identificar y clasificar el tejido de manera precisa [12].

El proceso de obtención de una imagen hiperespectral se basa en el uso de sensores especializados que descomponen la luz reflejada o emitida por los objetos en diferentes longitudes de onda. Estos datos son luego procesados para generar una imagen que puede revelar detalles ocultos que no serían visibles con la fotografía tradicional.

La finalidad y utilidad de estas imágenes es muy variada, pues facilitan mucho el trabajo en diversos campos relacionados con la investigación. Entre los más destacables tenemos:

- Teledetección y medio ambiente: debido a la enorme cantidad de bandas espectrales que detectan, permiten identificar la firma espectral de materiales presentes en el terreno. Esto facilita clasificar detalladamente la cobertura del suelo, identificar vegetación, detectar especies invasoras o estudiar la calidad del agua y los cambios ecológicos con mucha efectividad [18].
- Agricultura de precisión: permiten monitorizar la salud de cultivos detectando estrés, enfermedades o deficiencias nutricionales, medir contenido de clorofila o nitrógeno y estimar rendimiento o humedad del suelo. Esto apoya decisiones como riego o fertilización precisa [2].
- Industria y seguridad alimentaria: detección de contaminantes físicos, presencia de huesos o residuos orgánicos e identificación de componentes como grasa o humedad [16, 17].
- Aplicaciones médicas y detección de cáncer: permiten realizar métodos diagnósticos no invasivos capaces de distinguir tejidos sanos de tumorales. Se han logrado sensibilidades del 87 % y especificidad del 88 % en detección de cáncer de piel, y del 86 % al 95 % en cáncer colorrectal [3, 13].
- Detección de falsificaciones y defensa: identifican firmas espectrales específicas en billetes, bebidas u otros productos, detectando falsificaciones con un sistema espectral en rangos de 400 a 500 nm con probabilidad de acierto superior al 99 %. También se emplea en defensa

para detectar objetos camuflados mediante diferencias sutiles espectrales invisibles al ojo humano [15].

- Minería y geología: caracteriza minerales identificando sus firmas espectrales individuales. Esto permite generar mapas detallados de composición y distribución de minerales en la superficie terrestre o en muestras geológicas [19, 21].

En este trabajo nos centraremos en la parte de las aplicaciones médicas. Para ello, se hará uso de unos dispositivos de hardware programables llamados FPGA, pero antes hablemos de la motivación que sirvió de inspiración para llevar a cabo esta investigación.

3.2. El proyecto HELICoiD como motivación

La idea principal que motiva el desarrollo de este Trabajo de Fin de Grado nace al estudiar investigaciones recientes como el proyecto europeo **HELICoiD** (*HypErspectraL Imaging Cancer Detection*) [13]. Este proyecto propone una técnica sin contacto y no invasiva cuyo objetivo principal es asistir a los neurocirujanos en tiempo real durante sus operaciones. Mediante el uso de cámaras hiperespectrales, el sistema busca diferenciar entre tejido cerebral sano y tumoral durante las intervenciones quirúrgicas, evitando así la extirpación excesiva de tejido normal o la permanencia de restos tumorales.

La necesidad médica detrás de esto es vital. A diferencia de otros tumores, el *glioblastoma* y otros tipos de cáncer cerebral se infiltran en el tejido sano y sus límites son extremadamente difusos, lo que los hace casi indistinguibles a simple vista para el cirujano [5]. Al capturar la información de la escena, el sistema de HELICoiD procesa la firma espectral del cerebro para mostrar al médico un mapa visual que delimita exactamente las zonas tumorales, tal y como se observa en la Figura 3.2.



Figura 3.2: Operación cerebral usando la tecnología de HELICoiD para la visualización de tumores en tiempo real [13].

La cadena de procesamiento Espacial-Espectral

Para lograr esta clasificación con alta precisión, el método de trabajo de HELICoiD emplea una aproximación que tiene en cuenta tanto las características espectrales como las espaciales de la imagen. Esta cadena de procesamiento se compone de tres algoritmos fundamentales (PCA, SVM y K-NN) que explicaremos más en detalle en el siguiente capítulo:

1. **Reducción de Dimensionalidad:** Dado que el hipercubo original contiene cientos de bandas espectrales, el algoritmo de Análisis de Componentes Principales (PCA) se ejecuta para extraer las características más importantes y comprimir la información, generando una representación de la imagen como una sola banda [10, 11].
2. **Clasificación Supervisada:** De forma paralela a la ejecución del PCA, se utiliza una Máquina de Vectores de Soporte (SVM) que actúa como clasificador píxel a píxel. Este modelo, previamente entrenado con datos etiquetados por médicos especialistas, evalúa la firma espectral de cada píxel y genera un mapa de probabilidades que indica a qué clase de tejido pertenece [4].
3. **Filtrado Espacial-Espectral:** El último paso de la cadena es el algoritmo de K-Vecinos Más Cercanos (K-NN). Este actúa como un filtro que toma como entradas el mapa de probabilidades generado por el SVM y la imagen de guía generada por el PCA. Utilizando tanto las coordenadas espaciales como la similitud espectral, el K-NN asegura la coherencia espacial del resultado definiendo con exactitud las fronteras del tumor.

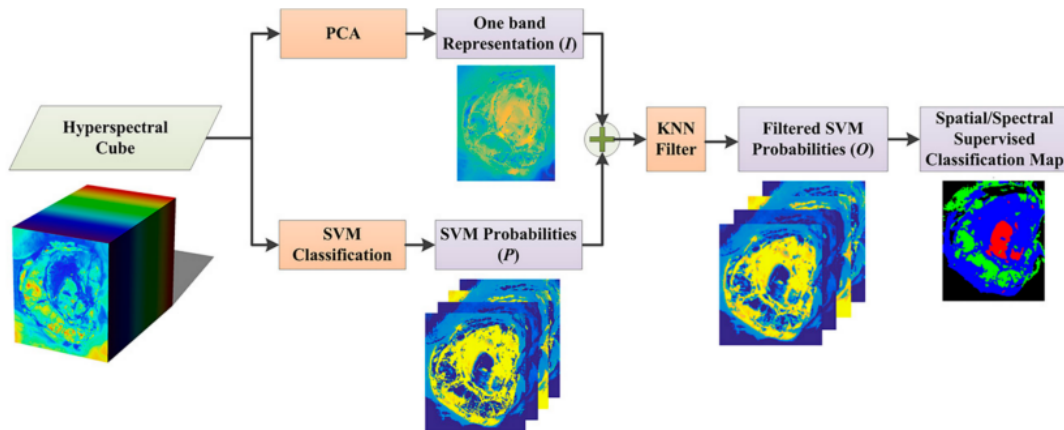


Figura 3.3: Cadena de procesamiento de imágenes HSI. Se observa la ejecución paralela del algoritmo de extracción de características (PCA) y el clasificador supervisado (SVM), cuyos resultados convergen en el filtro espacial-espectral (K-NN) [10].

El cuello de botella computacional y la justificación del Hardware

Sin embargo, el gran obstáculo para que esta tecnología triunfe en un entorno quirúrgico es su enorme coste computacional. Clasificar hipercubos de datos requiere procesar cientos de miles de valores por cada escena, y si la ejecución de la cadena completa (PCA + SVM + K-NN) tarda demasiado, el mapa llega tarde a las manos del cirujano y el sistema pierde toda su utilidad clínica.

Es esta barrera tecnológica la que sirve de inspiración para plantear la solución en el presente trabajo. En el proyecto HELICoiD ha establecido que la clasificación espacial-espectral es una de las etapas más exigentes a nivel temporal, pero que su paralelización mediante aceleradores hardware permite reducir significativamente sus tiempos de ejecución [4]. Por ello, para evitar el

cueillo de botella que supone procesar algoritmos como el K-NN en software convencional, surge la idea de llevar el clasificador directamente al hardware, diseñando una arquitectura usando lenguaje VHDL sobre una placa FPGA para explotar al máximo la computación paralela de estos dispositivos [6].

3.3. Placas programables FPGA

Una **FPGA** (*Field-Programmable Gate Array*) es un circuito integrado que puede ser reconfigurado por el usuario tras su fabricación. Esto la diferencia de la mayoría de circuitos integrados, como los procesadores (CPU), que tienen una arquitectura fija diseñada para ejecutar una variedad de instrucciones predefinidas [20]. Concretamente para este trabajo, utilizaremos una **Basys 3** equipada con un chip de la familia *Artix-7*.

A nivel interno, una FPGA está compuesta por una matriz de componentes configurables que trabajan en conjunto. Para comprender el consumo de recursos de los diseños hardware, es fundamental conocer los elementos que conforman su arquitectura:

- **Bloques Lógicos:** Son la unidad fundamental de procesamiento. Contienen tablas de búsqueda (**LUTs** o *Look-Up Tables*) para implementar cualquier función lógica combinatorial, y biestables (**FFs** o *Flip-Flops*) para el almacenamiento de estados y el diseño de circuitos secuenciales. En arquitecturas modernas, estos elementos se agrupan en estructuras de mayor nivel conocidas como Módulos Lógicos Adaptativos (**ALMs** o *Adaptive Logic Modules*).
- **Bloques de Memoria (BRAM/RAM o Block RAM):** Dado que el procesamiento elevado de datos requiere almacenamiento rápido, las FPGAs incorporan bloques de memoria internos. Esto permite guardar variables temporales, porciones de imágenes (ventanas) o *datasets* en el propio chip, reduciendo enormemente la latencia respecto a la memoria externa.
- **Bloques de Procesamiento Digital de Señales (DSPs o Digital Signal Processors):** Son circuitos aritméticos integrados físicamente en la placa. Están optimizados para realizar operaciones aritméticas a gran velocidad, lo que resulta crítico para acelerar algoritmos con alta carga matemática.
- **Red de Interconexiones (Routing):** Es el conjunto de cables programables y conmutadores que permiten conectar dinámicamente las entradas y salidas de todos los componentes anteriores, definiendo así el flujo de datos del circuito.

Combinando estos recursos de diferentes maneras, se pueden crear circuitos complejos que procesan datos de forma muy eficiente e implementan tareas digitales de forma paralela y adaptada a su propósito específico.

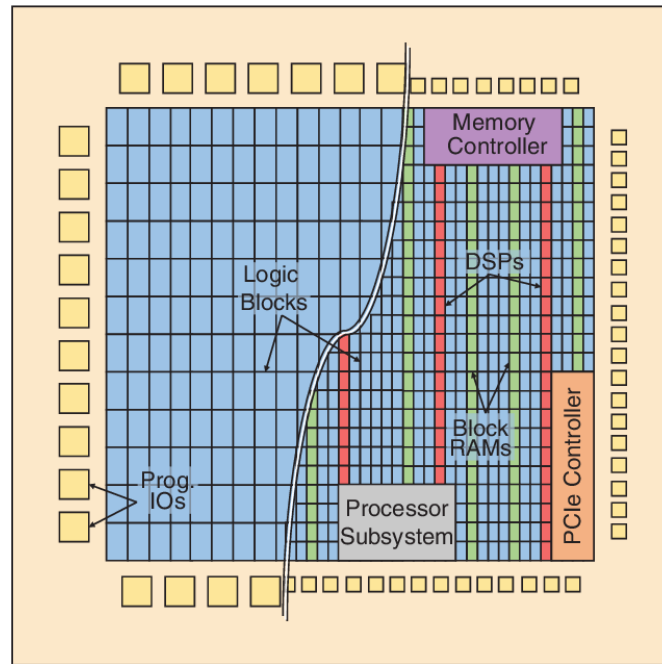


Figura 3.4: Evolución de la arquitectura de una FPGA: a la izquierda, el modelo clásico y a la derecha el modelo moderno (con DSPs, Block RAMs y controladores) [22].

El aspecto más destacable de estos circuitos es su capacidad de reprogramación: se pueden adaptar y actualizar tantas veces como se desee para abordar diferentes tareas, permitiendo una gran flexibilidad en diseño y funcionalidad. Esta cualidad hace que sean especialmente útiles en prototipos de hardware o aplicaciones en las que los requisitos cambian frecuentemente. Su flexibilidad y bajo consumo relativo las hacen ideales en aplicaciones de procesamiento de imágenes en tiempo real. A la hora de trabajar con ellas utilizaremos un lenguaje de descripción de hardware (HDL).

3.4. Lenguajes de descripción de hardware

Para configurar una FPGA, se utilizan lenguajes específicos que actúan como puente entre el diseño lógico y la placa. Esto se denominan **Lenguajes de Descripción de Hardware (HDL o *Hardware Description Languages*)**. Son lenguajes usados para describir, simular, verificar y sintetizar circuitos digitales, encargados de especificar el comportamiento del hardware a bajo nivel, de estructurar las interconexiones de los bloques lógicos así como la evolución de estas señales en el tiempo [20].

Su principal diferencia frente a los lenguajes de programación software es que los HDL modelan sistemas concurrentes, permitiendo que múltiples instrucciones puedan ejecutarse al mismo tiempo de forma paralela, a diferencia del código en software que se ejecuta secuencialmente.

HDL frente a HLS

En los últimos años, han ganado mucha popularidad las herramientas de **Síntesis de Alto Nivel (HLS o *High-Level Synthesis*)**. Estas tecnologías permiten programar las FPGAs utilizando lenguajes de programación software convencionales (como C o C++) mediante compiladores especializados. Su principal ventaja es que acerca el mundo del hardware a los desarrolladores de software, reduciendo los tiempos de programación al ocultar la complejidad física del chip.

Sin embargo, para el desarrollo de este trabajo se ha optado por utilizar directamente un lenguaje HDL puro, concretamente **VHDL**, debido a su nivel de control y la eficiencia: mientras que en HLS es el compilador quien toma las decisiones sobre cómo traducir las líneas de código a bloques físicos, generando a veces un uso ineficiente de los recursos de la FPGA, VHDL permite diseñar la arquitectura a medida, teniendo control total sobre cada ciclo de reloj de la gestión de la memoria y la segmentación de las operaciones [9].

Dado que el objetivo de este proyecto es acelerar el algoritmo K-NN para el diagnóstico médico en tiempo real, el uso de VHDL garantiza poder aprovechar al máximo el rendimiento y el espacio del chip.

Capítulo 4

Previo al algoritmo K-NN

El proceso de clasificación de las imágenes hiperespectrales para la detección de tumores no está compuesto únicamente por un único algoritmo, sino que se rige en una cadena de procesamiento espacial-espectral. Esta cadena, como vimos por encima en el capítulo anterior, está compuesta por tres algoritmos principales: PCA, SVM y K-NN. Este último es el algoritmo central de la implementación y será explicado en profundidad en el próximo capítulo; sin embargo, para entender su función, es indispensable conocer el contexto de los datos que recibe como entrada.

A nivel de arquitectura, las imágenes hiperespectrales no se procesan de forma puramente secuencial. Para optimizar el tiempo de respuesta en un entorno quirúrgico, los algoritmos PCA y SVM se ejecutan de manera concurrente. El algoritmo K-NN actuará como la etapa final, fusionando los resultados de ambos para producir el diagnóstico definitivo, tal y como se ilustra en la Figura 3.3 del capítulo anterior.

Aunque el enfoque de este trabajo no es detallar la especificación del PCA ni del SVM (ya que las imágenes que utilizaremos vendrán procesadas por ellos), a continuación se exponen sus fundamentos para comprender qué información aportan.

4.1. Reducción de Dimensionalidad (PCA)

Por un lado, el hipercubo de datos original pasa por un algoritmo de reducción de dimensionalidad conocido como **PCA** (*Principal Component Analysis*), introducido formalmente por Harold Hotelling. Su objetivo es retener la mayor cantidad de información relevante posible, eliminando la redundancia espectral [20].

Es importante destacar que este algoritmo no convierte la imagen en una fotografía 2D plana, sino que reduce considerablemente el número de bandas espectrales (la “profundidad” del cubo) conservando su resolución espacial (filas y columnas). En nuestro caso, el hipercubo se comprime habitualmente a una única componente principal que concentra la mayor varianza de los datos y sirve como mapa de guía espacial para el K-NN.

Este algoritmo se divide en cuatro pasos principales:

1. Organizar los datos de la matriz original, calculando la media de cada una de las bandas espectrales y restándosela a los píxeles correspondientes. Esto centra la matriz en el origen, aumentando la precisión al eliminar sesgos causados por la distribución heterogénea de los datos.
2. Calcular la matriz de covarianza del conjunto de datos original para expresar las correlaciones entre bandas.

3. Obtener los valores y vectores propios de la matriz de covarianza, ordenándolos de forma decreciente de acuerdo con su valor propio.
4. Proyectar la imagen original en el subespacio de los vectores propios elegidos para obtener la componente principal deseada.

4.2. Clasificación Supervisada (SVM)

De forma concurrente a la ejecución del PCA, los datos originales son procesados por un segundo algoritmo conocido como **SVM** (*Support Vector Machine*). Este algoritmo actúa como un clasificador base píxel a píxel [20].

Este algoritmo separa los datos en clases maximizando el margen. Para ello, busca el hiperplano óptimo que deje la mayor distancia posible entre los puntos más cercanos de las distintas clases (tejido sano, tumor, vasos sanguíneos, etc.). El objetivo de esta clasificación no es separar todos los puntos a la perfección, sino aislar las anomalías encontrando el plano de máxima distancia.

Como resultado, el SVM no emite un diagnóstico cerrado, sino que genera un mapa de probabilidades. Por cada píxel de la imagen, indica la probabilidad que tiene dicho píxel de pertenecer a cada una de las clases médicas.

Así, resumiendo ambas funcionalidades, podemos concluir que el PCA aporta el mapa guía simplificado, mientras que el SVM aporta las probabilidades que servirán para la clasificación final. Tras el entendimiento de estas dos fases previas, podemos meternos de lleno en el grueso del trabajo: cómo el K-NN filtra y unifica esta información en hardware.

Capítulo 5

¿Qué es el K-NN y cómo lo implementaremos?

El algoritmo **K-NN** (*K-Nearest Neighbors*) pertenece al conjunto de técnicas de aprendizaje automático supervisado, y puede ser utilizado tanto para problemas de clasificación como de regresión. Trata de buscar los k puntos más cercanos (llamados vecinos) a un punto concreto para poder determinar su valor a la hora de clasificarlo, teniendo en cuenta los valores de los otros [20].

El valor de k definirá cuántos vecinos se utilizarán para determinar la clasificación de un punto específico. Si $k = 1$, la instancia se asignará a la misma clase que su vecino más cercano. Este parámetro es crucial, pues en función de su valor el algoritmo puede conducir a un ajuste excesivo o insuficiente. Los valores más bajos de k pueden tener una varianza alta, pero un sesgo bajo, y los valores más altos de k pueden provocar un sesgo alto y una varianza más baja.

En nuestra versión del algoritmo, que estará implementado en una primera instancia en C++, trabajaremos con un conjunto de datos D , que consistirá en un *dataset* ordenado de píxeles, representados en forma de *structs*, conteniendo los datos (PCA_i, r_i, c_i, P_i) .

- PCA_i : Un *float* que representa el valor que le otorga el algoritmo PCA al píxel i .
- r_i : Un *int* que representa la fila de la matriz original para el píxel i .
- c_i : Un *int* que representa columna de la matriz original para el píxel i .
- $P_i = (p_{i1}, p_{i2}, \dots)$ un *vector* indicando la probabilidad que tiene el píxel i de pertenecer a cada una de las diferentes clases. Estas han sido obtenidas durante la ejecución del algoritmo SVM.

Una vez conocida la estructura de los píxeles y cómo estos se representan, podemos pasar a la parte de clasificación de las imágenes. Para ello, el algoritmo se divide en 5 pasos:

1. **Definir la ventana de trabajo:** Como se ha explicado anteriormente, la idea del algoritmo consiste en comparar los píxeles a clasificar con los k más cercanos, y así determinar a qué clases podrían pertenecer. Sin embargo, dada la magnitud de las imágenes, es recomendable ceñirnos a una región más pequeña para no gastar tanto espacio en memoria a la hora de ejecutarlo. Para ello, solo trabajaremos con un sector del *dataset* original, que irá seleccionando regiones específicas en función de qué píxeles estén más o menos cerca del que queremos clasificar.

2. **Calcular la distancia entre los puntos:** Una vez definida la ventana, se calculará la distancia euclídea entre cada píxel d_i perteneciente a ella y el píxel p que se desea clasificar. Cabe destacar que, en este caso, se omitirá la raíz cuadrada en la fórmula, ya que al ordenar los resultados obtenidos de menor a mayor para identificar los píxeles más afines, esta operación no afecta al resultado y solo añade un coste computacional innecesario. La expresión es la siguiente:

$$d(p, d_i) = (PCA_p - PCA_{di})^2 + (r_p - r_{di})^2 + (c_p - c_{di})^2$$

3. **Identificar los k vecinos más cercanos:** Una vez tengamos todas las distancias calculadas entre los píxeles del *dataset* y el píxel p , las ordenaremos de menor a mayor para seleccionar los k vecinos más cercanos, es decir, cuya distancia haya sido la menor y que, por tanto, son los que más influirán a la hora de determinar la clase del píxel.
4. **Seleccionar basándonos en las probabilidades del SVM:** A continuación, para cada clase se sumarán las probabilidades asociadas a los píxeles vecinos de pertenecer a ella. Luego, cada una de estas sumas se dividirá entre k , obteniendo así una media ponderada que representa el grado de afinidad del píxel p con las respectivas clases.

$$C_j = \frac{1}{k} \cdot \sum_{d_i \in D} p_{ij}, \quad \forall j \in \{1, 2, \dots\}$$

5. **Asignación de clase:** Por último, le asignaremos la clase con mayor media de las obtenidas anteriormente al píxel p con el que estamos trabajando, y habremos concluido la clasificación.

Veamos la implementación software en C++:

```
int kNN_algorithm(int k, pixel p, vector<vector<pixel>>img) {
    vector<pixel>dataset; // Creamos el dataset de pixeles

    // Nos aseguramos de no salirnos de los limites por filas
    int startX = max(0, p.x - HALF_WINDOW);
    int endX = min(FILAS - 1, p.x + HALF_WINDOW);

    // Hacemos lo mismo para las columnas
    int startY = max(0, p.y - HALF_WINDOW);
    int endY = min(COLUMNAS - 1, p.y + HALF_WINDOW);

    // Almacenamos en el dataset los pixeles de la ventana
    for (int i = startX; i <= endX; i++) {
        for (int j = startY; j <= endY; j++) {
            if (i != p.x || j != p.y) {
                // El pixel p no lo tenemos en cuenta
                dataset.push_back(img[i][j]);
            }
        }
    }

    vector<pair<float, pixel>> distanciasPix;
    for (const pixel& pix : dataset) {
        // Calculo de la distancia euclídea
        float distEuclídea = pow((p.PCA_value - pix.PCA_value), 2) +
```

```

        pow((p.x - pix.x), 2) +
        pow((p.y - pix.y), 2);
    distanciasPix.push_back({ distEuclidea, pix });
}

// Ordenamos por distancia ascendente
sort(distanciasPix.begin(), distanciasPix.end(),
    [](const auto& a, const auto& b) {
        return a.first < b.first;
    });

// Seleccionamos los k vecinos mas cercanos
vector<pixel> vecinos;
for (int i = 0; i < k; i++) {
    vecinos.push_back(distanciasPix[i].second);
}

// Usaremos un set para ordenar las clases en funcion de la
// mayor probabilidad de pertenecer a ella para el pixel p
set<pair<int, float>, comp> clases;

for (int i = 0; i < NUM_CLASES; i++) {
    float prob = 0;
    for (int j = 0; j < k; j++) {
        prob += vecinos[j].probsSVM[i];
    }
    clases.insert({ i + 1, (prob / k) });
}

// Devolvemos la clase con mayor probabilidad
return clases.begin()->first;
}

```

```

// codigo del main...

//Salida por pantalla
for (int i = 0; i < FILAS; i++) {
    for (int j = 0; j < COLUMNAS; j++) {

        int resultado = kNN_algorithm(k, img[i][j], img);
        fout << "Clase:" << resultado << endl;
    }
    cout << "Fila" << i + 1 << "procesada\n";
}

```

Capítulo 6

Implementación en la FPGA

Una vez entendido el funcionamiento, prodecemos a implementarlo en lenguaje VHDL a través de cuatro módulos principales, cada uno correspondiente a las etapas en las que se divide el algoritmo, y siguiendo el orden representado en la Figura 6.1. Para facilitar el entendimiento de las explicaciones y que queden más completas, estas irán acompañadas de fragmentos de código que han sido extraídos directamente de Vivado.

Estrategia de Diseño y Modularidad

Antes de detallar el funcionamiento de cada etapa, cabe destacar una decisión de diseño arquitectónico aplicada a todo el proyecto. Para mantener el código principal limpio, legible y escalable, todas las declaraciones de los núcleos IP generados por Vivado (operadores aritméticos en punto flotante, memorias BRAM y FIFOs) se han separado y agrupado en un único paquete de VHDL personalizado llamado `ip_components_pkg.vhd`.

El uso de esta librería compartida evita repetir código y hace más sencilla la instanciación de los componentes en los distintos módulos del *pipeline*, dotando al proyecto de una estructura ordenada y fácil de mantener.

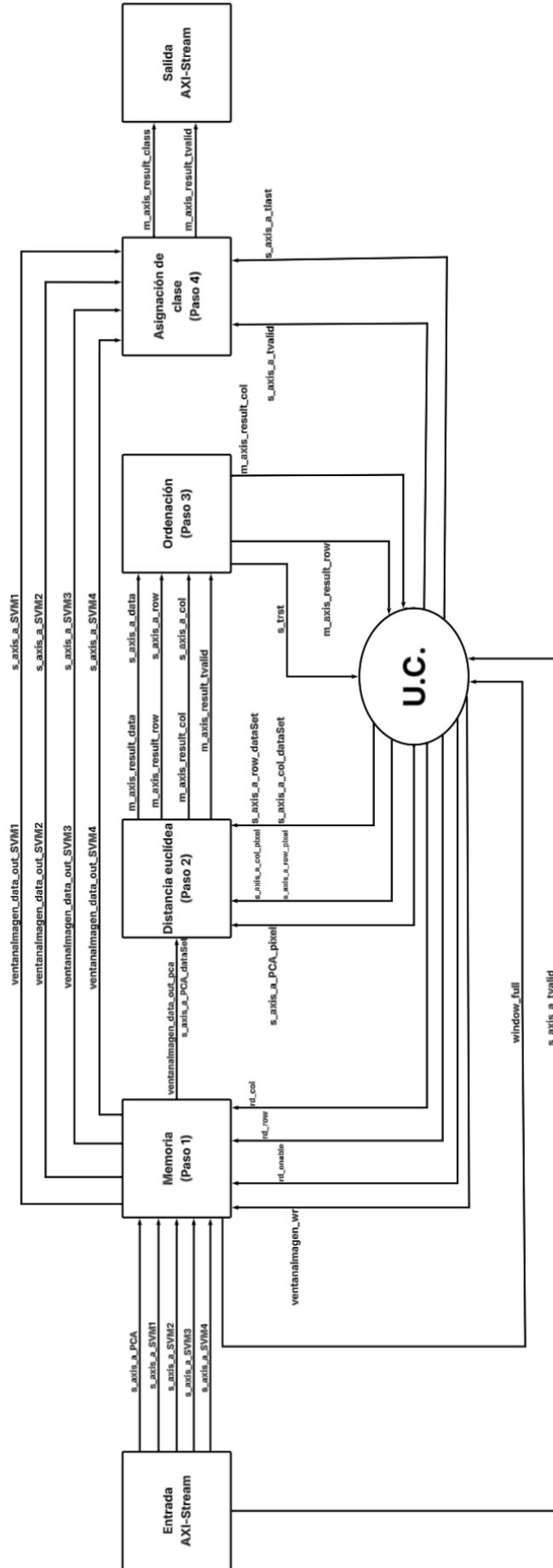


Figura 6.1: Cadena de montaje del algoritmo K-NN para la clasificación de píxeles.

6.1. Paso 1 (Memoria):

En primer lugar se elegirá la ventana de trabajo o *dataset*, tomando una porción de la imagen original para restringir la elección de los posibles vecinos a un número mucho más reducido de píxeles. Esta reducción es importante para hacer de la clasificación un proceso viable, ya que almacenar una imagen hiperespectral médica completa en la memoria interna de la FPGA consumiría una cantidad elevada de recursos.

Para llevarlo a cabo, se ha diseñado un sistema de gestión de memoria basado en una arquitectura de búfer circular (*Circular Line Buffer*). Este sistema se divide en dos submódulos: un componente de almacenamiento físico y un controlador que organiza el flujo de los datos.

Almacenamiento Físico: `ventanaimagen_buffer.vhd`

Este submódulo define la estructura de memoria donde se alojan temporalmente los píxeles. En lugar de utilizar un único bloque de memoria, la arquitectura divide el almacenamiento declarando múltiples BRAMs (bloques RAM) en paralelo, asignando físicamente una BRAM independiente para cada fila del *dataset*.

Este diseño permite que el sistema acceda a cualquier fila en un único ciclo de reloj. Para lograr esta estructura paralela, se aprovechan las sentencias de generación concurrente integradas directamente en el propio lenguaje VHDL (`for ... generate`), lo que permite establecer el tamaño de la ventana a través de un parámetro modificando únicamente un valor genérico. Además, se ha incorporado un registro de retardo (`rd_row_previous`) para compensar la latencia de un ciclo de reloj a las memorias BRAM, garantizando que el multiplexor de salida seleccione el dato correcto en el momento exacto:

```
-- Instanciación de memorias BRAM independientes por fila
gen_rows : for i in 0 to window_size generate
    bram_inst : BRAM_32x512
        port map (
            clka => clk,
            ena  => '1',
            wea  => wea_array(i), -- Habilitación para la fila i
            addra => wr_col,
            dina  => wr_data,
            clk_b => clk,
            enb   => rd_en,
            addrb => rd_col,
            doutb => dout_array(i)
        );
end generate;
```

Controlador Principal: `ventanaImagen.vhd`

Este submódulo actúa como el cerebro del sistema de memoria. Su función principal es recibir el flujo continuo (*stream*) de la imagen y distribuirlo adecuadamente. Dado que para cada píxel se necesita evaluar tanto su valor del algoritmo PCA como sus cuatro probabilidades de clase (obtenidas del algoritmo SVM), el controlador instancia cinco copias del almacenamiento físico descrito en el punto anterior (`mem_pca`, `mem_svm1`, `mem_svm2`, `mem_svm3` y `mem_svm4`), operando con todas ellas de forma paralela.

El controlador gestiona internamente unos contadores de fila y columna que avanzan de forma automática con la llegada de cada píxel válido. El comportamiento de búfer circular se consigue al hacer que el contador de filas se reinicie a cero tras alcanzar el límite inferior de

la ventana, sobrescribiendo en tiempo real la información de las filas más antiguas que ya han quedado fuera del área de influencia del clasificador K-NN.

Un aspecto crítico en arquitecturas segmentadas (*pipeline*) es la sincronización de las etapas. El algoritmo de clasificación no puede comenzar a evaluar distancias hasta que el área de búsqueda inicial esté completamente cargada. Por ello, el controlador genera una bandera de control (`window_full`) que solo se activa cuando los contadores alcanzan los límites máximos de la ventana, indicando al resto del sistema que los datos ya son válidos para ser procesados:

```
-- Señal de control para indicar que el buffer esta cargado
window_full <= '1' when (wr_row_cnt = window_size)
                    and (wr_col_cnt = cols-1) else '0';
```

El aspecto más destacable de este controlador en la fase de lectura es su mecanismo de indexación dinámica. El módulo K-NN solicita los píxeles utilizando sus coordenadas reales con respecto a la imagen original completa. Para evitar que el clasificador deba calcular qué porción de la imagen está cargada actualmente en la FPGA, el controlador traduce la coordenada real a una dirección física de la memoria BRAM mediante una operación aritmética de módulo:

```
-- Traducción de fila real de la imagen a la fila de la ventana
rd_row_int <= to_integer(unsigned(rd_row));
rd_row_win_int <= rd_row_int mod (window_size + 1);
```

Gracias a esta arquitectura, el módulo de cálculo de distancias opera como si tuviera acceso instantáneo a toda la imagen. Simultáneamente, el hardware optimiza la memoria interna de la FPGA almacenando únicamente los píxeles que se necesitan en cada momento.

6.2. Paso 2 (Distancia Euclídea):

Una vez el *dataset* está definido, se procede a calcular la distancia euclídea entre cada píxel de la ventana anterior y el píxel p a clasificar. Para llevar a cabo esta operación de forma segmentada, se ha diseñado una arquitectura *pipeline* (ilustrada en la Figura 6.2) compuesta por tres restadores, tres multiplicadores, dos sumadores, un conversor de formato numérico y memorias de sincronización FIFO (*First-In, First-Out*).

Para darle al sistema una mayor flexibilidad, el módulo se puede parametrizar mediante declaraciones `generic`, permitiendo ajustar el ancho de bus de las coordenadas espaciales (`num.bit_row` y `num.bit_col`) según la resolución de la imagen.

El aspecto más crítico de este módulo es la diferencia de latencias que existe entre la presencia de aritméticas de distintos tipos. Los dos primeros caminos operan con las distancias espaciales (filas y columnas) mediante números enteros, mientras que el tercero procesa la información espectral (PCA) en punto flotante. Dado que las operaciones en punto flotante son más lentas, los datos espaciales terminarían su cálculo varios ciclos antes.

Encapsulamiento AXI-Stream

Para gestionar esta diferencia de tiempos en el camino espacial, se ha aplicado una técnica de diseño basada en envoltorios (*wrappers*). Los núcleos IP aritméticos generados por Vivado (restadores, sumadores y multiplicadores de enteros) poseen una latencia fija, pero carecen de las señales de control necesarias para integrarse en un flujo de datos como el del algoritmo K-NN.

Para solucionar esto, se han diseñado submódulos superiores (como `sub32Signed_axi_wrapper.vhd`) entre otros, que instancian el operador matemático y le añaden una lógica de control basada en registros de desplazamiento (*shift registers*). Esta lógica retrasa la señal de validez (`tvalid`) exactamente el mismo número de ciclos que tarda el operador en realizar el cálculo:

```
-- Logica interna del wrapper para sincronizar la senal tvalid
process(clk)
begin
    if rising_edge(clk) then
        -- Desplazamiento del pipeline de validez segun la latencia
        valid_pipeline(valid_pipeline'high downto 1)
        <= valid_pipeline(valid_pipeline'high-1 downto 0);

        -- Captura de nueva entrada valida
        if s_axis_a_tvalid = '1' and s_axis_b_tvalid = '1' then
            valid_pipeline(0) <= '1';
        else
            valid_pipeline(0) <= '0';
        end if;
    end if;
end process;
m_axis_result_tvalid <= valid_pipeline(valid_pipeline'high);
```

Sincronización Final

Para evitar un desajuste temporal, la arquitectura realiza una suma parcial de las componentes espaciales, la convierte a punto flotante y la introduce en una memoria FIFO que actúa como buffer de retardo. La necesidad de realizar esta conversión (mediante el componente `convSignedAPF_i`) radica en la naturaleza del sumador final (`adder32PF`). Puesto que el hardware no puede sumar directamente un valor entero con uno en punto flotante, este paso actúa como puente para unificar los formatos antes de obtener la distancia total. Esta sincronización se gestiona conectando la señal de lectura (`rd_en`) a la validez del resultado PCA (`expPCA_result_tvalid`), garantizando que ambos valores lleguen al sumador final en el mismo ciclo:

```
-- Buffer de retardo que alinea el camino entero con el de PF
fifoSigned32x32_i : fifo32x32
PORT map (
    clk      => clk,
    din      => convSignedAPF_result_tdata,
    wr_en    => convSignedAPF_result_tvalid,
    rd_en    => expPCA_result_tvalid, -- Sincronizacion con FP
    dout     => salida_FIFO,
    full     => open,
    empty    => open
);
```

Adicionalmente, el diseño debe asegurar que cada distancia calculada siga asociada a su píxel de origen, pues este será necesario para los pasos siguientes a la hora de hacer la clasificación. Para lograr esto, mientras se realiza el cálculo matemático, el módulo introduce las coordenadas originales y la señal de final de trama (`tlast`) en tres memorias FIFO paralelas de mayor profundidad (`fifoRow`, `fifoCol` y `fifoLast`).

Estas memorias actúan como colas de espera que retienen temporalmente la identidad del píxel. Al utilizar la señal de validez del cálculo final (`resultadoFinal_tvalid_i`) como habilitador de lectura simultáneo para todas ellas, se asegura que la distancia salga del módulo emparejada con sus coordenadas y su bit de control. Esta implementación es importante para que la etapa de ordenación identifique correctamente a los vecinos y los acumuladores sepan cuándo detener la suma:

```
-- Ejemplo de sincronizacion de coordenadas y senal de control (tlast)

fifoRow: fifo32x64
  port map(
    clk    => clk,
    din    => s_axis_a_row_dataSet,
    wr_en  => s_axis_a_tvalid,      -- Escribe al entrar el dato
    rd_en  => resultadoFinal_tvalid_i, -- Lee al terminar el calculo
    dout   => m_axis_result_row,
    full   => open,
    empty  => open
  );

fifoCol: fifo32x64
  port map(
    clk    => clk,
    din    => s_axis_a_col_dataSet,
    wr_en  => s_axis_a_tvalid,      -- Escribe al entrar el dato
    rd_en  => resultadoFinal_tvalid_i, -- Lee al terminar el calculo
    dout   => m_axis_result_col,
    full   => open,
    empty  => open
  );

fifoLast: fifo1x64
  port map(
    clk    => clk,
    din    => FIFO_last_in, -- Contiene s_axis_a_tlast
    wr_en  => s_axis_a_tvalid,
    rd_en  => resultadoFinal_tvalid_i,
    dout   => FIFO_last_out,
    full   => open,
    empty  => open
  );
```

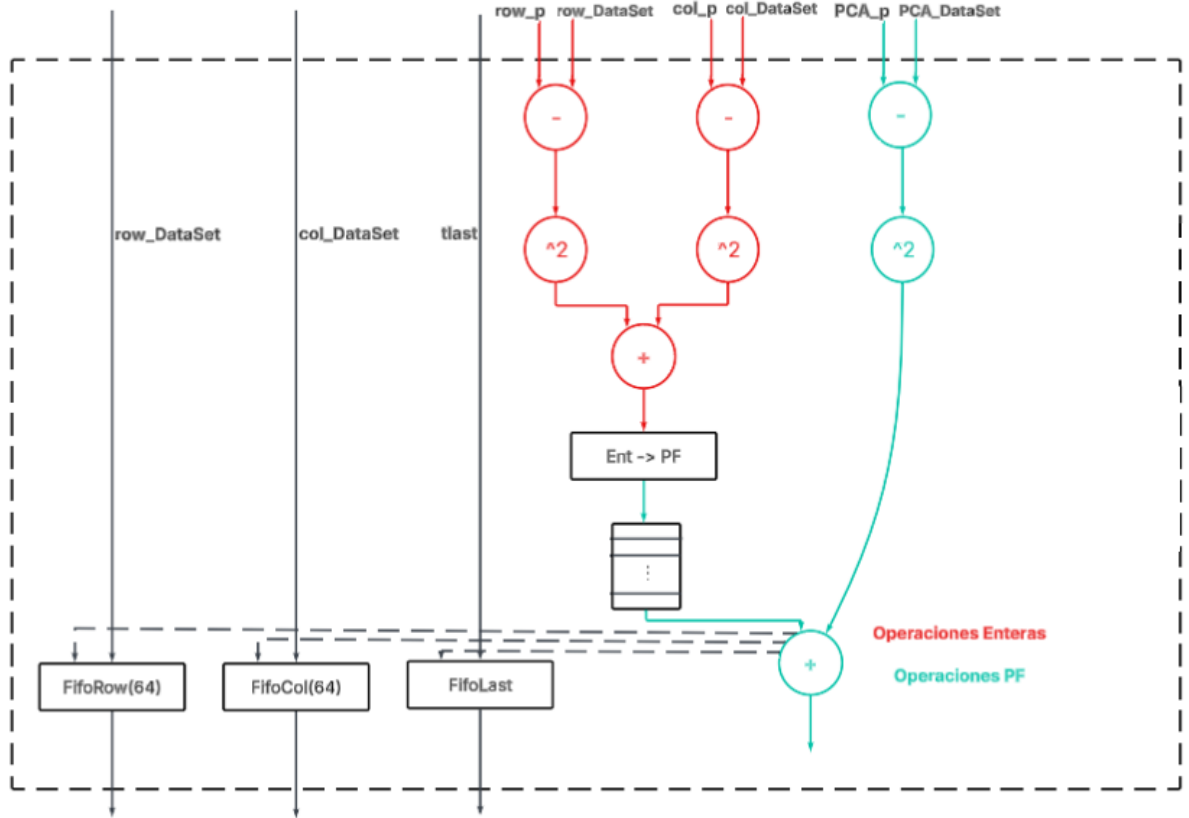


Figura 6.2: Módulo que calcula la distancia euclídea entre p y cada píxel del *dataset*.

6.3. Paso 3 (Ordenación):

Una vez calculadas las distancias euclídeas, el siguiente objetivo es identificar los $k = 40$ vecinos más cercanos al píxel p que se está clasificando. Para realizar esta búsqueda de forma continua y sin detener el flujo de datos, se ha diseñado una arquitectura basada en una cadena de módulos en cascada.

Al igual que en el cálculo de la distancia euclídea, el sistema utiliza parámetros genéricos (**generic**) para mayor escalabilidad. De este modo, la cantidad de vecinos a retener (**num_vecinos**) puede modificarse fácilmente, adaptando el clasificador K-NN a diferentes configuraciones sin alterar la lógica de este.

Estructura de Ordenación en Cascada: ordenacion.vhd

Este módulo actúa como un envoltorio (*wrapper*) estructural que orquesta la cadena. Para lograr encontrar exactamente a los 40 vecinos más cercanos, el diseño encadena 40 copias independientes del submódulo (**celda_minimo**).

Al igual que en la etapa de memoria, se emplean sentencias de generación (**for ... generate**) combinadas con vectores de señales (**array**) que actúan como buses de interconexión interna. De esta forma, los puertos de salida de la celda i se conectan de manera directa a los puertos de entrada de la celda $i + 1$, propagando así la información de las distancias, sus coordenadas espaciales y las señales de control:

```

-- Generacion de N instancias de celda_minimo conectadas en cascada
gen_minimos : for i in 0 to num_vecinos-1 generate
    celda_inst : entity work.celda_minimo
        port map (
            aclk => aclk,
            s_axis_a_data => data_sig(i), -- Entrada etapa i
            -- (Mapeo analogo para coordenadas, tvalid, tlast, trst)
            m_axis_result_data => data_sig(i+1), -- Salida etapa i+1
            -- (Mapeo analogo para coordenadas, tvalid, tlast, trst)
        );
    end generate gen_minimos;

```

Unidad de Comparación y Almacenamiento: celda_minimo.vhd

Este submódulo es la unidad lógica fundamental de la ordenación y actúa como un filtro. Internamente, la celda posee registros que almacenan la distancia mínima que ha procesado hasta el momento (`actual_data`), junto con sus coordenadas espaciales (`actual_row` y `actual_col`).

Un requisito necesario en esta etapa es poder “limpiar” la memoria cuando se finaliza la clasificación de un píxel y se comienza con el siguiente. Para ello, si la celda recibe una señal de reinicio (`s_axis_a_trst = '1'`), el registro interno vuelve a inicializarse a su valor máximo posible.

Cuando un nuevo píxel candidato llega a la celda, su distancia se compara con el valor almacenado mediante un comparador combinacional. Dependiendo de este resultado (`comp`), el hardware toma una de estas dos decisiones en el siguiente ciclo de reloj:

- **Inserción y desplazamiento** (`comp = '1'`): Si la distancia del candidato entrante es menor, la celda lo guarda en sus registros (actualizando su mínimo local) y expulsa el valor que tenía previamente, empujándolo hacia la salida para que sea evaluado por la siguiente celda.
- **Paso a través** (`comp = '0'`): Si la distancia del candidato es mayor o igual, el mínimo local se mantiene en el registro, y el candidato simplemente atraviesa la celda.

Esta lógica condicional, que garantiza que las coordenadas espaciales siempre viajen emparejadas con su distancia euclídea, se describe de la siguiente manera:

```

-- Evaluacion del comparador combinacional
comp <= '1' when (s_axis_a_data < actual_data) else '0';

-- Logica sincrona de actualizacion, reseteo y propagacion
if (s_axis_a_trst = '1') then
    actual_data <= ('0' & (31-1 downto 0 => '1')); -- Reseteo al max
elsif (s_axis_a_tvalid = '1' and comp = '1') then
    -- El nuevo valor es menor: Se almacena y se expulsa el antiguo
    actual_data <= s_axis_a_data;
    m_axis_result_data <= actual_data;
    -- (Propagacion analoga para las coordenadas)
elsif (s_axis_a_tvalid = '1' and comp = '0') then
    -- El nuevo valor es mayor o igual: Atraviesa la celda
    m_axis_result_data <= s_axis_a_data;
end if;

```

Gracias a esta topología, los valores más pequeños quedan “atrapados” en las primeras celdas de la cadena, mientras que los valores mayores van siendo empujados hacia la derecha. El sistema garantiza que cualquier valor que logre salir por la última celda (la k -ésima) será definitivamente descartado, asegurando que al finalizar el procesamiento de la ventana, las 40 celdas contendrán los vecinos más influyentes ordenados de menor a mayor distancia.

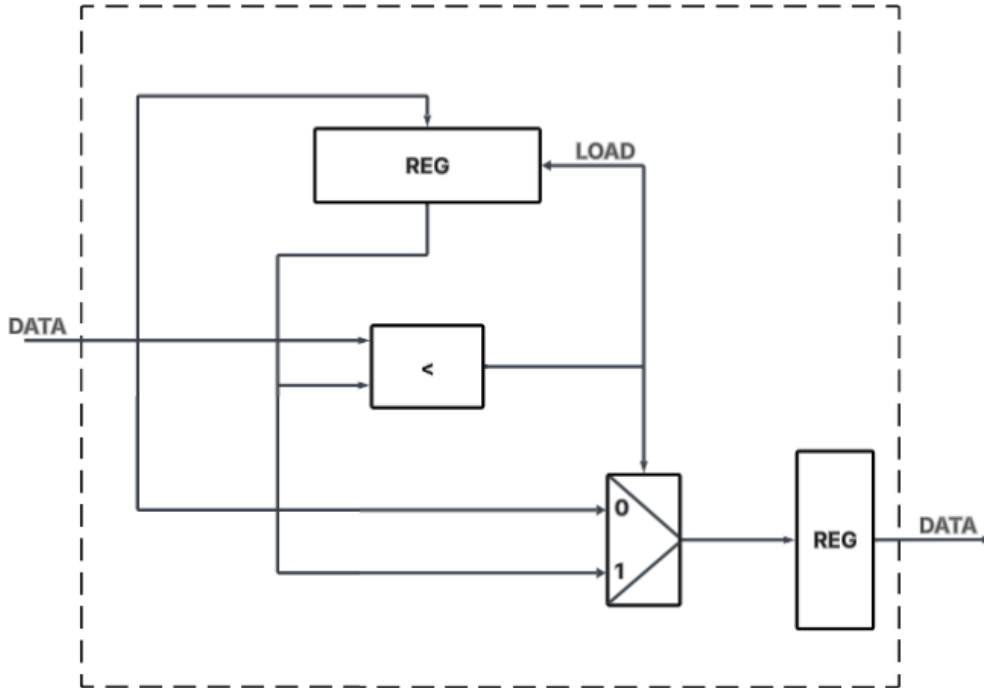


Figura 6.3: Módulo de comparación `celda_minimo.vhd`.

6.4. Paso 4 (Asignación de Clase):

Tras identificar a los vecinos más cercanos, la etapa final consiste en determinar a qué tipo de clase pertenece el píxel p . A diferencia de un modelo K-NN estándar que utiliza un sistema de votación por mayoría simple, este diseño implementa una votación ponderada más robusta, sumando las probabilidades de pertenencia a cada clase obtenidas por el algoritmo SVM ejecutado previamente.

Para desarrollar este proceso final, la arquitectura (ilustrado en la Figura 6.4) se encapsula en el submódulo superior `asignacionDeClase.vhd`, el cual coordina internamente dos submódulos secuenciales: una fase de acumulación paralela y un árbol de comparación.

Acumulación de Probabilidades: `acumuladores.vhd`

Este submódulo recibe el flujo de datos de los vecinos seleccionados y realiza el sumatorio de sus probabilidades. Dado que existen cuatro clases posibles de tejido, el diseño instancia cuatro unidades de acumulación en punto flotante (`acc32PF`) que operan paralelamente, procesando las señales `SVM1` a `SVM4` de forma simultánea.

Un aspecto fundamental del control hardware en esta etapa es saber exactamente cuándo detener la acumulación y evaluar el resultado definitivo. Para ello, el diseño aprovecha la señal de finalización de trama (`tlast`). Cuando el cuadragésimo vecino termina de procesarse, esta

bandera se activa a la salida de los acumuladores. De esta forma, el módulo superior conecta esta señal (`tlast`) directamente al puerto de validez (`tvalid`) de la siguiente etapa. Esto garantiza que los comparadores ignoren las sumas parciales y solo se activen en el ciclo exacto en el que la suma total ha concluido. Además, se hace uso de cuatro registros (`REG1`, `REG2`, `REG3`, `REG4`) que almacenan los resultados obtenidos por los acumuladores, garantizando que la fase de comparación que viene a continuación se haga sobre los valores correctos.

Árbol de Decisión: `comparadores.vhd`

Una vez validados los cuatro valores acumulados finales, el sistema debe encontrar cuál es el máximo absoluto, ya que este indicará la clase definitiva del píxel. Para minimizar la latencia y acortar la ruta combinacional, el hardware no compara todos los valores a la vez, sino que utiliza una topología de árbol binario de comparación en dos niveles lógicos:

- **Primer nivel (Eliminatorias):** Se instancian dos comparadores en paralelo. El primero evalúa la Clase 1 frente a la Clase 2 (`cmp1_2`), mientras que el segundo evalúa la Clase 3 frente a la Clase 4 (`cmp3_4`). Mediante lógica de multiplexación, el sistema selecciona y propaga a los dos ganadores (`max_1_2` y `max_3_4`).
- **Segundo nivel (Final):** Un tercer y último comparador (`cmpFinal`) evalúa a los dos ganadores del nivel anterior para decidir cuál es el valor probabilístico más alto de todos.

El resultado de estas comparaciones produce tres banderas de un solo bit (el bit menos significativo de cada salida del comparador). Estas tres señales combinadas contienen toda la información necesaria para determinar el ganador. En VHDL, esta última decodificación se resuelve mediante lógica combinacional, que lee dichas banderas y asigna una etiqueta de 2 bits ("00", "01", "10" o "11") correspondiente a la clase final del píxel:

```
-- Lógica de decodificación de clase basada en los flags del árbol
m_axis_result_tclass <=
  "00" when (cmp_final(0)='1' and cmp_1_2(0)='1') else -- Clase 1
  "01" when (cmp_final(0)='1' and cmp_1_2(0)='0') else -- Clase 2
  "10" when (cmp_final(0)='0' and cmp_3_4(0)='1') else -- Clase 3
  "11"; -- Clase 4
```

Con la asignación de este valor de 2 bits a la salida del sistema, concluye íntegramente el *pipeline* hardware del algoritmo K-NN, logrando clasificar el píxel hiperespectral de forma determinista y altamente paralelizada.

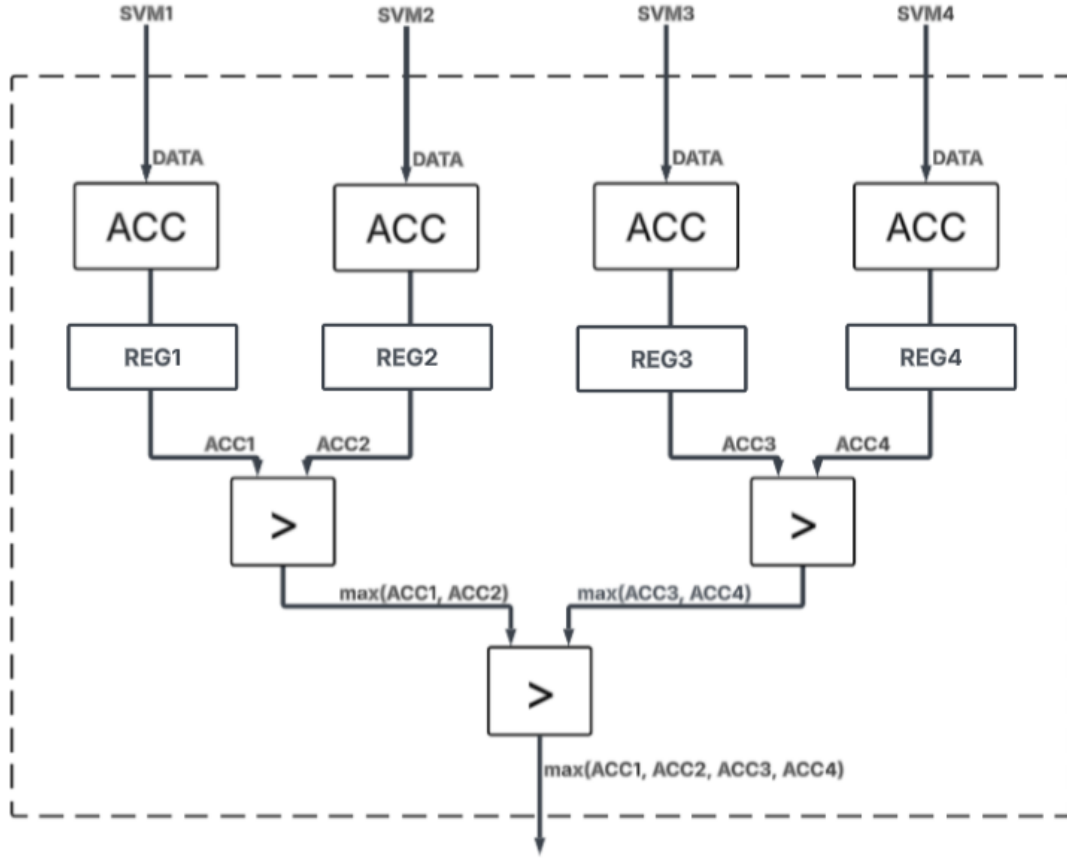


Figura 6.4: Módulo que calcula la asignación final de la clase al píxel p .

6.5. Unidad de Control como Máquina de Estados

Para estructurar el correcto funcionamiento del sistema y la sincronización de las señales entre los módulos que ya hemos visto de la implementación (Memoria, Distancia Euclídea, Ordenación y Asignación de Clase), la Unidad de Control (UC) se ha diseñado como una **Máquina de Estados Finita** (FSM o *Finite State Machine*).

Esta FSM se encarga de dirigir el flujo de datos píxel a píxel, asegurando que cada módulo reciba la información en el momento adecuado. El ciclo de procesamiento sigue un flujo secuencial lógico que se repite para cada píxel de la imagen hasta alcanzar el final de la misma. Lo podemos apreciar en la Figura 6.5, donde las transiciones sin señal indican que se pasa de un estado a otro directamente. Las transiciones entre los estados principales siguen este orden:

1. **Inicialización:** (`idle` $\rightarrow$ `write_window`)
2. **Lectura del píxel:** (`read_center` $\rightarrow$ `store_center`)
3. **Recorrido de la ventana espacial:** (`read_window` $\rightarrow$ `last_windows_pixel`)
4. **Extracción de vecinos:** (`wait_ordenacion` $\rightarrow$ `push_ordenacion`)
5. **Lectura de características SVM:** (`read_vecinos_first` $\rightarrow$ `read_vecinos` $\rightarrow$ `last_read_vecinos`)
6. **Clasificación y avance:** (`wait_clasificacion` $\rightarrow$ `next_pixel` / `completar_fila`)
7. **Fin de ejecución:** (`done`)

Para comprender de forma más detallada el comportamiento interno del sistema, a continuación se describe brevemente la función que desempeña cada uno de estos estados dentro de la unidad de control:

- **idle y write_window:** Son los estados iniciales. El sistema se mantiene escribiendo el flujo de datos de entrada en el buffer de memoria local hasta que se llena la ventana espacial inicial (`window_full = '1'`).
- **read_center y store_center:** La UC solicita a la memoria la información completa del píxel central. Aunque la memoria devuelve tanto la componente principal (PCA) como las probabilidades SVM, el sistema solo almacena temporalmente el valor del PCA. Esto se debe a que dicho valor servirá como referencia para el cálculo de las distancias euclídeas con el resto de la ventana espacial en las etapas posteriores.
- **read_window y last_windows_pixel:** Constituyen el bucle de cálculo principal. La UC recorre iterativamente todos los píxeles adyacentes de la ventana y los envía de forma continua al módulo de Distancia Euclídea. Al enviar el último píxel de la ventana, se activa la señal `tlast`.
- **wait_ordenacion:** El sistema pausa la lectura de memoria y espera a que el módulo de ordenación termine de recibir y procesar todas las distancias generadas.
- **push_ordenacion:** Introduce valores nulos temporales en el módulo de ordenación. Esta técnica sirve para “empujar” los datos internos y obligar al módulo a expulsar las coordenadas espaciales de los k vecinos con menor distancia euclídea.
- **read_vecinos_first, read_vecinos y read_vecinos_last:** Utilizando las coordenadas extraídas en el paso anterior, la UC consulta la memoria nuevamente, esta vez para recuperar las probabilidades SVM de esos vecinos concretos, enviándolas progresivamente al módulo de Asignación de Clase.
- **wait_clasificacion:** Estado de pausa en el que la UC espera a que el módulo de asignación finalice el recuento de votos y emita la predicción final de la clase (`m_axis_result_tvalid = '1'`).
- **next_pixel y completar_fila:** Tras emitir el resultado, estos estados actualizan las coordenadas del píxel p desplazándolo hacia la derecha. Si se cambia de fila, `completar_fila` gestiona la entrada de nuevos píxeles en la ventana de memoria. Una vez actualizado, el flujo vuelve a `read_center`.
- **done:** Estado final que se alcanza cuando las coordenadas de procesamiento igualan los límites máximos de la imagen, indicando que el hipercubo ha sido clasificado por completo.

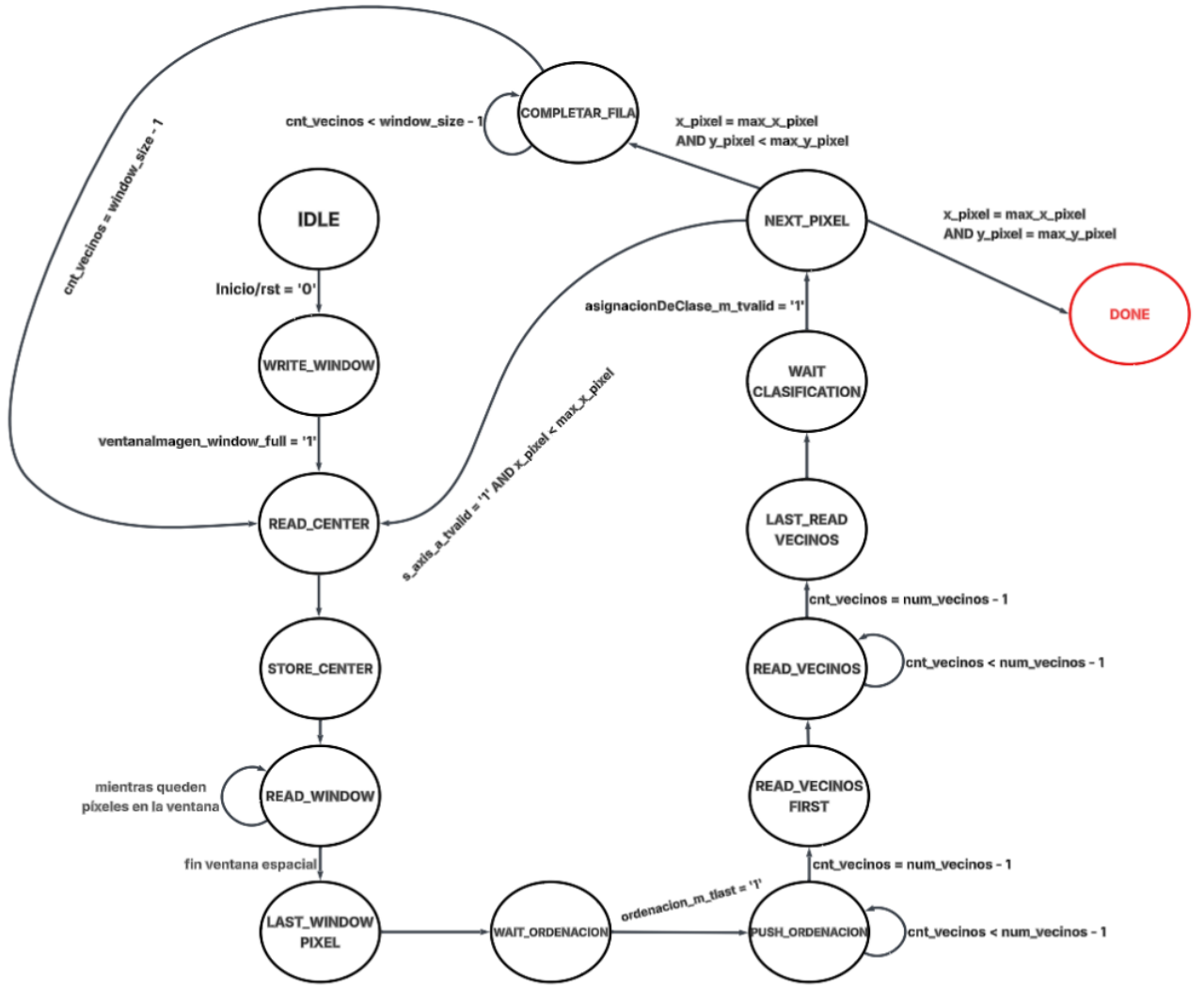
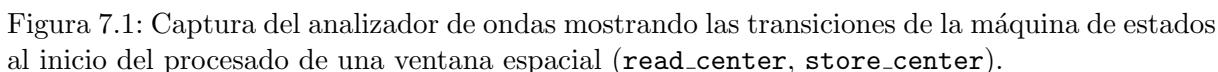


Figura 6.5: Diagrama de la Máquina de Estados Finita implementada en la Unidad de Control, desde la inicialización (IDLE) hasta la clasificación de todos los píxeles de la imagen (DONE).

Resultados experimentales

A continuación, se presentan los resultados obtenidos tras la síntesis, implementación y simulación del clasificador K-NN en el entorno de desarrollo Vivado, utilizando un banco de pruebas (*testbench*) con datos extraídos de imágenes médicas reales.



Adaptación de formatos inicial

Dado que los archivos extraídos del modelo en software contienen números decimales reales en formato texto, pero nuestra arquitectura hardware procesa la información en vectores de bits estandarizados, es necesario realizar una conversión previa. Para ello, el banco de pruebas hace uso de la librería estándar `ieee.float_pkg.all` e implementa una función específica que transforma las variables de tipo `real` en un vector lógico de 32 bits (`std_logic_vector`) en punto flotante, siguiendo el estándar IEEE 754 de precisión simple:

```
function real_to_slv(x : real) return std_logic_vector is
    variable f : float32;
begin
    f := to_float(x, 8, 23);
    return to_slv(f);
end function;
```

Lectura de los datos de entrada

El proceso encargado de generar los estímulos (`stim_proc`) para el *testbench* lee secuencialmente los ficheros de texto que contienen la imagen médica preprocesada (`Output_PCA` y `Output_SVM`). Este proceso asegura que el hardware recibe los datos respetando el protocolo de comunicación AXI Stream, esperando a que el módulo K-NN esté listo (`s_axis_a_tready = '1'`) antes de validar y enviar un nuevo píxel.

```
-- Leer PCA y SVMs desde los archivos .txt
readline(pca_file, Lpca);
read(Lpca, v_pca);
readline(svm_file, Lsvm);
read(Lsvm, v_svm1);
read(Lsvm, v_svm2);
read(Lsvm, v_svm3);
read(Lsvm, v_svm4);

-- Sincronizacion AXI Stream para nueva lectura
wait until rising_edge(clk) and s_axis_a_tready = '1';

s_axis_a_PCA    <= real_to_slv(v_pca);
s_axis_a_SVM1   <= real_to_slv(v_svm1);
s_axis_a_SVM2   <= real_to_slv(v_svm2);
s_axis_a_SVM3   <= real_to_slv(v_svm3);
s_axis_a_SVM4   <= real_to_slv(v_svm4);

s_axis_a_tvalid <= '1';
wait until rising_edge(clk);
s_axis_a_tvalid <= '0';
```

Comprobación del resultado final

Finalmente, para garantizar el correcto funcionamiento de la arquitectura sin tener que revisar miles de señales en el analizador de ondas, se ha implementado un proceso comprobador (`check_proc`). Este módulo lee el fichero *Golden Reference* (`Output_kNN.txt`), que contiene las predicciones generadas por el algoritmo en C++.

Cada vez que el hardware emite un resultado válido (`m_axis_result_tvalid = '1'`), el *testbench* lo compara con el valor esperado, y si detecta que son diferentes, es decir, no ha predicho

la clase como debería, el simulador detiene la ejecución lanzando un error de tipo (`severity failure`).

Al no haberse reportado errores, se puede confirmar que el diseño hardware replica a la perfección el comportamiento del algoritmo original.

```
while not endfile(golden_file) loop
    wait until rising_edge(clk);

    if m_axis_result_tvalid = '1' then
        readline(golden_file, L);
        read(L, expected);
        expected := expected - 1;

        if to_integer(unsigned(m_axis_result_tdata)) /= expected then
            report "ERROR" & integer'image(out_count) &
                "esperado=" & integer'image(expected) &
                "obtenido=" &
                integer'image(to_integer(unsigned(m_axis_result_tdata)))
                severity failure;
            errores := errores + 1;
        else
            report "OK" & integer'image(out_count);
        end if;
    end if;
end loop;
```

7.2. Análisis de Rendimiento y Tiempo de Procesamiento

Uno de los objetivos de este trabajo era demostrar que la aceleración hardware permite acercar esta tecnología al tiempo real, mejorando el cuello de botella que aparecía al utilizar software tradicional en la detección de tumores.

Para poder interpretar correctamente los resultados temporales de la simulación, es fundamental establecer primero la frecuencia de operación del diseño. En este caso, el *testbench* ha sido configurado para utilizar una señal de reloj (*clk*) con una frecuencia de **100 MHz**, lo que equivale a un periodo de 10 nanosegundos (ns) por cada ciclo de reloj. Esta frecuencia es un estándar realista y muy habitual en los diseños para dispositivos FPGA.

Teniendo este dato en cuenta vemos que, como indica la consola de simulación de la Figura 7.3, el tiempo total simulado para procesar la imagen completa, cuyo tamaño es de 496 filas x 442 columnas, asciende a aproximadamente **780.7 ms** (más concretamente, el proceso finaliza en la marca de los 780.703.055 ns). Si relacionamos esto con la frecuencia de operación dividiendo el tiempo total entre los 10 ns de cada periodo, deducimos que a la arquitectura le ha tomado alrededor de 78 millones de ciclos de reloj procesar y clasificar la totalidad de los píxeles del hipercubo.

Lograr procesar un volumen tan masivo de datos médicos en menos de un segundo confirma que la arquitectura propuesta es viable para asistir al cirujano de forma fluida y continua en un quirófano, cumpliendo así con el objetivo principal que tenía este trabajo.

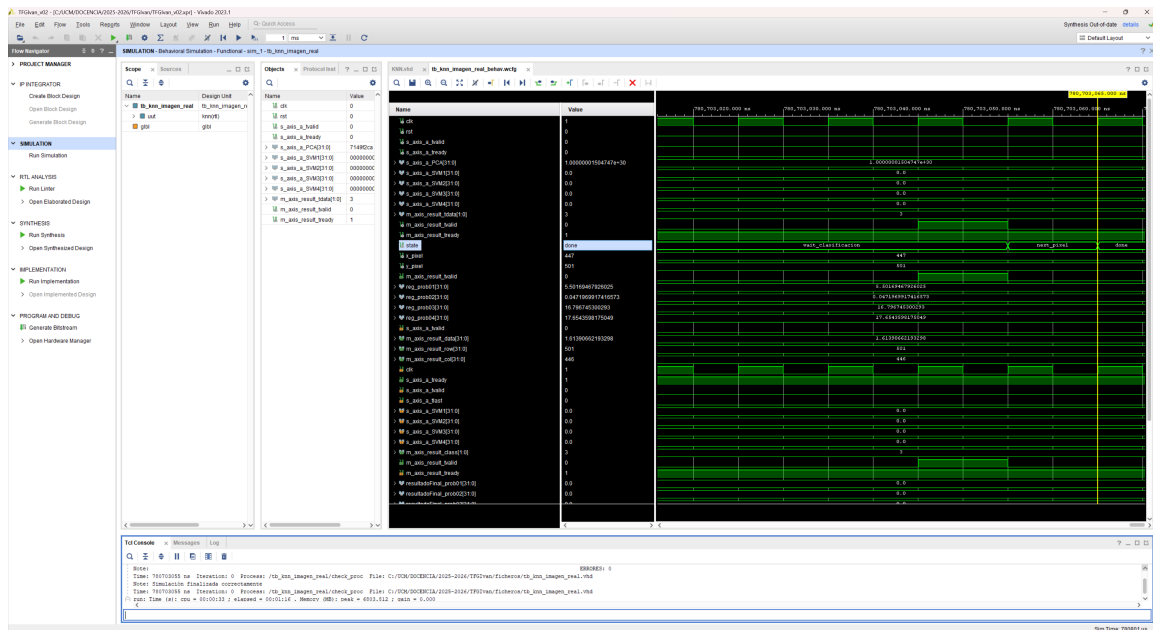


Figura 7.3: Vista del entorno de simulación de Vivado tras finalizar el procesamiento de la imagen. En la consola Tcl inferior se registra un tiempo total de simulación de 780.7 ms.

7.3. Análisis Temporal del Circuito

En el diseño digital, no basta con que un circuito funcione a nivel lógico; las señales eléctricas deben propagarse por los transistores lo suficientemente rápido como para cumplir con las restricciones de reloj del sistema interno.

Como se puede comprobar en la Figura 7.4, el reporte de tiempos (*Design Timing Summary*) generado tras la síntesis del diseño en la FPGA muestra resultados satisfactorios. El parámetro conocido como **WNS** (*Worst Negative Slack*), presenta un valor positivo de **1.480 ns**. Cabe recordar que el *Slack* representa el margen de tiempo que tienen los datos para viajar desde un registro origen hasta un registro destino (atravesando la lógica combinacional necesaria) dentro de un mismo ciclo de reloj. En arquitecturas hardware, tener un *Slack* positivo garantiza que el camino crítico del diseño (la ruta más lenta entre dos registros) cumple con los requisitos de *Setup*, evitando así que se produzcan errores de sincronización o cálculos incorrectos debido a los retardos físicos de los componentes.

Además de garantizar la estabilidad funcional, este margen positivo tiene una implicación directa sobre el rendimiento máximo de la arquitectura. Dado que la simulación y la implementación se han configurado con un reloj de 100 MHz (periodo de 10 ns), y sabiendo que sobran 1,480 ns en la ruta más restrictiva, podemos calcular el periodo mínimo teórico soportado por el circuito restando ambos valores ($10\text{ns} - 1,480\text{ns} = 8,520\text{ns}$). Si invertimos este periodo mínimo, obtenemos una frecuencia máxima de operación teórica (F_{max}) aproximada de **117 MHz**.

Esto significa que el diseño no ha alcanzado su límite de rendimiento físico: si los requisitos médicos del sistema lo exigieran, la placa podría configurarse para trabajar a mayor velocidad sin necesidad de modificar el código VHDL, lo que reduciría aún más los 780 ms de tiempo total de clasificación obtenidos en la simulación.

Asimismo, se reporta que no existe ningún *endpoint* fallido (*Number of Failing Endpoints: 0*) sobre un total de 27.253 conexiones analizadas. La combinación de cero fallos de enrutamiento y un amplio margen temporal de seguridad garantiza la estabilidad temporal del diseño.

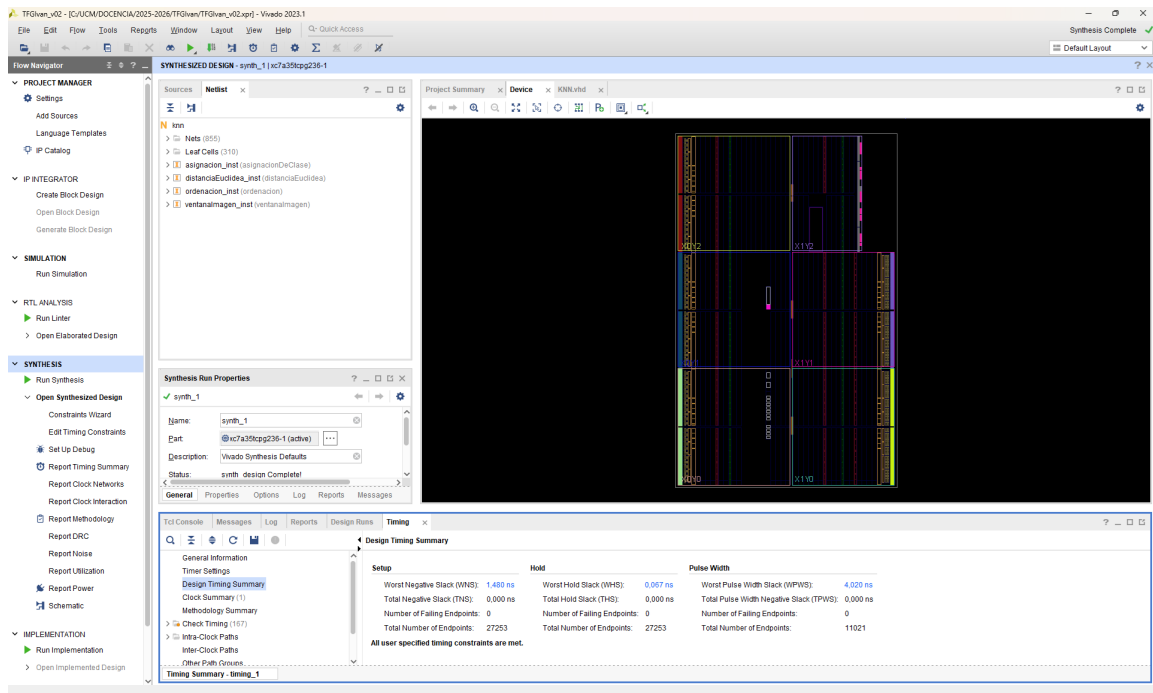


Figura 7.4: Reporte de tiempos tras la implementación del diseño. Se observa un *Worst Negative Slack* de 1.480 ns y cero errores de *Setup/Hold*.

7.4. Implementación y Utilización de Recursos

Finalmente, se ha evaluado cómo se distribuyen físicamente los distintos módulos del diseño dentro de la FPGA. En la Figura 7.6 correspondiente a la vista de dispositivo (*Device View*) de Vivado, es posible observar el mapa de ocupación del chip una vez finalizado el proceso de ubicación y conexión de los elementos (*Place and Route*).

La herramienta colorea de forma distinta las agrupaciones lógicas correspondientes a cada módulo de la arquitectura instanciada en el diseño de nivel superior (**knn**). Se distinguen claramente las regiones asignadas a módulos clave como la **distanciaEuclidea_inst** (responsable de la alta carga aritmética), la **ordenacion_inst** (encargada de mantener los k-vecinos más cercanos) y la **asignacion_inst** (que asigna la clase correspondiente a cada píxel).

Para respaldar esta distribución de forma cuantitativa, se ha analizado el reporte de utilización de recursos extraído de la herramienta. Como se puede observar en la Figura 7.5, a nivel de lógica general, la arquitectura consume un total de 7818 Slice LUTs (lo que representa un 37,59 % de los 20800 disponibles en la placa) y 10361 Slice Registers (un 24,91 % de 41.600). Este consumo moderado indica que el control y la máquina de estados del circuito dejan espacio en la placa para utilizar más en caso de ser necesario.

Hierarchy						
Name	Slice LUTs (20800)	Slice Registers (41600)	Block RAM Tile (50)	DSPs (90)	Bonded IOB (106)	BUFGCTRL (32)
▼ N knn	7818	10361	34.5	23	168	1
> I asignacion_inst (asignacionDeClase)	3470	4376	0	8	0	0
> I distanciaEuclidea_inst (distanciaEuclidea)	1010	1844	2	15	0	0
> I ordenacion_inst (ordenacion)	2619	4047	0	0	0	0
> I ventanaImagen_inst (ventanaImagen)	670	17	32.5	0	0	0

Figura 7.5: Reporte de utilización de recursos post-síntesis extraído de Vivado.

Pasemos a la utilización de los DSPs. Como se explicó en el Capítulo 2, estos componentes son bloques hardware preoptimizados, dedicados específicamente a realizar operaciones aritméticas complejas a gran velocidad. El sistema hace uso de 23 DSPs de los 90 disponibles (un 25,56 %). Como era previsible tras entender el funcionamiento del algoritmo K-NN, gran parte de esta carga recae sobre el módulo `distanciaEuclidea_inst` (pues necesita 15 DSPs para resolver las multiplicaciones y acumulaciones de la fórmula al ser el que contiene más operaciones aritméticas), mientras que el módulo `asignacion_inst` utiliza los 8 restantes para acelerar el recuento y cálculo de las probabilidades SVM. Gracias a la utilización de estos componentes, se consigue reducir la carga matemática del código, evitando así saturar los bloques lógicos de la placa, y resultando en un diseño mucho más compacto y rápido.

Por otro lado, el análisis nos muestra cuál es el recurso más crítico del sistema y la principal limitación de la arquitectura: la memoria interna. El diseño en su totalidad ocupa 34,5 bloques BRAM de las 50 disponibles en una *Basys 3*, lo que supone agotar el 69 %. De este consumo, 32,5 bloques están destinados íntegramente al módulo `ventanaImagen_inst`. Esta alta dependencia implica que el volumen de los datos de entrada es un factor determinante para el límite de la implementación: la cantidad de memoria física disponible restringe el tamaño máximo de la ventana espacial que se puede almacenar y procesar simultáneamente en el chip.

Asimismo, la eficiencia del circuito radica en cómo se han agrupado espacialmente todos estos recursos. Al mantener las celdas lógicas, los bloques aritméticos DSP y la memoria BRAM físicamente adyacentes para cada submódulo, se acorta la longitud de las conexiones internas por las que viaja la señal eléctrica. Esta ubicación estratégica es el factor principal que ha permitido minimizar los retardos de interconexión (*Net Delay*), garantizando que los datos circulen a la velocidad necesaria entre los registros para lograr el *Slack* positivo mencionado en la sección anterior.

En conclusión, los resultados experimentales extraídos de Vivado respaldan el diseño propuesto: la arquitectura es funcionalmente correcta, hace un uso eficiente y justificado de los recursos del silicio, y es lo suficientemente rápida como para habilitar el uso de imágenes hiperespectrales en aplicaciones neuroquirúrgicas en tiempo real.

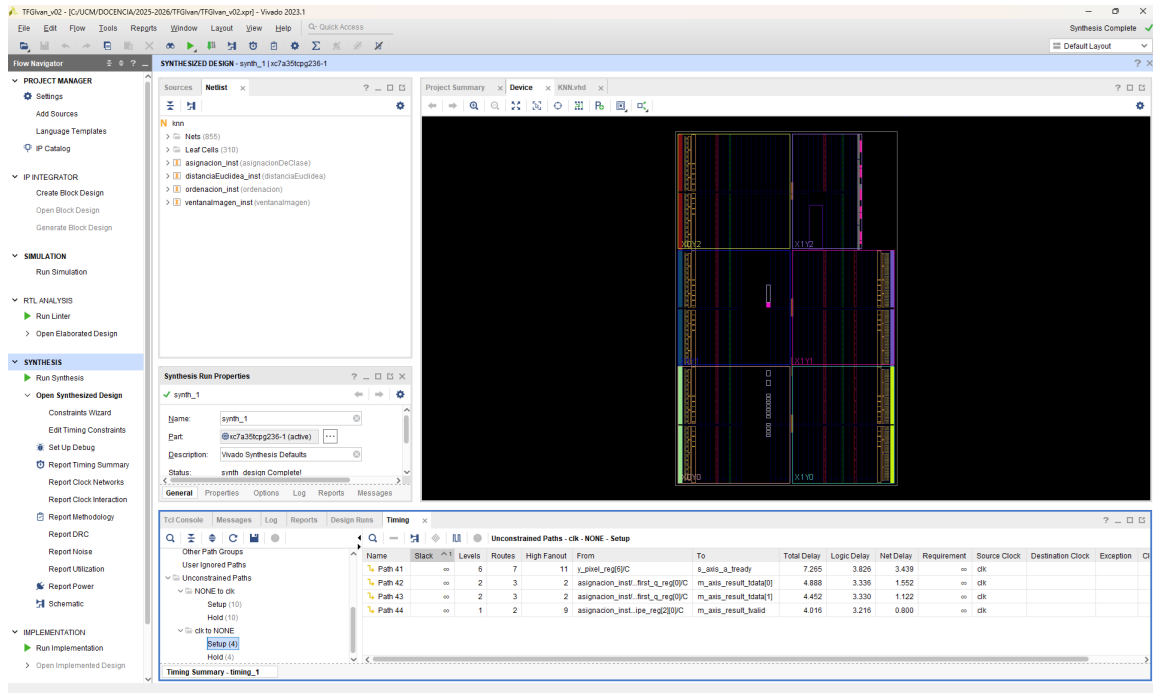


Figura 7.6: Vista de dispositivo mostrando el mapa de ocupación del chip tras el proceso de enrutamiento. Las regiones coloreadas corresponden a los distintos módulos instanciados en el clasificador K-NN.

7.5. Validación Física en la FPGA

Tras haber confirmado el correcto funcionamiento lógico mediante simulación y verificado que el diseño cumple con las restricciones temporales y de recursos, el paso final ha consistido en validar la arquitectura directamente sobre la placa *Basys 3*. Esta prueba certifica que el sistema opera de manera fiable en un entorno real, gestionando las latencias físicas de los componentes.

Para esta validación *on-chip*, se ha diseñado un módulo de nivel superior (`top_knn_fpga`) que actúa como *wrapper* del clasificador. Dado que la placa no tiene acceso directo al sistema de ficheros, la introducción de los datos se realiza mediante bloques BRAMs precargados con datos reales mediante ficheros `.coe`.

Integración de componentes de memoria

El diseño contiene el núcleo K-NN con dos memorias IP Core: una de entrada (`mem_PCA_SVM`) que contiene las características de los píxeles y una de referencia (`mem_KNN`) con las clases de referencia obtenidas tras la ejecución del algoritmo en software. Como se observa en el siguiente fragmento, el bus de salida de la memoria de entrada (160 bits) se segmenta para alimentar las entradas PCA y SVM de la arquitectura:

```
-- Instancia Modulo kNN
uut: entity work.knn
    port map (
        clk => clk, rst => rst,
        s_axis_a_tvalid => bram_data_valid,
        s_axis_a_tready => s_axis_tready,
        s_axis_a_PCA => douta_in(159 downto 128),
        s_axis_a_SVM1 => douta_in(127 downto 96),
        s_axis_a_SVM2 => douta_in(95 downto 64),
        s_axis_a_SVM3 => douta_in(63 downto 32),
```

```

s_axis_a_SVM4    => douta_in(31 downto 0),
m_axis_result_tdata => m_axis_tdata,
m_axis_result_tvalid => m_axis_tvalid,
m_axis_result_tready => '1'
);

```

Control de lectura y compensación de latencia

Un punto importante al probar el sistema en la placa es cómo coordinamos el flujo de datos del protocolo AXI con el tiempo que tarda la memoria en responder. Como la memoria trabaja al ritmo del reloj, no nos entrega el dato al instante, sino que tarda un ciclo en tenerlo listo. Por eso, el proceso `proc_lectura` va pidiendo las direcciones y activa la señal `bram_data_valid` con ese mismo ciclo de retraso. De esta forma, nos aseguramos de que el dato llegue a tiempo, coincidiendo con el momento en que avisamos al clasificador de que ya puede empezar a procesar

```

proc_lectura: process(clk)
begin
    if rising_edge(clk) then
        if rst = '1' then
            addra_in    <= (others => '0');
            bram_data_valid <= '0';
        else
            -- Si el modulo kNN esta listo y hay datos pendientes
            if s_axis_tready = '1' and unsigned(addra_in) < 7072 then
                addra_in <= std_logic_vector(unsigned(addra_in) + 1);
                bram_data_valid <= '1'; -- Compensa latencia 1 ciclo
            else
                bram_data_valid <= '0';
            end if;
        end if;
    end if;
end process;

```

Autocomprobación y señalización mediante LEDs

Finalmente, la arquitectura implementa una lógica de comparación en tiempo real. Por cada predicción emitida por el hardware, se consulta la memoria de validación y se comparan ambos valores. El resultado se comunica al exterior mediante los LEDs de la placa, proporcionando una confirmación visual si el proceso ha tenido éxito:

```

proc_verificacion: process(clk)
begin
    if rising_edge(clk) then
        if rst = '1' then
            addr_knn <= (others => '0');
            led_ok    <= '0';
            led_err    <= '0';
        else
            if m_axis_tvalid = '1' then
                -- Comparacion con Golden Reference en el chip
                if m_axis_tdata = expected_knn then
                    led_ok <= '1';
                else
                    led_err <= '1'; -- Error persistente detectado
                end if;
            end if;
        end if;
    end if;
end process;

```

```
        if unsigned(addr_knn) < 7071 then
            addr_knn <= td_logic_vector(unsigned(addr_knn)+1);
        end if;
    end if;
end if;
end process;
```

Las pruebas físicas han resultado satisfactorias, con la activación constante del diodo `led_ok` tras procesar los píxeles que componen el conjunto de prueba. Estos resultados concluyen la fase de experimentación, demostrando que el diseño no solo es teóricamente robusto, sino plenamente funcional sobre hardware reconfigurable para aplicaciones médicas en tiempo real.

Capítulo 8

Conclusiones y trabajo futuro

8.1. Conclusiones

El presente Trabajo de Fin de Grado surgió con el propósito de abordar una barrera tecnológica crítica en el ámbito médico: el alto coste computacional que impide utilizar imágenes hiperespectrales en tiempo real durante las neurocirugías. Tomando como inspiración el marco del proyecto HELICoiD, el objetivo principal ha sido diseñar e implementar en hardware (mediante lenguaje VHDL sobre tecnología FPGA) un acelerador para el algoritmo espacial-espectral K-NN, encargado de emitir el mapa de clasificación final de los tejidos cerebrales.

Tras estudiar el análisis junto con las pruebas experimentales documentadas en esta memoria, se puede afirmar que los objetivos planteados al inicio del proyecto se han cumplido con éxito, alcanzando un alto grado de eficiencia funcional y temporal. Las conclusiones más destacadas que se extraen de este desarrollo son las siguientes:

- **Exactitud algorítmica y fiabilidad:** La implementación en VHDL ha demostrado ser completamente funcional y equivalente a su contraparte en software (C++). Gracias al entorno de verificación (*testbench*) diseñado y un sistema de autocomprobación automatizado, se ha confirmado que el hardware procesa las probabilidades de SVM y la banda PCA sin introducir errores lógicos ni pérdidas de información.
- **Cumplimiento del tiempo real:** Se ha conseguido mejorar el cuello de botella computacional. Operando a una frecuencia de reloj estándar de 100 MHz, la arquitectura ha logrado procesar y clasificar un hipercubo de datos médicos completo en aproximadamente 780,7 ms. Bajar de la barrera del segundo confirma que el sistema es viable para asistir al cirujano de forma fluida y continua en quirófano.
- **Eficiencia estructural y escalabilidad:** El diseño ha logrado aprovechar de manera óptima los recursos físicos de la FPGA, destacando el uso de los bloques DSP para la alta carga aritmética de la distancia euclídea. Además, el análisis temporal ha obtenido un *Slack* positivo (WNS de 1,480 ns), lo que otorga al sistema de un margen de mejora que permitiría elevar la frecuencia operativa hasta los 117 MHz sin modificar el código.
- **Validación física exitosa:** Más allá de la simulación informática, el diseño se ha implementado y validado físicamente sobre una placa Basys 3. Mediante la introducción de datos preprocesados en memorias BRAM y la ejecución *on-chip*, se ha demostrado empíricamente que el sistema opera de manera fiable en un entorno hardware real.

En definitiva, se ha logrado construir una arquitectura hardware robusta, modular y paralela que valida el uso de FPGAs como solución definitiva para llevar la clasificación avanzada de imágenes médicas a un entorno clínico real.

8.2. Trabajo futuro

Aunque los resultados obtenidos cumplen con las metas de este trabajo, la estructura modular del diseño abre la puerta a diversas líneas de investigación y desarrollo para continuar perfeccionando el sistema. Se proponen las siguientes vías de trabajo futuro:

1. **Integración de las etapas de preprocesado:** Actualmente, la arquitectura asume que los datos ya han pasado por las etapas previas de PCA y SVM en un ordenador, recibéndolos a través de ficheros de texto. Un siguiente paso sería desarrollar también en VHDL estos dos módulos. De esta forma, la placa FPGA podría recibir la información de la cámara médica en bruto y realizar el proceso completo de clasificación de forma 100 % autónoma, sin depender de un PC externo.
2. **Integración en vivo con sensores hiperespectrales:** Tras haber validado el núcleo hardware con datos estáticos en la placa, resultaría de gran interés conectar la FPGA a una cámara hiperespectral real. Esto permitiría procesar el flujo de vídeo médico en vivo, validando la latencia del sistema y los anchos de banda bajo condiciones de entrada/salida físicas dentro de un entorno quirúrgico real.
3. **Exploración de los límites de frecuencia:** Aprovechando el *Slack* positivo obtenido (que permite subir teóricamente a 117 MHz), se podría someter al circuito a pruebas de estrés a mayores frecuencias de reloj. Sumado a una posible introducción de nuevas etapas de segmentación (*pipelining*) en la ruta crítica, el tiempo de procesamiento actual de 780 ms podría reducirse aún más, permitiendo trabajar con imágenes de mayor resolución.
4. **Optimización del consumo de memoria:** El análisis de recursos reveló que almacenar la ventana espacial consume casi el 70 % de las memorias internas (BRAM) de la placa Basys 3. Para evitar este cuello de botella y poder utilizar ventanas de búsqueda más grandes, un trabajo futuro podría centrarse en adaptar el código para usar la memoria RAM externa que también incluye la propia placa, liberando así los recursos internos del chip.

Capítulo 9

Conclusions and future work

9.1. Conclusions

This Bachelor’s Thesis emerged with the purpose of addressing a critical technological barrier in the medical field: the high computational cost that prevents the use of hyperspectral imaging in real-time during neurosurgeries. Taking the framework of the HELICoiD project as inspiration, the main objective has been to design and implement in hardware (using VHDL language on FPGA technology) an accelerator for the spatial-spectral K-NN algorithm, responsible for generating the final classification map of brain tissues.

After analyzing the design alongside the experimental tests documented in this manuscript, it can be stated that the objectives set at the beginning of the project have been successfully met, achieving a high degree of functional and temporal efficiency. The most notable conclusions drawn from this development are the following:

- **Algorithmic accuracy and reliability:** The VHDL implementation has proven to be fully functional and equivalent to its software counterpart (C++). Thanks to the designed verification environment (*testbench*) and an automated self-checking system, it has been confirmed that the hardware processes the SVM probabilities and the PCA band without introducing logical errors or information loss.
- **Real-time compliance:** The computational bottleneck has been successfully mitigated. Operating at a standard clock frequency of 100 MHz, the architecture has managed to process and classify a complete hypercube of medical data in approximately 780.7 ms. Dropping below the one-second barrier confirms that the system is viable for assisting the surgeon fluidly and continuously in the operating room.
- **Structural efficiency and scalability:** The design has optimally utilized the physical resources of the FPGA, highlighting the use of DSP blocks for the high arithmetic load of the Euclidean distance. Furthermore, the timing analysis yielded a positive *Slack* (WNS of 1.480 ns), providing the system with a margin of improvement that would allow raising the operating frequency up to 117 MHz without modifying the code.
- **Successful physical validation:** Beyond computer simulation, the design has been physically implemented and validated on a Basys 3 board. By feeding preprocessed data into BRAM memories and executing *on-chip*, it has been empirically demonstrated that the system operates reliably in a real hardware environment.

In short, a robust, modular, and parallel hardware architecture has been built, validating the use of FPGAs as a definitive solution to bring advanced classification of medical images into a real clinical environment.

9.2. Future work

Although the obtained results fully meet the goals of this work, the modular structure of the design opens the door to several lines of research and development to further perfect the system. The following avenues for future work are proposed:

1. **Integration of preprocessing stages:** Currently, the architecture assumes that the data has already gone through the previous stages of PCA and SVM on a computer, receiving them through text files. A logical next step would be to develop these two modules in VHDL as well. In this way, the FPGA board could receive the raw information from the medical camera and perform the entire classification process 100 % autonomously, without relying on an external PC.
2. **Live integration with hyperspectral sensors:** After validating the hardware core with static data on the board, it would be of great interest to connect the FPGA to a real hyperspectral camera. This would allow processing the live medical video stream, validating the system's latency and bandwidths under physical input/output conditions within a real surgical environment.
3. **Exploration of frequency limits:** Taking advantage of the obtained positive *Slack* (which theoretically allows increasing to 117 MHz), the circuit could be subjected to stress tests at higher clock frequencies. Coupled with a possible introduction of new *pipelining* stages in the critical path, the current processing time of 780 ms could be further reduced, allowing work with higher resolution images.
4. **Memory consumption optimization:** The resource analysis revealed that storing the spatial window consumes almost 70 % of the internal memories (BRAM) of the Basys 3 board. To avoid this bottleneck and be able to use larger search windows, future work could focus on adapting the code to use the external RAM memory also included on the board itself, thus freeing up the chip's internal resources.

Bibliografía

- [1] Pixxel Space Academy. *Understanding Hyperspectral Data Cubes*. <https://www.pixxel.space/academy/article/understanding-hyperspectral-data>. 2024.
- [2] Varios Autores. «Hyperspectral imaging for precision nitrogen management: A comparative exploration of two methodological approaches to estimate optimal nitrogen rate». En: *ResearchGate / GFZpublic* (2025). URL: <https://www.researchgate.net/publication/388433613>.
- [3] Varios Autores. «Modern Trends and Recent Applications of Hyperspectral Imaging: A Review». En: *MDPI* (2025). URL: <https://www.mdpi.com/2227-7080/13/5/170>.
- [4] Himar Fabelo et al. «An Intraoperative Visualization System Using Hyperspectral Imaging to Aid in Brain Tumor Delineation». En: *Sensors (Basel)* 18.2 (2018), pág. 430.
- [5] Himar Fabelo et al. «Spatio-spectral classification of hyperspectral images for brain cancer detection during surgical operations». En: *PLOS ONE* 13.3 (2018), e0193721.
- [6] Giampaolo Florimbi, Himar Fabelo, Samuel Ortega et al. «Parallel Implementations Assessment of a Spatial-Spectral Classifier for Hyperspectral Clinical Applications». En: *IEEE Access* 6 (2018), págs. 62850-62863.
- [7] J. Fowers, G. Brown, P. Wadden et al. «Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing». En: *2010 International Conference on Field-Programmable Technology*. IEEE. 2010, págs. 1-8.
- [8] Geoinnova. *Qué son las firmas espectrales y su importancia*. <https://geoinnova.org/blog-territorio/firmas-espectrales/>. 2021.
- [9] Sakari Lahti et al. «Are We There Yet? A Study on the State of High-Level Synthesis». En: *IEEE Access* (2019). URL: https://www.researchgate.net/publication/325025819_Are_We_There_Yet_A_Study_on_the_State_of_High-level_Synthesis.
- [10] Raquel Lazcano, Daniel Madroñal, Giampaolo Florimbi et al. «Accelerating the K-Nearest Neighbors Filtering Algorithm for Real-Time Hyperspectral Processing». En: *Sensors* (2019). Extensión de estudios previos sobre filtrado K-NN en entornos quirúrgicos.
- [11] Raquel Lazcano et al. «Accelerating the K-Nearest Neighbors Filtering Algorithm to Optimize the Real-Time Classification of Human Brain Tumor in Hyperspectral Images». En: *Sensors* 18.7 (2018), pág. 2314.
- [12] Guolan Lu y Baowei Fei. «Medical hyperspectral imaging: a review». En: *Journal of biomedical optics* 19.1 (2014), pág. 010901.
- [13] Gustavo Marrero Callicó, Samuel Ortega Sarmiento y Eduardo Juárez Martínez. «Proyecto HELICoiD: detección de tumores cerebrales mediante imágenes hiperespectrales». En: *Bit* 208 (2018), págs. 64-67.
- [14] Inc. MicroImages. *Introduction to Hyperspectral Imaging*. Tutorial sobre espectrometría y reflectancia espectral. 2024.
- [15] Photon etc. *Hyperspectral Imaging in Counterfeit Ink and Banknote Identification*. <https://www.photonetc.com/white-papers/hyperspectral-imaging-in-counterfeit-ink-and-banknote-identification>. 2026.
- [16] Hongbin Pu y Da-Wen Sun. «Evaluation of Food Quality and Safety with Hyperspectral Imaging (HSI)». En: *ResearchGate* (2015). URL: <https://www.researchgate.net/publication/284534948>.
- [17] Specim. *Food quality and composition analysis with hyperspectral imaging*. <https://www.specim.com/food-quality-and-composition-analysis-with-hyperspectral-imaging/>. Accedido: 2026. 2025.
- [18] Specim. *Hyperspectral imaging for environmental monitoring*. <https://www.specim.com/hyperspectral-imaging-applications/environmental-monitoring/>. Accedido: 2026. 2025.
- [19] Specim. *Mining and Mineral Exploration with Hyperspectral Imaging Solutions*. <https://www.specim.com/hyperspectral-imaging-applications/mining-and-mineral-exploration-with-hyperspectral-imaging-solutions/>. 2025.
- [20] Antonio Álvarez Sánchez, Eneko Retolaza Ardanaz y Fabrizio Nicolás Zeballos. «Aceleración mediante el Intel oneAPI Toolkit de algoritmos para la detección de cáncer de piel y tumores cerebrales a través de imágenes hiperespectrales». Trabajo de Fin de Grado. Universidad Complutense de Madrid, 2024.
- [21] U.S. Geological Survey (USGS). *Hyperspectral Imaging of Mineral Resources from New and Old Origins*. <https://www.usgs.gov/centers/gggsc/science/hyperspectral-imaging-mineral-resources-new-and-old-origins-minerals-nations>. 2024.
- [22] Jiashu Xu. «Comparison and Application of FPGA and ASIC in Digital System Design». En: *Proceedings of the International Conference on Decision Science & Management*. Association for Computing Machinery, 2024, págs. 216-220. URL: <https://doi.org/10.1145/3686081.3686117>.