Proyecto Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Implementación de técnicas de Machine-Learning en FPGAs para la identificación de partículas

Autor: José Ignacio López Flores
Tutores: Fernando Muñoz Chavero
José María Hinojo Montero

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

DINEL
DPTO. INGENIERÍA ELECTRÓNICA

Proyecto Fin de Carrera
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Implementación de técnicas de Machine-Learning en FPGAs para la identificación de partículas

Autor:
José Ignacio López Flores

Tutores:
Fernando Muñoz Chavero
Catedrático de Universidad

José María Hinojo Montero
Profesor Titular

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Proyecto Fin de Carrera: Implementación de técnicas de Machine-Learning en FPGAs para la identificación de partículas

Autor: José Ignacio López Flores

Tutores: Fernando Muñoz Chavero

José María Hinojo Montero

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Agradecer especialmente a mi familia y amigos que me han apoyado a lo largo de todos estos años en mi etapa de estudiante del grado.

Quisiera, además, señalar la profesionalidad de los profesores que he tenido durante todo mi periodo académico, demostrando sus ganas y pasión por la docencia.

Resumen

La identificación de partículas es una parte clave en el estudio de la física de altas energías, fundamental para la colección de datos que se usarán para la deducción de nuevas teorías físicas o verificación de hipótesis.

La técnica utilizada en este proyecto para la identificación de partículas es el análisis de la forma de pulso. Esta técnica es capaz de identificar partículas con gran precisión, gracias al empleo de técnicas de 'Machine-Learning'.

A lo largo de este proyecto, se realizará un estudio de las técnicas de 'Machine-Learning', a través del software MATLAB, con el fin de poder implementarlas dentro de una FPGA Artix-7, optimizando así los recursos necesarios y la carga o coste computacional que éstas suponen. Dicho estudio se basará en la observación del comportamiento de las técnicas de 'Machine Learning' al trabajar en punto fijo, comprobando cuál debe ser la resolución para cada parámetro y cómo aplicar ciertos algoritmos.

Una vez terminado el estudio se estimarán los recursos necesarios para la implementación final, y se realizará una comparativa con un ejemplo de aplicación sobre la FPGA.

Abstract

The identification of particles is a key part in the study of high energy physics, essential for the collection of data that will be used for the deduction of new physical theories or verification of hypotheses.

The technique used in this project for particle identification is pulse shape analysis. This technique can identify particles with great precision, thanks to the use of 'Machine-Learning' techniques.

Throughout this project, a study of the 'Machine-Learning' techniques will be carried out, through the MATLAB software, to be able to implement them within an Artix-7 FPGA, thus optimizing the necessary resources and the load or cost. computational that they imply. This study will be based on the observation of the behaviour of the 'Machine Learning' techniques when working at a fixed point, checking what the resolution should be for each parameter and how to apply certain algorithms.

Once the study is finished, the necessary resources for the final implementation will be estimated, and a comparison will be made with an application example on the FPGA.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

PSA	Análisis de la forma de pulso
PCA	Análisis de los componentes principales
SVM	Support Vector Machine
RBF	Radiactive beam facilities
Kr	Kriptón
Ar	Argón
C	Carbono
FPGA	Field Programmable Gate Array
Hz	Herzios
eV	Electrón - Voltio
CAD	Convertidor analógico Digital
LSB	Least Significant Bit
LUT	Look-up Table
RAM	Random Access Memory
PLL	Phase – Locked Loop
MSPS	Muestras por sesgundos
FF	Flip - Flop
CASS	Celda de control de adicción, sustracción y desplazamiento
ANN	Redes neuronales artificiales
I/O	Entradas y salidas

1 INTRODUCCIÓN

La ciencia es un esfuerzo continuo por descubrir la belleza oculta en la realidad y comprender el misterio que nos rodea.

-Marie Curie -

A lo largo de este capítulo se darán las razones por las que se ha realizado este proyecto, así como el objetivo de este.

1.1 Contexto del proyecto

La física de partículas es una rama de la física que se ocupa del estudio de las partículas elementales y las interacciones que rigen el comportamiento de la materia en el nivel más fundamental. Dicha rama es conocida también como física de altas energías, ya que muchas partículas sólo pueden ser observadas cuando éstas colisionan en aceleradores de partículas.

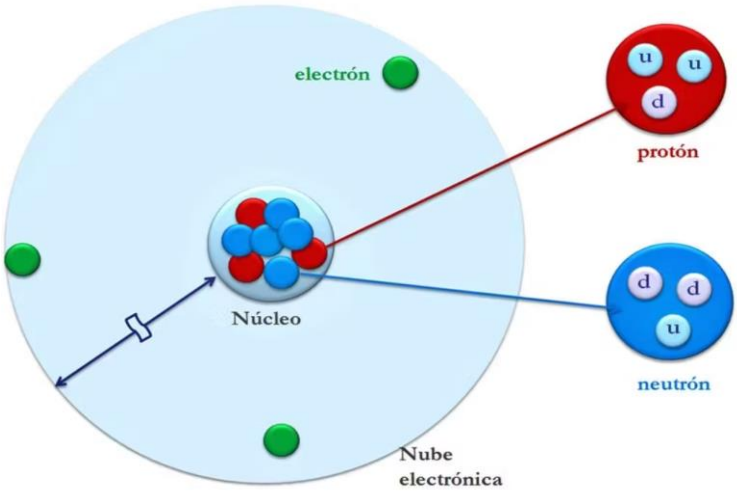

Figura 1-1. Estructura atómica (imagen extraída en [17]).

En esta disciplina, se investiga una amplia variedad de partículas, que van desde los constituyentes fundamentales de la materia, llamados fermiones, los cuales están formados por quarks y leptones; hasta las partículas que interactúan con la propia materia, los bosones. También se estudian partículas más masivas, como los protones y los neutrones, que están compuestos por quarks y se encuentran confinados dentro de los núcleos atómicos.

La identificación de partículas es un aspecto crucial en la física de partículas. Consiste en determinar qué tipo de partícula se ha producido en un experimento o colisión de alta energía. Esto se logra mediante el análisis de las mediciones arrojadas por los detectores de partículas o sensores de colisiones de partículas. De forma que, a partir de dicho análisis se puede deducir propriedades y características de las partículas, como su carga eléctrica, masa, momento y tiempo de vida.

La identificación precisa de partículas es esencial para comprender los fenómenos fundamentales de la física de partículas. Permite confirmar la existencia de nuevas partículas y validar teorías existentes, como el Modelo Estándar de la física de partículas, así como descubrir partículas exóticas que podrían desafiar las teorías establecidas. Por tanto, la instrumentación y procesamiento posterior a la medición son elementos claves para el desarrollo de esta disciplina.

En las últimas dos décadas, el desarrollo de algoritmos basados en 'Machine-Learning' (Aprendizaje Automático) ha experimentado un crecimiento exponencial, beneficiando notablemente al ámbito científico gracias a la capacidad que ofrecen estas técnicas de modelos predictivos para estimar con gran precisión resultados futuros.

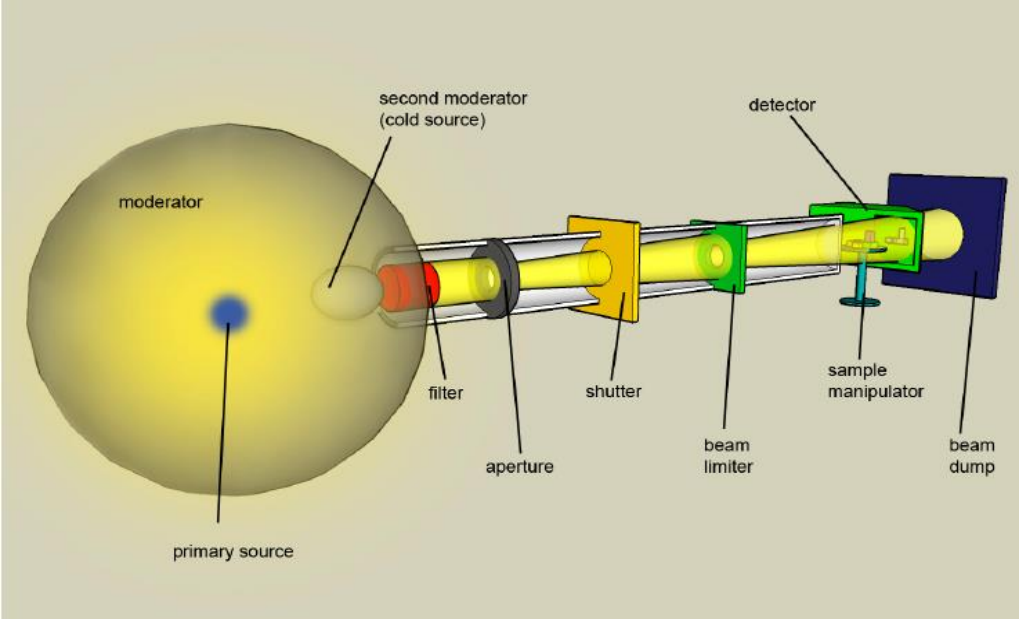

Figura 1-2. Ensayo de detección de partículas (Imagen extraída de [15]).

La identificación de partículas mediante técnicas de análisis de la forma de pulso (del inglés 'pulse shape analysis' PSA) es un método prometedor que se apoya de este gran avance en el desarrollo de técnicas de 'Machine-Learning' como es el análisis de componentes principales (PCA) y los algoritmos de tipo 'support-vector machine' (SVM), entre otros.

Dichas técnicas requieren una alta capacidad de procesamiento de datos. De forma que, el empleo de dispositivos como las FPGA sería adecuado para ello, ya que ofrecen grandes ventajas como el funcionamiento de distintos procesos en paralelo, un bajo coste y una alta configurabilidad

La identificación de partículas utilizando PSA se realiza en una instalación de haces radiactivos (del inglés 'radiactive beam facilities' - RBF) como, por ejemplo, GINAL; que dispone del acelerador de partículas SPIRAL2. En dichas instalaciones, se acelera un haz de iones de un determinado tipo de partículas que impactarán sobre un sensor de estado sólido (normalmente de Silicio) y éste, a su vez, medirá la carga producida por el impacto. A partir de la lectura del sensor, se emplean técnicas de 'Machine-Learning' como redes

neuronales o clasificadores lineales como es 'Support-Vector Machine' (SVM) para poder clasificar las partículas colisionadas. [1,2,3,4].

1.2 Objetivo

El objetivo de este proyecto es la implementación eficiente de un sistema de identificación PSA en una FPGA Artix-7, capaz de discernir entre los siguientes pares de isótopos: ^{80}Kr y ^{84}Kr; ^{36}Ar y ^{40}Ar; ^{12}C y ^{13}C.

La distinción de los elementos químicos se realizará mediante un sencillo algoritmo de medición del tiempo; mientras que, la identificación de los isótopos se ejecutará a través de elaboradas técnicas de 'Machine-Learning', las cuales requerirán un gran consumo de recursos.

En este proyecto, se buscará que los recursos empleados sean mínimos sin tener un gran deterioro en la identificación y, de este modo, puedan ofrecer resultados fiables.

En el enfoque actualmente utilizado, se emplean redes neuronales que requieren una carga computacional considerable para realizar la clasificación de isótopos. Sin embargo, en este proyecto se explorarán técnicas alternativas con el objetivo de reducir dicha carga computacional.

En primer lugar, se aplicará una técnica de extracción de características conocida como Análisis de Componentes Principales (PCA, por sus siglas en inglés). Esta técnica permite reducir la dimensionalidad de los datos al proyectarlos en un espacio de menor tamaño, manteniendo la información más relevante. La aplicación de PCA puede ayudar a simplificar el procesamiento y análisis posterior.

Posteriormente, se utilizarán técnicas eficientes de clasificación, como Máquinas de Vectores de Soporte (SVM, por sus siglas en inglés). Los SVM son algoritmos de aprendizaje automático que se basan en la separación óptima de los datos en un espacio multidimensional. Estas técnicas de clasificación son conocidas por su eficacia y velocidad en la clasificación de conjuntos de datos complejos.

El propósito principal de este enfoque es lograr una clasificación precisa de los isótopos utilizando técnicas más eficientes en términos de carga computacional, en comparación con las redes neuronales utilizadas anteriormente. Esto permitirá obtener resultados confiables mientras se minimiza el consumo de recursos en la FPGA Artix-7.

2 FUNDAMENTOS MATEMÁTICOS

Las matemáticas involucradas en este proyecto son técnicas o algoritmos de 'Machine-Learning' que se explicarán a continuación. Estas realizan un preciso análisis de datos, a través del estudio estadísticos de estos.

2.1 PCA

PCA (Análisis de Componentes Principales) es una técnica de análisis estadístico que se utiliza para reducir la dimensionalidad de un conjunto de datos. El objetivo del PCA es encontrar una representación más compacta y significativa de los datos, identificando las direcciones principales o componentes que explican la mayor parte de la variabilidad. Al proyectar los datos en estas nuevas direcciones, se obtiene una versión simplificada del conjunto de datos original, lo que facilita su interpretación y visualización. PCA es ampliamente utilizado en diversas áreas, como el reconocimiento de patrones, el procesamiento de imágenes y el análisis de datos multidimensionales.

El cálculo del PCA implica varios pasos. En primer lugar, se normalizan los datos de entrada,

El objetivo de este paso es estandarizar el rango de las variables continuas iniciales para que cada una de ellas contribuya de manera equitativa al análisis.

Más específicamente, es crítico realizar la estandarización previa al PCA debido a que esta técnica es altamente sensible a las varianzas de las variables iniciales. Si existen diferencias significativas entre los rangos de las variables iniciales, aquellas variables con rangos más amplios dominarán sobre aquellas con rangos más pequeños (por ejemplo, una variable que varíe entre 0 y 100 dominará sobre una variable que varíe entre 0 y 1), lo cual conducirá a resultados sesgados. Por lo tanto, transformar los datos a escalas comparables puede evitar este problema.

Matemáticamente, esto se puede lograr restando la media y dividiendo por la desviación estándar para cada valor de cada variable.

Una vez que se realiza la estandarización, todas las variables se transformarán a la misma escala.

En el paso 2, se calcula la matriz de covarianza con el objetivo de comprender cómo las variables del conjunto de datos de entrada varían con respecto a la media entre ellas y detectar posibles relaciones entre ellas. La matriz de covarianza es una matriz simétrica $p \times p$ (donde p es el número de dimensiones) que contiene las covarianzas asociadas a todas las posibles combinaciones de variables iniciales. Esta matriz nos proporciona información sobre las correlaciones entre las variables, siendo el signo de la covarianza el aspecto relevante: si es positiva, las variables aumentan o disminuyen juntas (correlación); si es negativa, una variable aumenta mientras la otra disminuye (correlación inversa).

En el paso 3, se calculan los autovectores y autovalores de la matriz de covarianza para identificar los componentes principales. Los componentes principales son nuevas variables construidas como combinaciones lineales de las variables iniciales. Estas combinaciones se realizan de tal manera que los nuevos componentes sean no correlacionados y contengan la mayor cantidad de información de las variables iniciales. Los componentes principales representan las direcciones en las que los datos explican la máxima varianza, es decir, las líneas que capturan la mayor parte de la información de los datos. Al organizar la información en componentes principales, es posible reducir la dimensionalidad sin perder mucha información al descartar componentes con poca información y considerar los componentes restantes como las nuevas variables.

Geométricamente, los componentes principales representan las direcciones óptimas para visualizar y evaluar los datos, ya que capturan la mayor cantidad de información. Los componentes principales proporcionan un nuevo sistema de ejes que permiten una mejor visualización de las diferencias entre las observaciones.

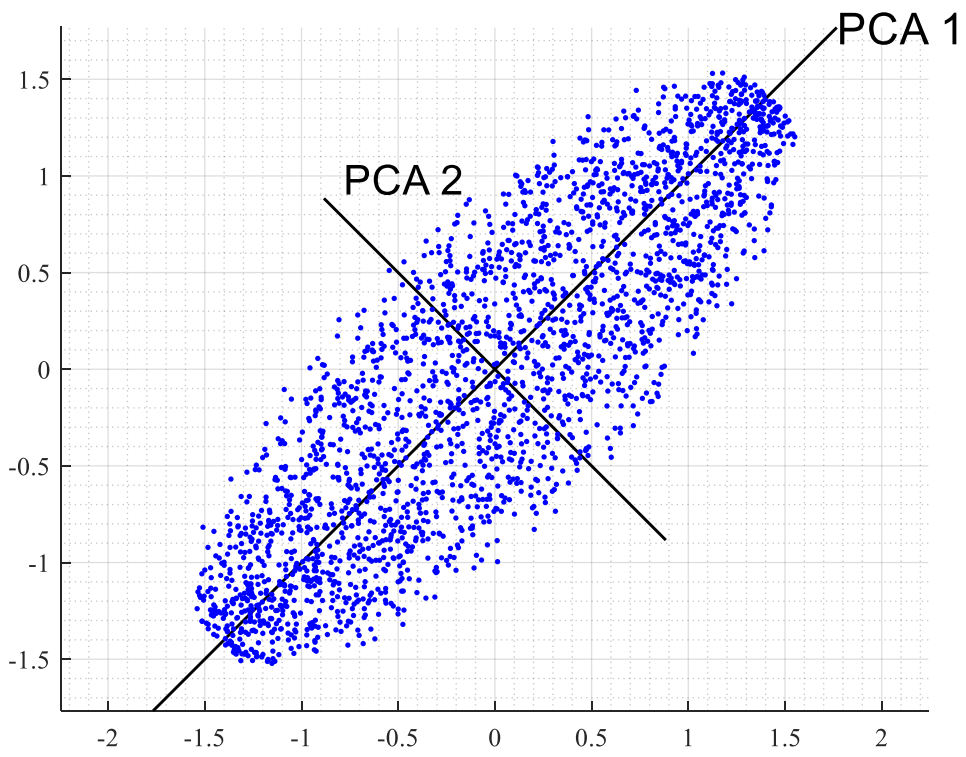

Figura 2-1. Ejemplo de Ejes Principales

De cara a aplicar esta técnica a nuevos datos de entrada, la información que albergue estos se mantendrá, de forma comprimida, dentro de unas nuevas variables no correlacionadas, dichas variables se denominan predictores. El número de predictores dependerá del número de autovectores escogidos.

De este modo, se define una matriz de transformación de dimensiones *p x n*, donde debe cumplirse que $p < n$. Por tanto, frente a un vector de datos de dimensión *n*, el resultado de multiplicar dicho vector por la matriz de transformación dará un vector de salida de dimensión *p* que contiene la información más importante. [5]

$$m_{px1} = T_{pxn}^{PCA} \cdot m_{nx1} \tag{2-1}$$

2.2 SVM (binario con Kernel polinómico de orden 3)

SVM (Support Vector Machines) es un algoritmo de aprendizaje supervisado utilizado para la clasificación y regresión de datos. El objetivo principal del SVM es encontrar un hiperplano en un espacio de alta dimensión que separe de manera óptima las clases de datos. En la clasificación, el hiperplano divide el espacio en diferentes regiones, asignando cada punto de datos a una clase específica. SVM busca el hiperplano que maximiza la distancia entre los puntos de datos de diferentes clases, lo que proporciona una buena generalización y capacidad de clasificación en datos no vistos como se observa en la Figura 2-2.

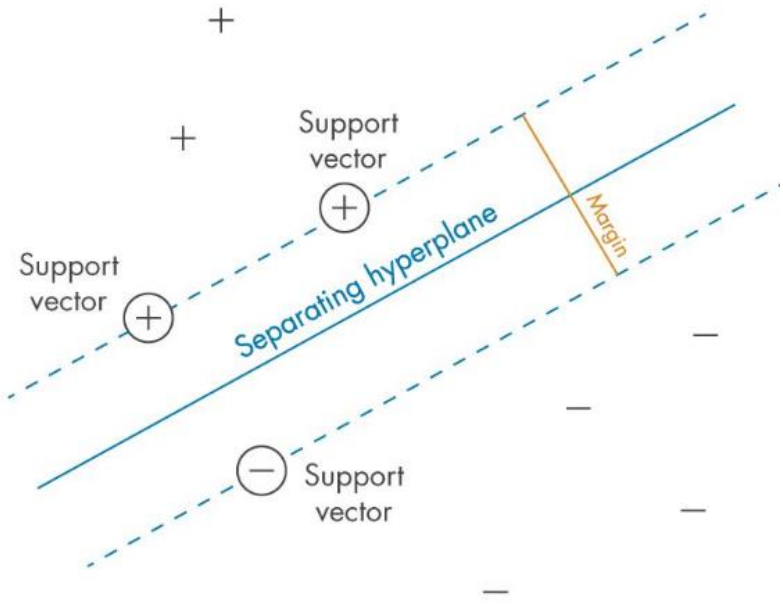

Figura 2-2. Ejemplo de generación de hiperplano (imagen extraída de [7])

Este hiperplano queda definido por datos de entrenamiento que delimitan la frontera entre las dos clases, llamados 'vectores soporte' o 'vector de características'. De esta forma, en función de la posición de un nuevo dato de entrada respecto a los vectores soporte se puede conocer a qué clase pertenece. Para ello, el algoritmo toma la decisión de clasificación mediante el resultado de una combinación lineal entre los vectores soporte y el dato de entrada, llamado 'score'. [6,7].

Además, SVM puede manejar eficientemente datos no lineales mediante el uso de funciones de kernel, que transforman los datos a un espacio de mayor dimensión donde pueden ser separados linealmente, esto se ve con claridad en el ejemplo mostrado en la Figura 2-3, en la cual la distribución inicial de los datos forzaba a tener un clasificador circular, altamente no-lineal, y una vez aplicado la función de kernel, la clasificación puede realizarse linealmente.

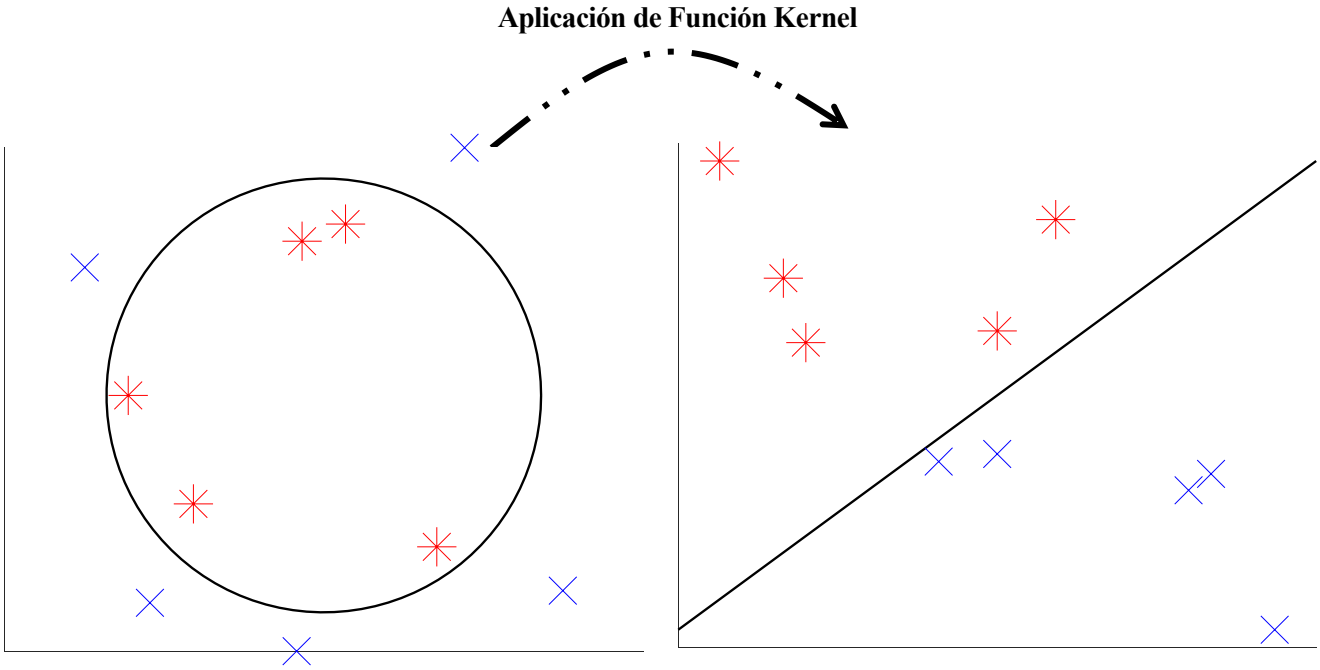

Figura 2-3. Transformación de variables mediante kernel

SVM se ha utilizado ampliamente en diversas aplicaciones, como reconocimiento de imágenes, detección de spam y análisis de textos, debido a su capacidad para manejar problemas de clasificación complejos. En este proyecto el SVM debe resolver un problema de clasificación identificando a qué isótopo se corresponde los datos de entrada suministrados.

El criterio de clasificación para cada par de isótopo se basará en lo siguiente:

1. Los datos de entrada serán procesados a través del PCA, y se obtendrán unos predictores asociados.
2. Los predictores se introducirán a un SVM, el cual, y para este proyecto, calculará un único 'score' y lo asociará a una clase u otra en función del signo de éste.

No se tiene en cuenta el caso donde el 'score' sea igual a cero debido a que es una situación altamente improbable.

$$score = \sum_{i=1}^{N} \alpha_i y_i G(Q_{SV}, Q_{predictor}) + \beta_0 \tag{2-2}$$

$$predicción = \begin{cases} isótopo\ 1, & score < 0 \\ isótopo\ 2, & score \geq 0 \end{cases}$$

Ambas técnicas, al ser algoritmos de 'Machine-Learning' dependerán de los datos del set de entrenamiento y no se modificarán una vez implementados en la FPGA. El proceso de entrenamiento y validación se realizarán a través del software MATLAB.

3 ESTUDIO PRE-IMPLEMENTACIÓN

El desarrollo de este proyecto se ha realizado, principalmente, mediante el software MATLAB, con el cual se ha podido abordar los distintos problemas desde distintos niveles de abstracción, desde funciones de alto nivel hasta el cálculo a nivel de bit. A continuación, se realizará un estudio previo a la implementación en la FPGA, de forma que se pueda realizar una planificación de los recursos necesarios

3.1 Consideraciones iniciales

La adquisición de los datos se produce a una tasa de muestreo más alta (frecuencia de muestreo entre 1 y 2 GHz) que la frecuencia de trabajo típica de una FPGA, que suele encontrarse entre los 100 y 450 MHz. Es por esta razón que el procesamiento y análisis de los datos se realizará una vez ya obtenidos.

Hay que mencionar que el análisis de la implementación se basará única y exclusivamente en la identificación del tipo de isótopo, ya que la identificación del elemento químico se realizará mediante la medición del tiempo de decaimiento o tiempo que tarda la señal en ir del 90 % del valor máximo hasta el 10 %.

El tiempo de decaimiento de cada uno de estos se diferencia considerablemente entre sí, debido fundamentalmente a la energía con la que impacta el haz de partículas sobre el sensor de estado sólido. A mayor energía de impacto, mayor es el tiempo de decaimiento. En consecuencia, sabiendo que la energía por nucleón es aproximadamente de 8 MeV, es fácil identificar partículas con números atómicos dispersos entre sí (^{12}C a 98.54 MeV, ^{13}C a 96.75 MeV, ^{36}Ar a 313.92 MeV, ^{40}Ar a 312.88 MeV, ^{80}Kr a 688.43 MeV y ^{84}Kr a 676.18 MeV).

Por tanto, midiendo el número de muestras entre un valor umbral hasta que la lectura vuelva a valer 0, se puede identificar el elemento químico asociado a la medida.

En la Figura 3-1, se puede comprobar que cada par de isótopo tiene un tiempo decaimiento característico.

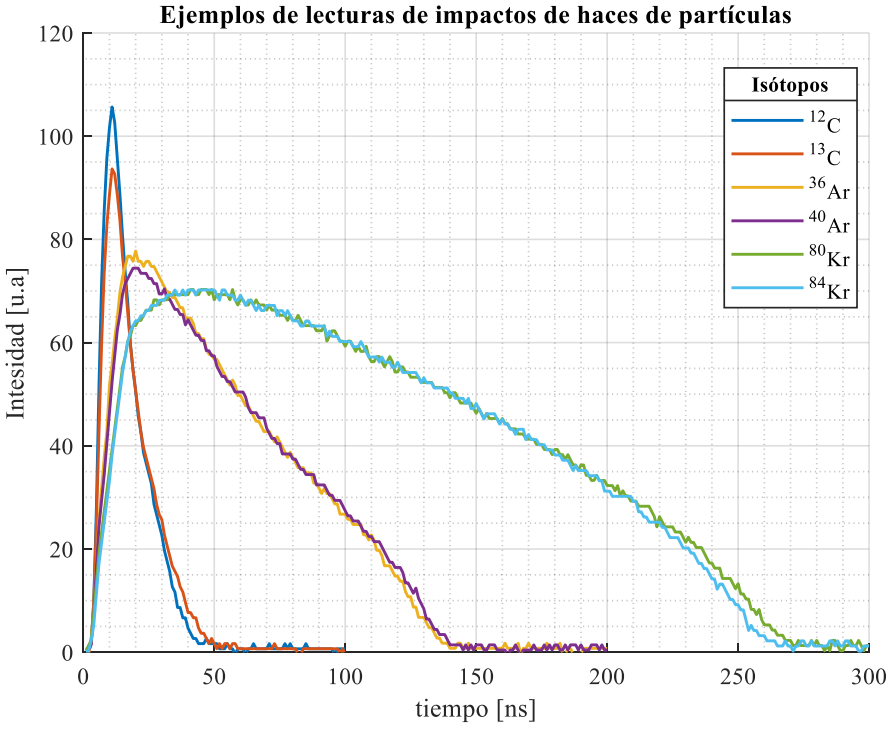

Figura 3-1. Representación de medidas de distintas partículas

Como se ha mencionado anteriormente, gracias a la medición del tiempo de decaimiento se puede identificar a qué partícula se corresponde cierta señal medida con relativa facilidad. Sin embargo, no ocurre lo mismo con la identificación del tipo de isótopo, ya que las variaciones de energía entre pares de isótopos rondan el 1 o 2 %. Y dichas variaciones no son suficientemente representativas como para hacer una distinción entre ellos.

Es por esta razón, que se utilizarán técnicas de 'Machine-Learning' para la identificación en específico del tipo de isótopo, capaces de realizar un análisis más fino de las características de la medida.

El entrenamiento y validación son procesos fundamentales para la elaboración y creación de las técnicas de 'Machine-Learning'. Además, de ser procesos que deben estar optimizados para garantizar resultados fiables. De este modo, estos procesos se realizarán a través de las toolbox de MATLAB 'Clasification Learner', 'Deep Network Designer' y 'Regression Learner' los cuales utilizan el método de 'Cross-Validation' para la evaluación de los modelos.

3.2 Enfoque de la investigación

El análisis del comportamiento de la implementación de técnicas de 'Machine-Learning' en una FPGA es, en esencia, un análisis de distintas operaciones en punto fijo de algoritmos que requieren un alto coste computacional. De este modo, a lo largo del proyecto se comprobará cómo se deben implementar estos algoritmos, qué resolución deben tener las variables implicadas y cuántos recursos consume de la FPGA para, de este modo, conseguir una alta precisión en la identificación de las partículas minimizando el consumo de recursos.

3.3 Recopilación de los datos

La recopilación de los datos previo a ser analizados, proceden de medidas experimentales donde se empleó un Convertidor Analógico-Digital (CAD) con una resolución de 8 bits. Los datos están asociados con 3 pares de isótopos: ^{80}Kr y ^{84}Kr; ^{36}Ar y ^{40}Ar; ^{12}C y ^{13}C. Para cada par de isótopos, se han realizado 4000 experimentos (2200 para el caso del Kriptón), de forma que la mitad de los experimentos se corresponden con uno de los pares de isótopos. En cada experimento, en función del elemento químico, el número de muestras adquiridas por el CAD varía, de modo que para los experimentos relativos al Kriptón se toman 300 muestras, para el Argón 200 y para el Carbono, 100. Esto último se debe al tiempo que tarda la señal en disiparse (tiempo de decaimiento), ya que, dada una tasa de muestreo igual para todos los experimentos, cuanto menos peso tenga la partícula, menos energía habrá en la colisión y, consecuentemente, menos tiempo llevará la lectura de la medida.

El PCA y el SVM ya vienen determinadas por los datos de entrenamiento procedentes del conjunto de datos explicados anteriormente. De forma que cada par de isótopos tendría su propio PCA y SVM.

Previamente, a la aplicación de las técnicas de 'Machine-Learning', se realiza un análisis previo para conocer los parámetros necesarios.

El número de componentes principales y, por tanto, de predictores elegidos será 6, ya que la varianza acumulada en los vectores que forman la matriz de transformación PCA deja de aumentar significativamente al considerar más predictores, como se observa en las Figura 3-2 para el Carbono, Figura 3-3 para el Argón y Figura 3-4 para el Kriptón.

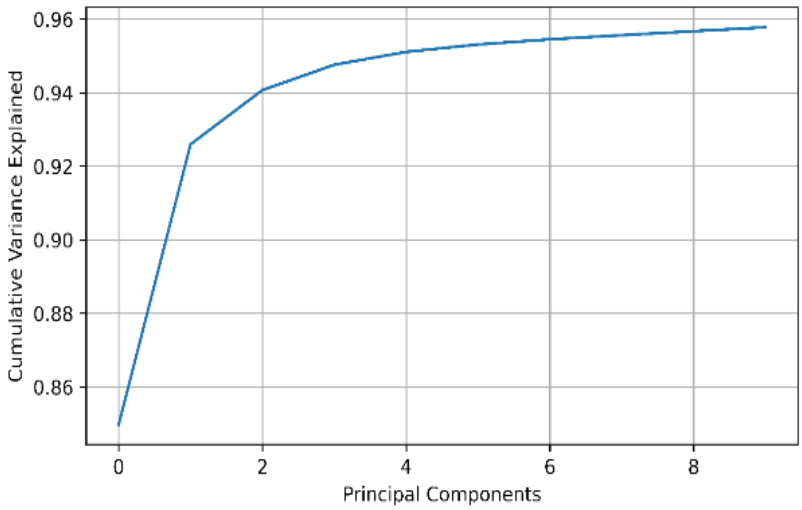

Figura 3-2.. Varianza acumulada de las componentes principales para el Carbono

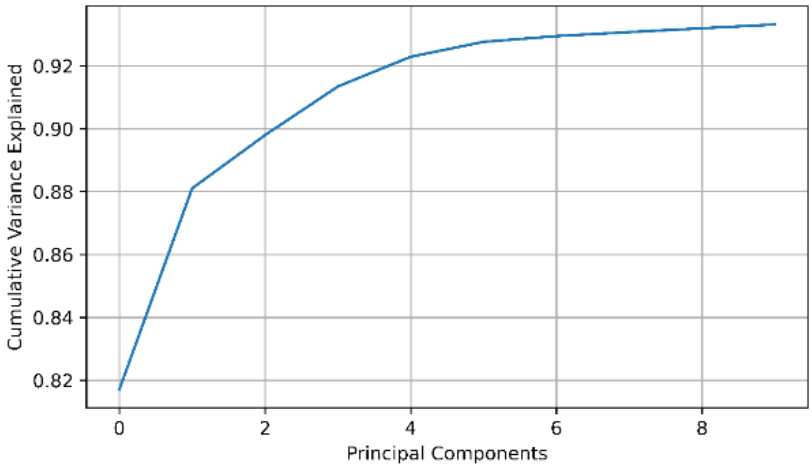

Figura 3-3.. Varianza acumulada de las componentes principales para el Argón

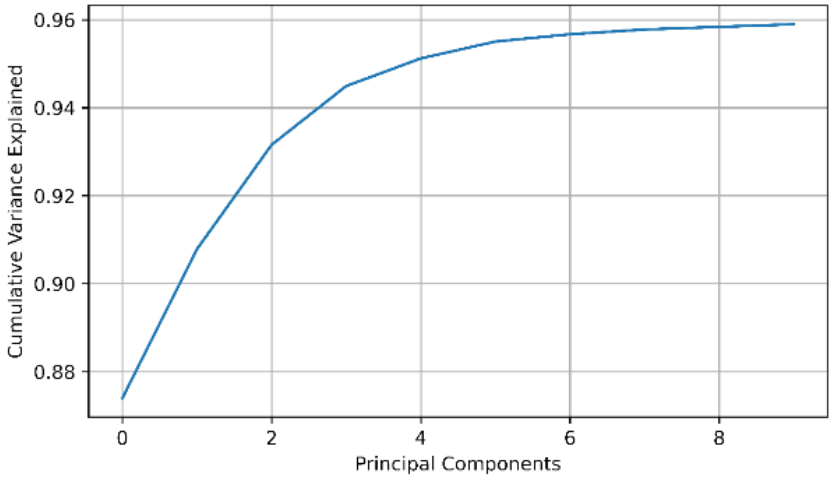

Figura 3-4.. Varianza acumulada de las componentes principales para el Kriptón

Por tanto, las matrices de transformación PCA de cada par de isótopos tendrán las siguientes dimensiones:

- Carbono: 6 por 100
- Argón: 6 por 200
- Kriptón: 6 por 300

La bondad del funcionamiento del modelo SVM se evaluará mediante el factor de mérito M. Este factor mide la eficiencia de un modelo SVM, comprobando el solapamiento entre los grupos identificados o clases entre sí.

En general, cuanto mayor sea M, mejor será la capacidad de clasificación asociado al modelo SVM. Se considera una buena capacidad de discriminación si M es superior a 0.75, y si M supera 1, la discriminación de los datos de entrada es casi total [8,18]. Dicho factor se calcula mediante la fórmula 3-1.

$$M = \frac{\|\overrightarrow{\mu_1} - \overrightarrow{\mu_2}\|}{(\sigma_1 + \sigma_2) \cdot 2.35} \tag{3-1}$$

Donde $\boldsymbol{\mu_{1,2}}$ es la media y $\boldsymbol{\sigma_{1,2}}$ la desviación típica de datos identificados dentro de una clase. Se considera dos grupos, ya que sólo existen dos tipos de isótopos para identificar para un mismo elemento químico.

A continuación, se comprobará qué técnica de clasificación aporta una mejor discriminación de los datos, probando entre un modelo SVM lineal y uno cúbico. De forma que, en la Tabla 3-1 se mostrarán los rendimientos obtenidos en cada uno de ellos.

Tabla 3-1. Factores de mérito

Isótopos	M de SVM lineal	M de SVM cúbico
Carbono	3.59	6.51
Argón	0.65	1.04
Kriptón	1.72	2.03

Se puede observar que el modelo SVM cúbico tiene asociado un mayor factor de mérito respecto al lineal. Además, de tener valores por encima de 1. Por tanto, se empleará el SVM cúbico como método de identificación de isótopos.

Este modelo SVM tiene asociados los siguientes parámetros por cada par de isótopos:

- Carbono 22 coeficientes y 22 vectores soporte de dimensión 1 por 6 con una predicción del 99.91 % con los datos de validación.
- Argón 357 coeficientes y 357 vectores soporte de dimensión 1 por 6 con una predicción del 90.98 % con los datos de validación.
- Kriptón 69 coeficientes y 69 vectores soporte de dimensión 1 por 6 con una predicción del 98.98 % con los datos de validación.

3.4 Análisis de los datos previos

Se tomará como ejemplo el Kriptón por ser un elemento de gran interés, debido, fundamentalmente, por dos razones.

Gracias a su elevado número de parámetros asociados y, por tanto, su alta demanda de carga computacional y recursos, se puede realizar un estudio elaborado en la optimización de las técnicas de 'Machine Learning'.

Además, este elemento tiene asociado una precisión en la identificación relativamente alta, de forma que ligeros deteriores debido a la cuantización o empleo de la aritmética en punto fijo no supondrá en una precisión final demasiado baja.

3.4.1 Datos de estrada

Los datos de entrada serán un vector de 300 características de 8 bits, los cuales siempre toman valores superiores a 0. El LSB del sensor representa 1 a.u ('adimensional unit') de corriente, por tanto, no es necesario realizar ningún escalado. Esto se debe a que los datos proceden de lectura directa del CAD.

Para poder procesarlo correctamente en el PCA, es necesario restarle la media global de cada muestra. Esta se corresponde con la media aritmética de cada muestra de todos los experimentos realizados (2200 experimentos).

Se considerará que este parámetro tiene una resolución de 8 bits. De este modo, el resultado de la normalización será también de 8 bits.

3.4.2 Coeficientes del PCA

La matriz de transformación PCA es una matriz de 6 por 300 (1800 componentes). El rango de los coeficientes es de -0.1748 a 0.3583, por tanto, a la hora de discretizarlos sólo tendrán representación en la parte fraccionaria.

Al observar el cociente entre el coeficiente de mayor magnitud y el de menor magnitud, se nota que existe una diferencia de cinco órdenes de magnitud. Por lo tanto, al cuantificar los coeficientes, se podría utilizar una resolución de bits que permita desechar ciertos coeficientes sin afectar significativamente el resultado de la transformación. Esto significa que se pueden asignar menos bits a los coeficientes de menor importancia, lo cual podría reducir la complejidad y el consumo de recursos en la implementación de la PCA sin deteriorar significativamente los resultados obtenidos.

3.4.3 Parámetros del SVM

El SVM emplea 69 coeficientes que tienen asociado cada uno un vector soporte de 6 dimensiones y un valor de sesgo. Los coeficientes toman valores entre -1 y 1. Algunos de ellos toman valores muy cercanos a 0, de modo que su contribución en el cálculo de la variable 'score' podría llegar a no ser suficientemente significativa. De esta forma, al cuantificarlos se puede considerar cierta resolución de bits que sólo incluya a aquellos coeficientes que presenten una contribución real en el cálculo. En consecuencia, al no considerar algunos coeficientes que tienen una contribución insignificante se reduce el número de iteraciones necesarias y el espacio en memoria ocupado.

3.5 Análisis de los algoritmos empleados

Al igual que en el apartado anterior, se utilizará como ejemplo el Kriptón por las razones mencionadas anteriormente. A continuación, se analizará los comportamiento y funcionamientos de los algoritmos asociados a las técnicas de 'Machine-Learning' que se van a aplicar en este proyecto.

3.5.1 PCA

La aplicación del PCA es el resultado de realizar el producto escalar entre los datos de entradas por la matriz de transformación. De forma que, será necesario implementar un sumatorio por cada predictor.

Siguiendo con el ejemplo del Kriptón, el cálculo se realizará de la siguiente forma:

$$Q_{1x6} = X_{1x300} \cdot M_{300x6} \tag{3-2}$$

Donde $\boldsymbol{Q}$ son predictores, $\boldsymbol{X}$ los datos de entrada y $\boldsymbol{M}$ la matriz de transformación.

$$X_{1x300} \cdot M_{300x6} = [\sum_{i=1}^{300} x_i \cdot m_{i,1} \; ; \sum_{i=1}^{300} x_i \cdot m_{i,2} \; ; \ldots ; \sum_{i=1}^{300} x_i \cdot m_{i,6}] \tag{3-3}$$

Se puede comprobar que el número de multiplicaciones que se deben aplicar es equivalente al número de coeficientes/características por el número de predictores, mientras que el número de sumas es el mismo número, pero restándole 1 por cada sumatorio. Por tanto, para la aplicación del PCA serían necesarias 1794 sumas y 1800 multiplicaciones.

3.5.2 SVM

La implementación del 'Support-Vector Machine' supone la integración de distintas funciones.

En primer lugar, los predictores deben estar normalizados, es decir, es necesario restarle la media global de éstos y dividirlo por su desviación típica. Una vez hecho esto, se realiza una combinación lineal entre los predictores (normalizados) y los vectores soporte. Para este caso, se trata de aplicar un kernel polinómico de orden 3. Seguidamente, al resultado de éste se le aplicará su correspondiente coeficiente y se introducirá dentro de un sumatorio y, una vez terminado todas las iteraciones del sumatorio, se le sumará el sesgo.

El cálculo del 'score' se realiza de la siguiente forma:

$$score = \sum_{i=1}^{N} \alpha_i y_i G(Q_{SV}, Q_{predictores}) + \beta_0 \tag{3-4}$$

Donde $\boldsymbol{\alpha}$ son los coeficientes del SVM, $\boldsymbol{y}$ las etiquetas de los coeficientes ($y_i = \pm 1$), $\boldsymbol{G(Q_{SV_i}, Q_{predictor})}$ la función kernel empleada (cúbico), $\boldsymbol{Q_{SV}}$ los predictores utilizados como vectores soporte, $\boldsymbol{Q_{predictor}}$ los predictores a identificar, $\boldsymbol{\beta_0}$ el sesgo y **N** el número de coeficientes.

El kernel empleado es una función polinómica de orden 3 que sigue esta fórmula:

$$G(Q_{SV}, Q_{predictores}) = (Q_{SV} \cdot \; Q_{predictores} + 1)^3 \tag{3-5}$$

La identificación del isótopo se realiza mediante la comparación del signo de la variable 'score' que es resultado de aplicar el algoritmo de SVM. Para este caso si el signo es positivo, los predictores de entrada están asociados a datos de ^{80}Kr, mientras que en el caso donde el signo sea negativo éstos estarían asociados a datos de ^{84}Kr.

Se puede comprobar que aplicar la función kernel implica realizar 6 sumas y 8 multiplicaciones. Asimismo, por cada iteración para el cálculo del 'score', se aplica una suma y una multiplicación, además de sumar el sesgo una vez calculado el sumatorio. Sin embargo, el número de sumas y multiplicaciones final dependerá del número iteraciones necesarias.

3.6 Análisis de la Discretización

En este apartado, se explicará la relación entre la cuantificación de variables y parámetros involucrados en el proceso de identificación de partículas y la precisión o desviación del valor calculado empleando aritmética de punto fijo frente a considerar aritmética flotante y precisión de 64 bits.

La predicción, sin cuantizar ningún parámetro, es del 99.75 %, utilizando todo el set de datos (2200

experimentos). Por tanto, dicho valor representa el valor ideal que se intentará no desviarse significativamente, al menos, no sobrepasar por debajo del 95 %.

La discretización o cuantización de los parámetros y algoritmos se aplica con el fin de reducir la carga computacional de los cálculos necesarios.

Sin embargo, el hecho de operar en punto fijo frente a emplear punto flotante introduce errores en la identificación, los cuales afectarán principalmente a los datos cercanos al hiperplano. Como ejemplo ilustrativo, la Figura 3-5 muestra un conjunto de datos que al ser discretizados sufren todos por igual cierto desplazamiento respecto al hiperplano. La mayoría siguen estando correctamente clasificados, mientras que uno se encuentra en la región opuesta a la que pertenece.

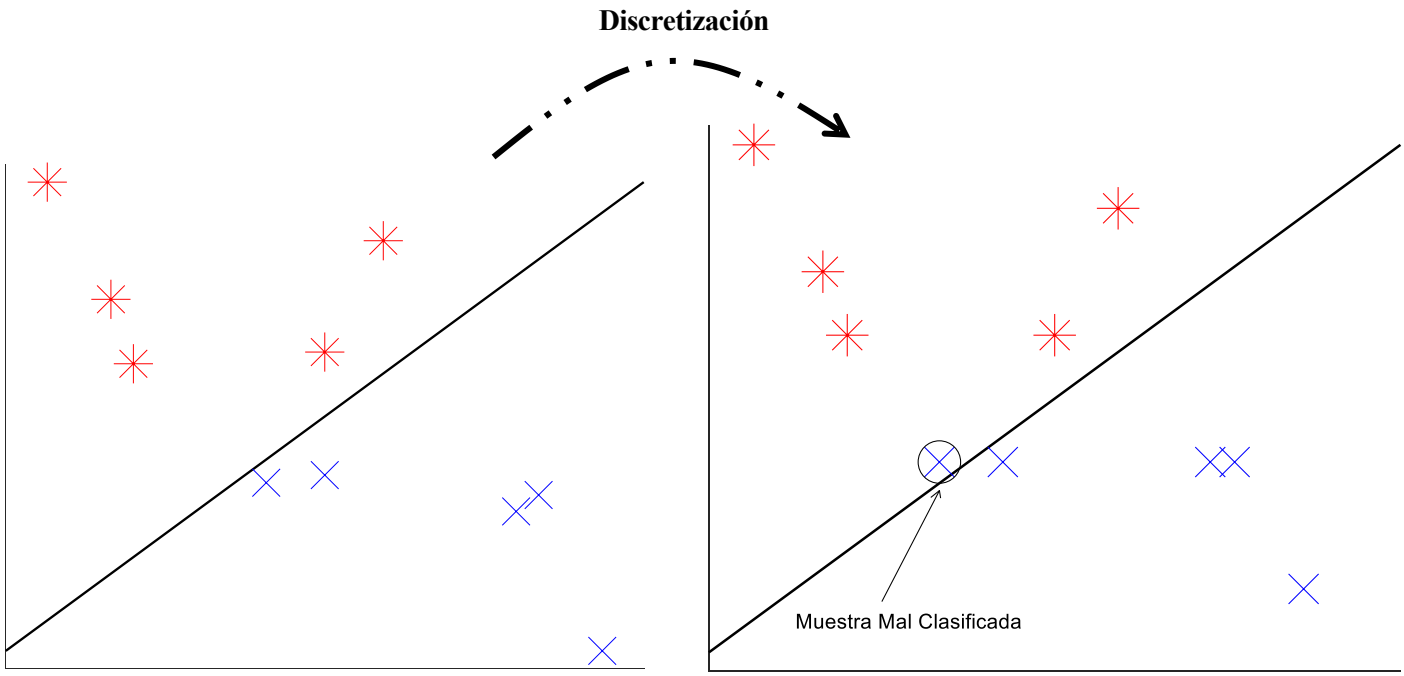

Figura 3-5. Ejemplo de inducción de error por discretización

Por esta razón, el análisis de la discretización se basará en cuánto empeora la predicción al cuantizar, en vez de analizar el error cometido en una variable intermedia.

3.6.1 Datos de entrada

Los datos de entrada proceden de la lectura de un CAD de 8 bits, por tanto, los datos que se procesarán y se enviarán al PCA tendrán la misma longitud. Pero antes de ser procesados por la PCA, a los datos de entrada se le restará su media global a cada característica, cuya resolución es de 8 bits.

Una vez restada dicha media, los datos de entrada conservarán los 8 bits de resolución.

Esto se debe a que, si se resta dos números positivos, el resultado será siempre menor en magnitud que cualquiera de los números del cálculo. Por tanto, la resolución de la diferencia conserva la resolución al tener ambos términos la misma longitud de bits.

Además, se observa que el máximo valor absoluto que alcanza esta variable no supera 16, por tanto, se dedicarán 3 bits para la parte fraccionaria, 4 bits para la parte entera y 1 para el signo.

3.6.2 PCA

El PCA tiene en cuenta 3 variables: los datos de entradas, los cuales tienen un tamaño 8 bits, como se ha mencionado anteriormente; los coeficientes de la matriz de transformación y los predictores (variables de salida del PCA). La resolución de las dos últimas variables se discutirá a continuación teniendo en cuenta las siguientes observaciones:

1. El máximo valor absoluto de los predictores que sea observado con los datos disponibles es el 147.6383. Este número es un valor puntual del predictor 1, cuya media es igual a cero y desviación típica 41.86. Por tanto, la probabilidad de que el predictor tome un valor superior al mencionado es del 0.02 %, altamente improbable. En consecuencia, se considera que la representación entera de los predictores debe ser de 8 bits, más el bit de signo.
2. El cociente entre el coeficiente de mayor magnitud frente al menor es de 5 órdenes de magnitud. Además, el rango entre valor máximo y mínimo es en torno a 0.3.
3. El valor absoluto de los coeficientes de la matriz PCA son inferiores a 1, por tanto, no tiene representación de la parte entera, exceptuando el bit de signo.

A continuación, se plantea realizar los sumatorios asociados al cálculo de la PCA con los siguientes enfoques: reducir el tamaño en bits de los predictores bien mediante la reducción de la precisión o resolución de los coeficientes, enfoque 1; o bien realizar truncamientos durante el sumatorio, enfoque 2. Para ello, se tendrá en consideración el truncamiento sólo con los bits asociados a la parte fraccionaria de cada una de las variables mencionadas.

Esta reducción de la resolución se realiza con el fin de tener un menor consumo de recursos al realizar el cálculo del sumatorio. Sin embargo, esta operación acarrea el problema de que el error por considerar una resolución insuficiente puede conllevar un mal cálculo de la predicción. Este error intencionado se ha planteado mediante dos enfoques para comprobar qué parámetro es más susceptible al error. Mediante el enfoque 1, donde sólo se introduce error en los coeficientes, o mediante el enfoque 2, el cual al truncar el sumatorio se reparte el error entre los coeficientes y los datos de entrada. De esta manera, se comprobará que elemento es más crítico para la identificación.

Los resultados de la cuantización se enviarán a un SVM que opera con punto flotante y utiliza la mayor resolución disponible. La comparación se basará en cuánto se diferencia la predicción a partir del valor ideal frente a la predicción a partir del valor cuantizado.

A continuación, se mostrará distintos ensayos, a través de las Figura 3-3, Figura 3-4, Figura 3-5 y Figura 3-6, que reflejarán la diferencia entre los enfoques mencionados anteriormente.

Cabe indicar que, en dichas figuras, la leyenda ‘PuntoFijo’ es relativa a la precisión de los resultados en operar en aritmética fija y sin realizar ningún truncamiento en los predictores. La leyenda ‘Truncamiento’ es relativa a la precisión de los resultados en operar en aritmética fija y con truncamiento. A su vez, se distingue dos resoluciones: los bits de los coeficientes de la matriz de transformación PCA (‘Frac PCA’) y los bits de los predictores tras el truncamiento (‘Frac SUM’). Finalmente, la curva etiquetada como ‘100% Aciertos’ representa las operaciones en punto flotante, cuya precisión real es del 99.75 %.

Además, es necesario mencionar que la parte fraccionaria del predictor sin truncar es la suma de la parte fraccionaria de los coeficientes más la de los datos de entrada (3 bits).

En estos dos primeros ensayos, reflejados en las Figura 3-6 y Figura 3-7, se mantendrá el mismo número de bits truncados en el sumatorio, mientras varía la resolución de los coeficientes de la matriz PCA.

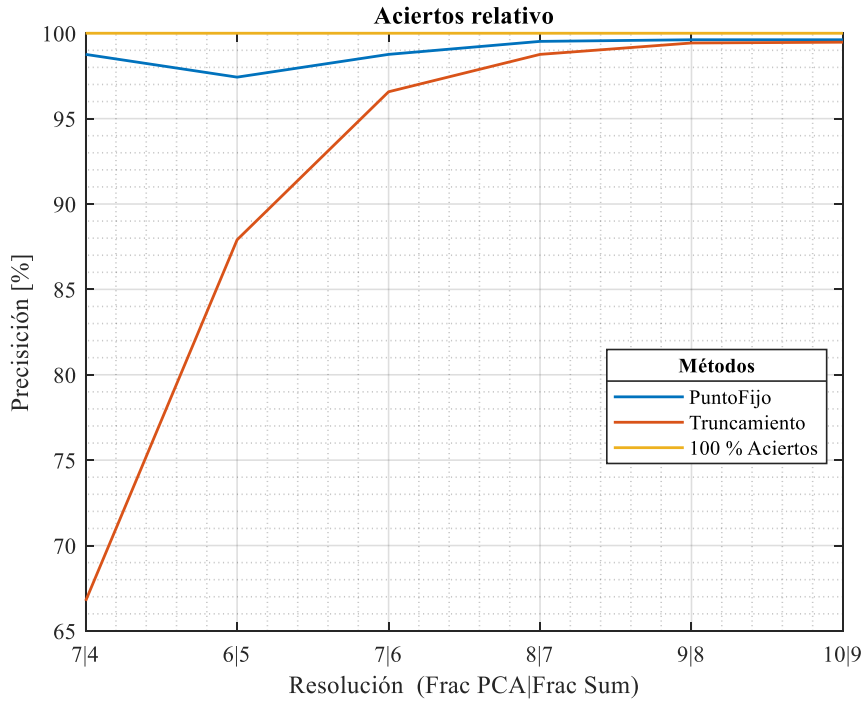

Figura 3-6. Representación del ensayo 1

En el ensayo 1, se prueba un rango amplio de resoluciones de los coeficientes para comprobar desde qué número de bits la precisión empieza a tender al valor ideal. Se observa que esto ocurre al utilizar más de 8 bits. El truncamiento se mantiene constante, despreciando 4 bits. De esta forma, la precisión, con el truncamiento, tiende al valor ideal cuando se emplea 9 bits en los coeficientes. Sin embargo, se aprecia que el truncamiento siempre tiene asociado una precisión menor

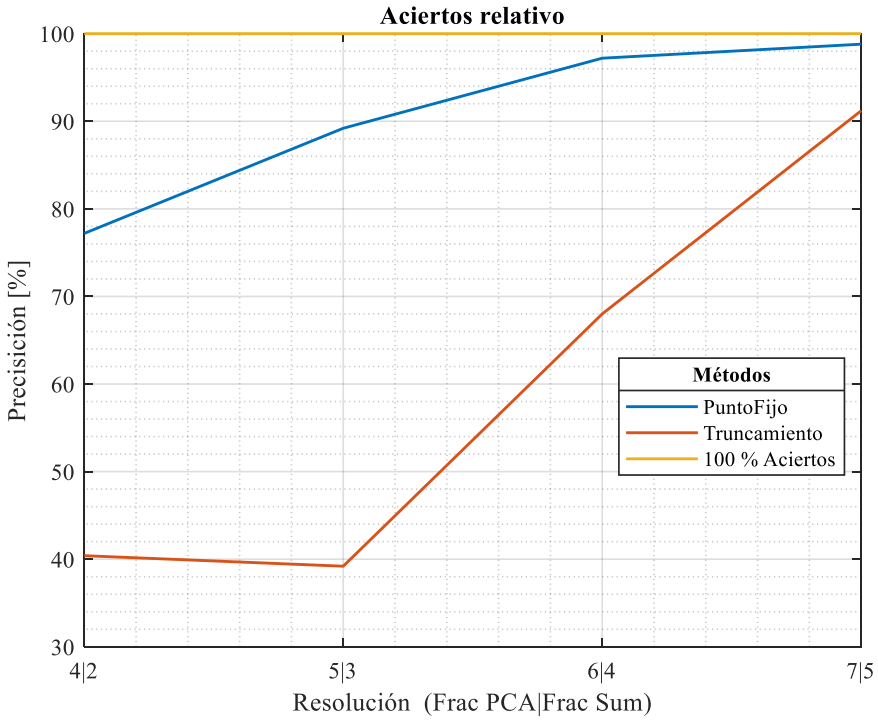

Figura 3-7. Representación del ensayo 2

En el ensayo 2 se realiza el mismo experimento que en el caso anterior, pero disminuyendo la resolución de los coeficientes y aumentando el truncamiento, de modo que, en esta situación se desprecia 5 bits. Se puede

comprobar que la precisión de la aritmética fija sin truncamiento permanece por encima del 95 % al emplear como mínimo 6 bits en los coeficientes. Además, se aprecia que el truncamiento realizado supone una pérdida considerable de precisión, de forma que ésta estaría por debajo del valor mínimo admisible para cualquier resolución.

En los dos siguientes ensayos, reflejados en las Figura 3-8 y Figura 3-9, se conservará la resolución de los coeficientes de la matriz PCA, mientras se trunca el predictor con diferentes números de bits durante el sumatorio.

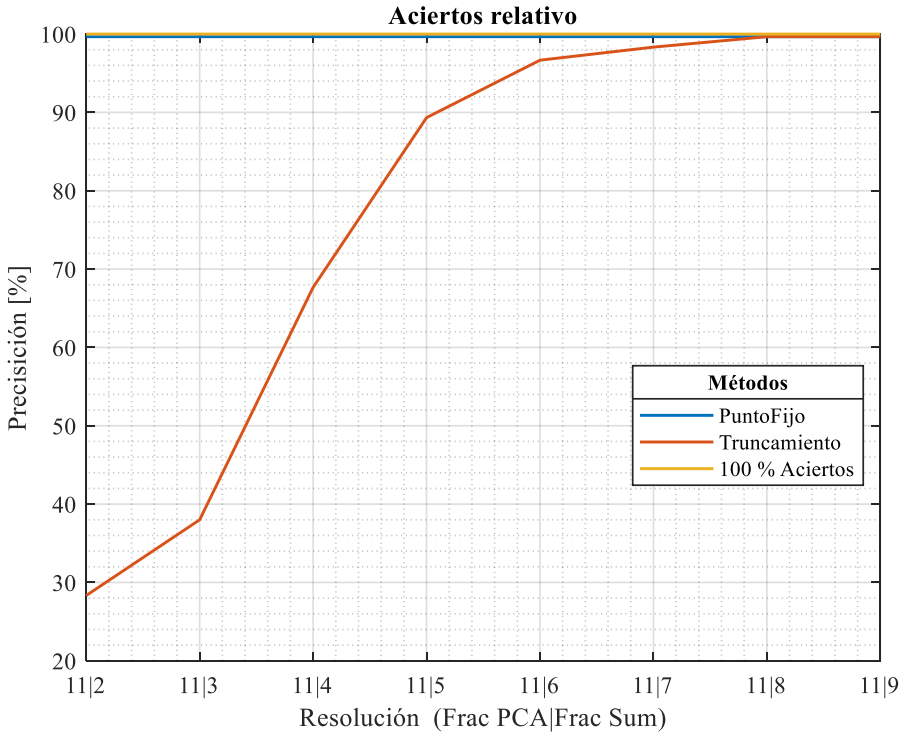

Figura 3-8. Representación del ensayo 3

En el ensayo 3 se toma un número de bits suficientemente grande para los coeficientes, de forma que la precisión se asemeje al comportamiento ideal, 11 bits. De este modo, al variar los bits truncados, se observa que si el predictor tiene menos de 6 bits en la parte fraccionaria la precisión cae por debajo del 95 %.

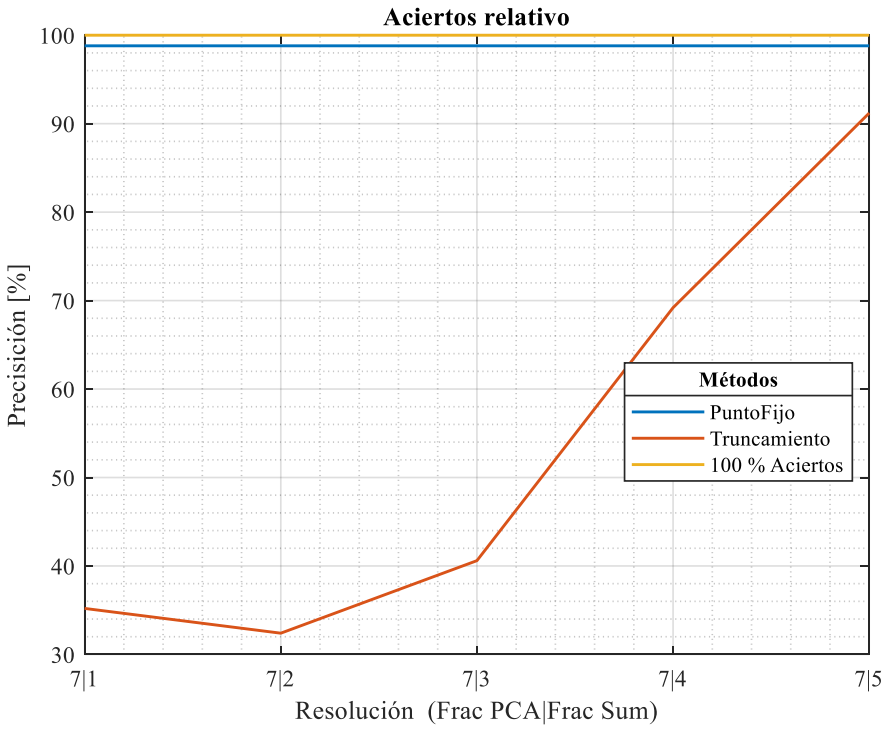

Figura 3-9. Representación del ensayo 4

En el ensayo 4 se conservará 7 bits para la resolución de los coeficientes y se probará con distintos truncamientos. Se observa que, al igual que en el ensayo anterior, con 6 bits en la parte fraccionaria el predictor no predice con la suficiente precisión.

Se puede comprobar que el cuello de botella reside en la precisión de la parte fraccionaria de los predictores, ya que la precisión de la predicción cae drásticamente cuando se realiza un truncamiento de más de 4 bits en el sumatorio. Por tanto, es preferible utilizar una resolución baja en los coeficientes y no realizar truncamiento. Un ejemplo de ello es que la precisión por utilizar 6 bits en los coeficientes y no emplear truncamiento es equivalente a tener coeficientes con longitud de 11 bits y utilizar solo 6 bits en la parte fraccionaria del predictor. Con la misma resolución y con el hecho de no emplear el truncamiento, se podría ahorrar casi el doble de bits.

Por tanto, tomando como criterio que la predicción debe ser mayor del 95 % y dejando cierto margen para las cuantizaciones futuras, se escoge los tamaños de palabras recogidos en la Tabla 3-2.

Tabla 3-2. Cuantización PCA

Variable	Tamaño total [bits]	Tamaño parte Fraccionaria [bits]
Coeficientes PCA	8	7
Predictores	17	9

3.6.3 SVM

El SVM emplea 3 variables: los coeficientes del SVM con su correspondiente signo ($\alpha_i \cdot y_i$), los vectores soporte y el sesgo. Además, antes de procesar es necesario normalizar los datos de entrada, es decir, restarles la media global y dividirlos por la desviación típica de los predictores (empleados para el proceso de

entrenamiento).

Antes de plantear la implementación, es necesario mencionar las siguientes observaciones:

1. Los coeficientes toman valores entre -1 y 1, algunos toman valores muy cercanos a 0, los cuales se pueden llegar a despreciar a la hora de la cuantización
2. La media global de los predictores es 0, por tanto, no sería necesario realizar ninguna resta.
3. La división de la desviación típica puede ser sustituida por un desplazamiento en bits hacia la derecha. El hecho de emplear esto implica una reducción considerable de recurso y coste computacional. Además, de esta manera se fuerza que el cociente resultante tenga una menor resolución, de esta forma, las siguientes etapas que tengan que trabajar con este dato demandarán un menor coste computacional.

Seguidamente, se realizarán simulaciones del cálculo del 'score', tomando distintas resoluciones de la parte fraccionaria para los parámetros involucrados en su cálculo. Además, de comprobar si la sustitución de la normalización por un desplazamiento de bits puede suponer una opción beneficiosa.

Las simulaciones se realizarán de la siguiente manera: por cada ensayo se cuantizará un único parámetro, mientras que el resto de las variables mantendrán su máxima resolución, incluido los predictores procedente de la PCA. Una vez analizadas todas las variables individualmente, se comprobará si el conjunto de las cuantizaciones no empeora considerablemente la predicción.

Es necesario mencionar que los coeficientes que se quedasen fuera, debido a la discretización, supondrían un considerable ahorro en memoria, ya que también se podría desechar el vector soporte asociado al coeficiente despreciado. Además, se ahorraría en realizar menos iteraciones, pero esta mejora no representa una ventaja significativa.

La sustitución de la operación de la división por el desplazamiento de bits hacia la derecha en la normalización de los predictores se basa en lo siguiente:

1. Se aproxima las desviaciones típicas de cada predictor a la potencia de 2 más cercana, este caso donde las desviaciones típicas son [49.5773 9.7642 8.1657 6.1322 4.2003 3.2871] pasarían a valer, [64 8 8 8 4 4], expresándolos como potencia de dos: $[2^6\ 2^3\ 2^3\ 2^3\ 2^2\ 2^2]$.
2. Seguidamente, al registro que alberga cada predictor se le realizará un desplazamiento de bits hacia la derecha tantas veces como sea el valor del exponente de la potencia de 2 anteriormente mencionada. De forma que, al primer predictor se desplazará 6 bits a la derecha, al segundo se desplazará 3 bits, con el tercero 3, y así con el resto de predictores

3.6.3.1 Discretización de los parámetros:

A continuación, se recopilan los resultados de distintos experimentos en los cuales se ha discretizado con un número de bits determinado los distintos parámetros involucrados.

Los ensayos se mostrarán por parejas, de forma que dentro del análisis de la discretización de cada parámetro habrá dos gráficas: una donde se realiza una normalización convencional y una segunda donde la normalización se sustituye por un desplazamiento de bits hacia la derecha. Esta última tendrá como distintivo el subtítulo 'Con desplazamiento de bits'.

3.6.3.1.1 Discretización de los coeficientes:

Las Figura 3-10 y Figura 3-11 mostrarán la dependencia de la precisión de la identificación respecto a la resolución de los coeficientes del SVM.

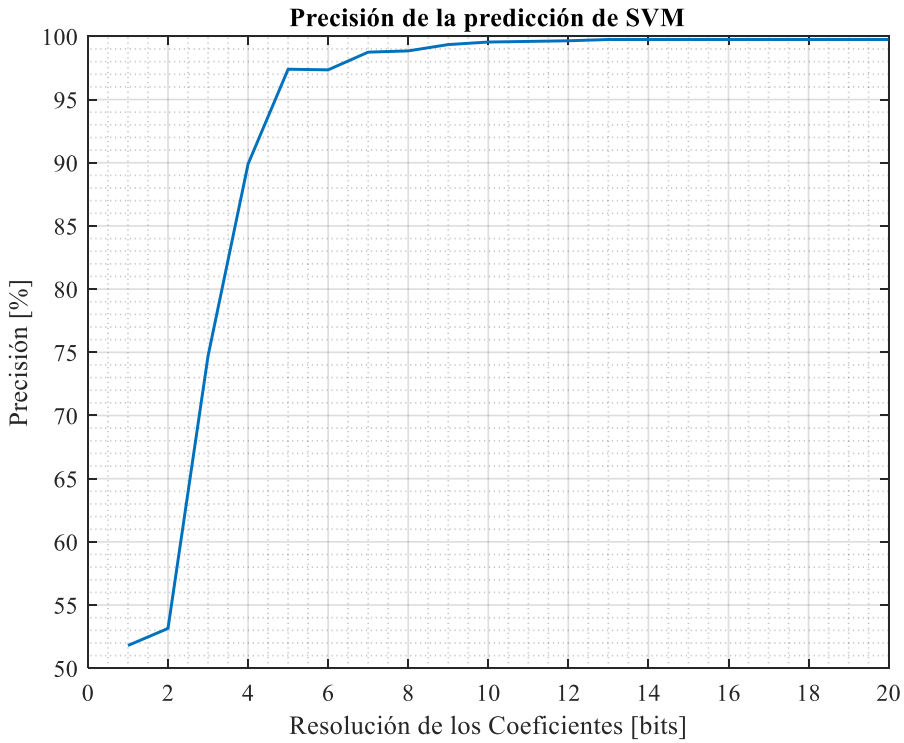

Figura 3-10. Representación del ensayo 5

En el ensayo 5 se aprecia que la precisión tiende al valor ideal cuando el número de bits es superior a 10. Además, por debajo de 6 bits, la precisión decrece considerablemente.

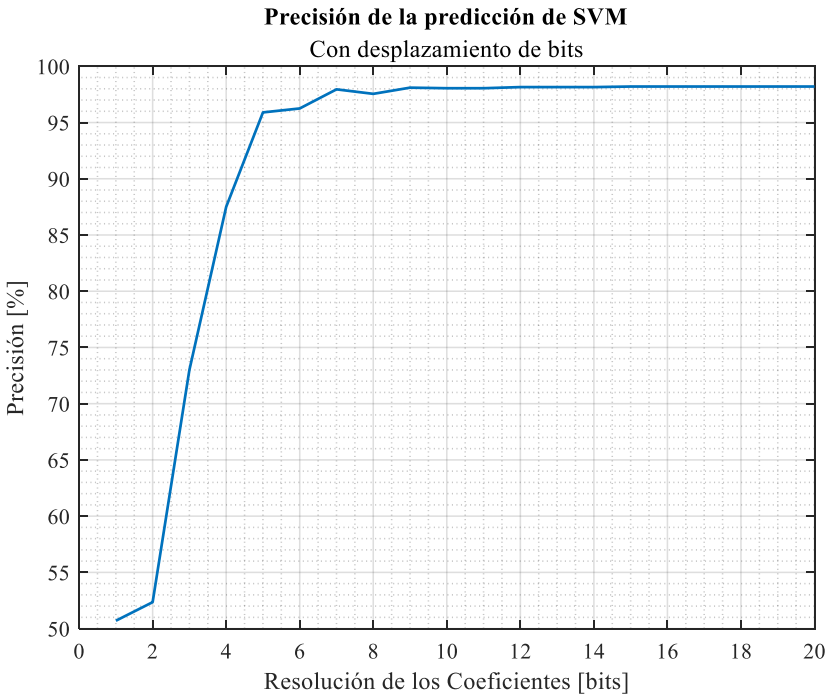

Figura 3-11. Representación del ensayo 6

En el ensayo 6 se observa el mismo comportamiento que en el ensayo anterior, aunque se destaca una reducción de 2 puntos en la precisión en todas las resoluciones.

3.6.3.1.2 Discretización de los vectores soporte (de cada componente):

Las Figura 3-12 y Figura 3-13 indicarán la relación entre la precisión de identificación y la resolución de los componentes de los vectores soporte.

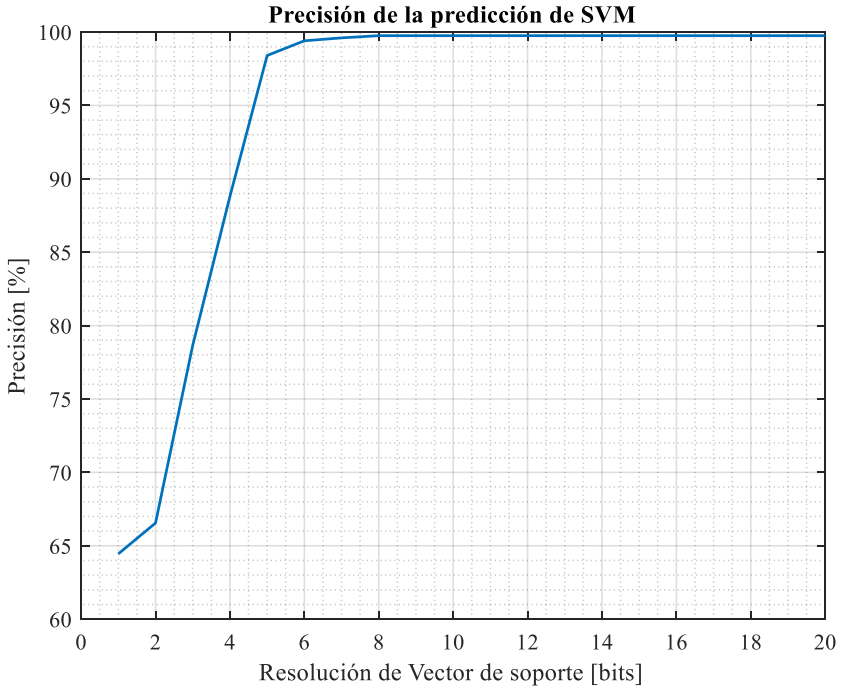

Figura 3-12. Representación del ensayo 7

En el ensayo 7 se puede comprobar que la precisión crece considerablemente al incrementar la resolución hasta 6 bits. A partir de dicho valor la precisión tiende lentamente al valor ideal.

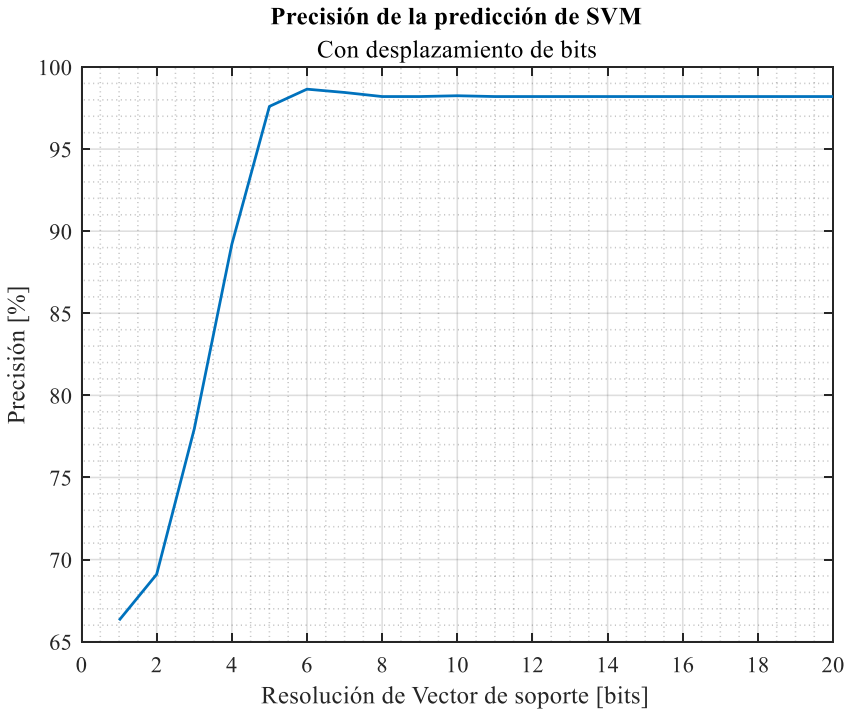

Figura 3-13. Representación del ensayo 8

En el ensayo 8 se observa la misma tendencia que en el caso anterior, y al igual que en el ensayo 6 el desplazamiento de bits disminuye en 2 puntos en la precisión.

3.6.3.1.3 Discretización del sesgo:

En las Figuras 3-14 y Figura 3-15 se mostrará el comportamiento de la precisión de la identificación respecto a variaciones en la resolución del sesgo.

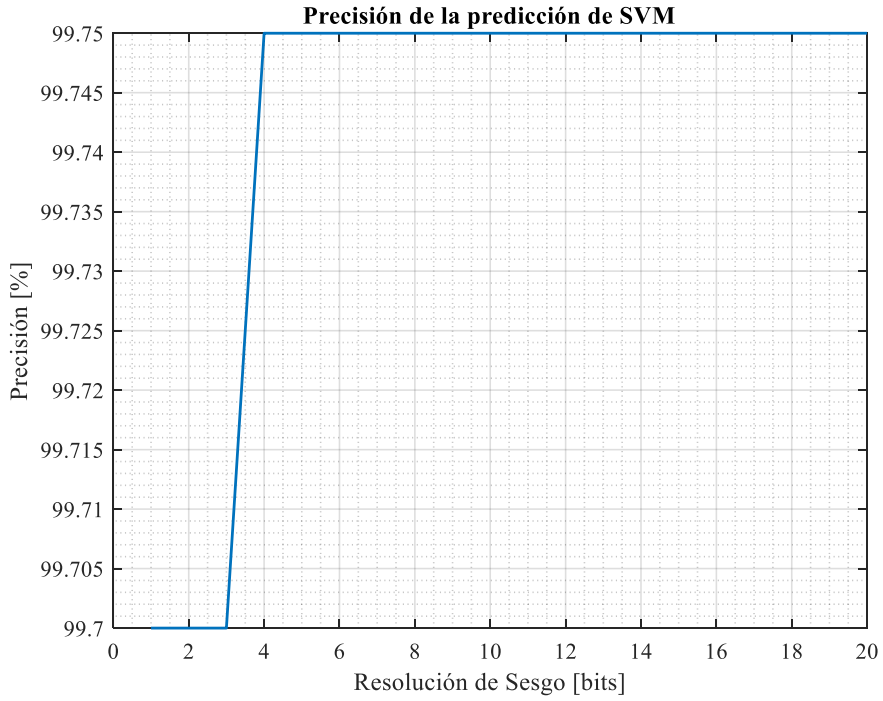

Figura 3-14. Representación del ensayo 9

En el ensayo 9 se puede comprobar que la precisión se aleja del valor ideal sólo un 0.05 % empleando menos de 4 bits.

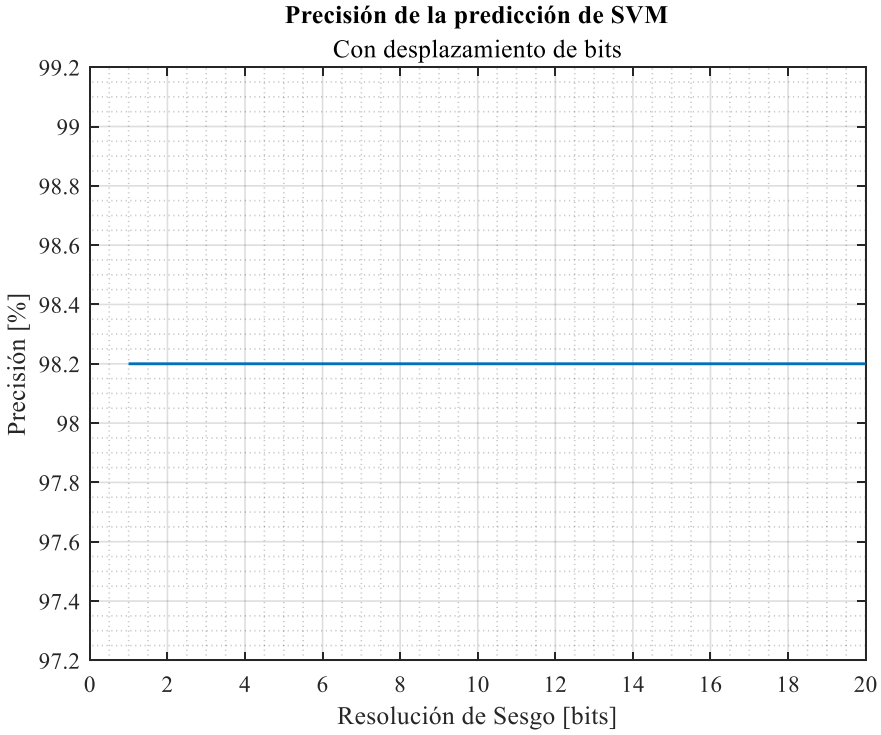

Figura 3-15. Representación del ensayo 10

En el ensayo 10 se aprecia que la precisión se mantiene constante para cualquier resolución. En este caso, la diferencia en la precisión respecto a la situación anterior es de 1.5 puntos.

En términos generales, se observa que la precisión es insensible a la resolución del sesgo, ya que las variaciones de la precisión son muy pequeñas (0.05 %) o inexistentes en el caso que se usase el desplazamiento de bits como normalización.

Por otro lado, se comprueba que tanto para el vector soporte como para los coeficientes la precisión máxima se alcanza a partir de una resolución de 8 bits.

Además, el empleo del desplazamiento de bits como sustituto de la normalización implica una reducción de la precisión de 2 puntos en todas las situaciones, excepto en el sesgo.

A continuación, se realizará un análisis de la precisión de la predicción considerando diferentes cuantizaciones para todos los parámetros. Además, los predictores involucrados estarán también determinados a partir del cálculo en punto fijo de la PCA, con las resoluciones mencionadas en el apartado anterior.

Al igual que en el análisis individual de cada parámetro se presentará una pareja de gráficas, las cuales mostrarán la diferencia entre utilizar o no desplazamiento de bits como sustituto de la normalización.

Las Figura 3-16 y Figura 3-17 son gráficas paramétricas donde se evaluará cómo se comporta la precisión de identificación con distintas resoluciones para los vectores soporte y coeficientes.

Para la resolución del sesgo se ha elegido 1 bit para la parte fraccionaria, ya que no supone una mejora significativa considerar más bits. Además, empleando desplazamiento de bits la precisión no aumenta con un número mayor de bits.

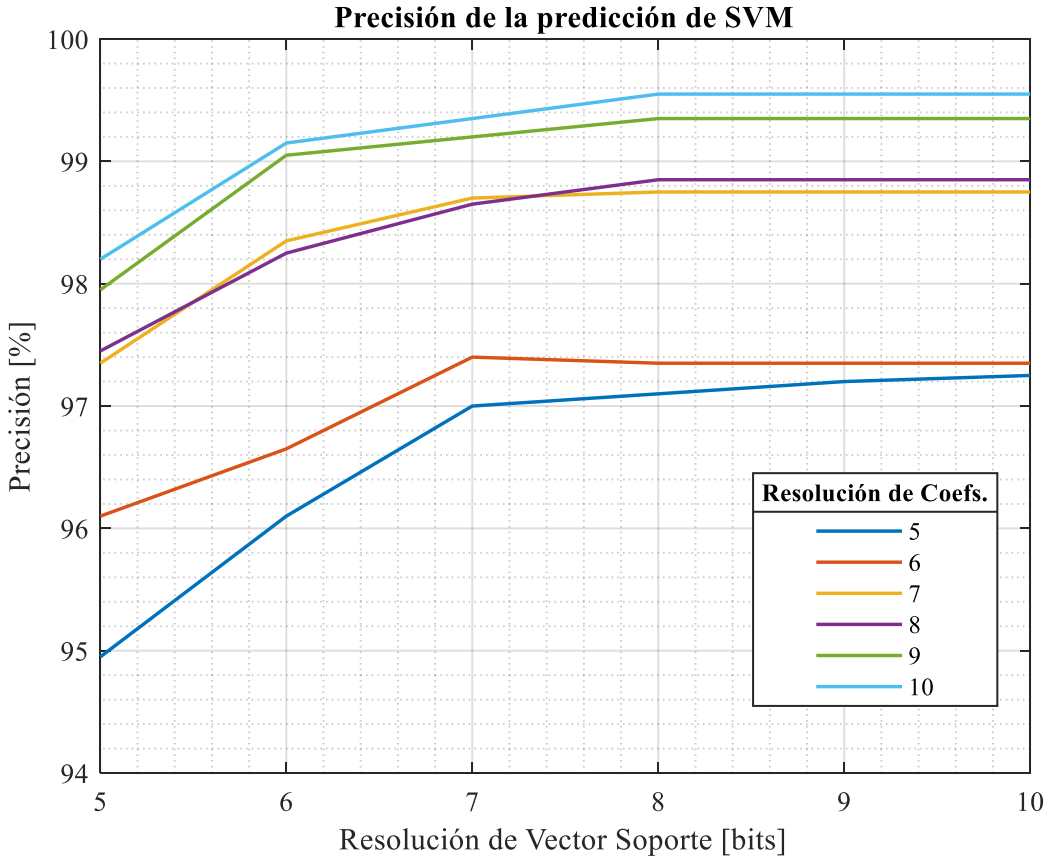

Figura 3-16. Representación del ensayo 11

En el ensayo 11 se observa que, a mayor resolución, tanto en el vector soporte como en los coeficientes, mayor precisión. Además, ésta se estabiliza cuando se utiliza más de 8 bits en los vectores soporte. A su vez, se puede comprobar que, en todos los casos, excepto donde la resolución del vector soporte y los coeficientes es igual a 5, se obtiene una precisión mayor al 95 %, y en la situación donde los parámetros tienen asociados 10 bits se alcanza una precisión cercana a la ideal, con un valor de 99.58 %.

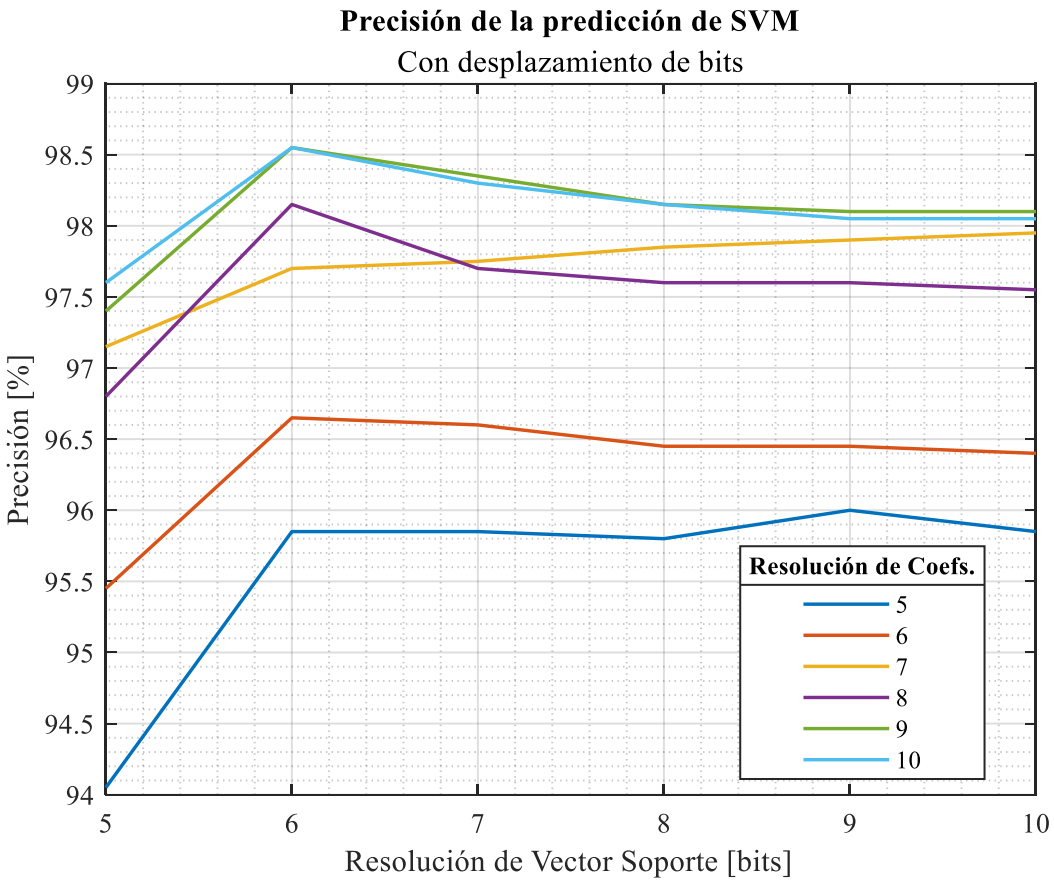

Figura 3-17. Representación del ensayo 12

En el ensayo 12 se aprecia un comportamiento similar respecto al caso anterior. La precisión se estabiliza utilizando más de 8 bits en la resolución en el vector soporte y todos los casos superan el 95 % de precisión, excepto el de menor resolución. Del mismo modo que en el resto de los ensayos donde se emplea el desplazamiento de bits, en general, la precisión disminuye ligeramente. Sin embargo, ésta no se aleja significativamente del valor ideal, alcanzando una precisión entorno al 98 % como valor máximo.

A la hora de elegir la resolución para cada elemento, es necesario tener en cuenta que existen 69 coeficientes y vectores soporte. Además, cada vector soporte está conformado por 6 componentes, por tanto, el hecho de considerar 1 bit más para la resolución del vector soporte implica un gasto de memoria de 414 bits o de 69 bits, en el caso que se aumentase la resolución de los coeficientes. Por esta razón, se realiza un análisis donde se comprueba qué combinación de discretizaciones posibles mejora la precisión de la identificación frente al espacio en memoria que ocupa los parámetros. La Tabla 3-3 recoge la estimación de memoria y precisión lograda para las diferentes combinaciones analizadas utilizando una normalización convencional.

Tabla 3-3. Resultados de predicción con normalización previa

Res. Vec. Sop. [bits]	Res. Coef [bits]	Num. Coef.	Memoria [bits]	Precisión [%]	Ratio Precisión/Memoria*1e6
5	5	63	2205	94.95	430.6
	6	63	2268	96.10	423.7
	7	66	2331	97.35	417.6
	8	66	2508	97.45	388.6
	9	68	2574	97.95	380.5

	10	68	2720	98.20	361
6	5	63	2583	96.10	372
	6	63	2646	96.65	365.3
	7	66	2709	98.35	363
	8	66	2904	98.25	338.3
	9	68	2970	99.05	333.5
	10	68	3128	99.15	317
7	5	63	2961	97.00	327.6
	6	63	3024	97.40	322.1
	7	66	3087	98.70	319.7
	8	66	3300	98.65	298.9
	9	68	3366	99.20	294.7
	10	68	3536	99.35	281

La Tabla 3-4 reúne el espacio en memoria estimado y la precisión obtenida para las distintas resoluciones analizadas anteriormente en la cuales se empleó el desplazamiento de bits.

Tabla 3-4. resultados de predicción sin normalización previa, sustituido por desplazamiento de bits

Res. Vec. Sop. [bits]	Res. Coef [bits]	Num. Coef.	Memoria [bits]	Precisión [%]	Ratio Precisión/Memoria*1e6
5	5	63	2205	94.05	426.5
	6	63	2268	95.45	420.9
	7	66	2331	97.15	416.8
	8	66	2508	96.80	386
	9	68	2574	97.40	378.4
	10	68	2720	97.60	358.8
6	5	63	2583	95.85	371.1
	6	63	2646	96.65	365.3
	7	66	2709	97.70	360.6
	8	66	2904	98.15	338

	9	68	2970	98.55	331.8
	10	68	3128	98.55	315.1
7	5	63	2961	95.85	323.7
	6	63	3024	96.60	319.4
	7	66	3087	97.75	316.7
	8	66	3300	97.70	296.1
	9	68	3366	98.35	292.2
	10	68	3536	98.30	278

Tomando como criterio que la predicción debe tener como mínimo un 95 %, tanto para el caso que se normalice con o sin desplazamiento de bits, el mejor escenario donde la relación precisión/memoria es mayor ocurre cuando la resolución del vector soporte es de 5 bits y la resolución de los coeficientes es de 6 bits, ambos en la parte fraccionaria.

La Tabla 3-5 recoge los tamaños de los parámetros del SVM analizado, de forma que se cumplan el requisito de tener una predicción superior al 95 %, con un 95.45 %.

Tabla 3-5. Cuantización SVM

Variable	Tamaño total [bits]	Tamaño parte Fraccionaria [bits]
Coeficientes SVM	8	6
Vectores Soporte	8	5
Sesgo	4	1

Otros cálculos que son necesarios tener en cuenta son el kernel polinómico de orden 3 y el sumatorio del SVM. Este primero involucra el cálculo del producto escalar entre dos vectores (predictor y vector soporte) y la multiplicación del resultado de éste 3 veces por sí mismo. Esta operación generaría una variable de 90 bits, ya que el producto escalar tendría como resultado una variable de 30 bits. Seguidamente, éste se multiplicaría por su coeficiente del SVM correspondiente y, finalmente, se implementaría dentro del sumatorio del SVM obteniendo de esta manera el 'score'. Con la cuantización de los coeficientes de los SVM explicados anteriormente, se realizaría 63 iteraciones, ya que se podría despreciar 6 coeficientes, y el 'score' tendría una resolución de 160 bits. Esta resolución está lejos de ser óptima, ya que solo el bit de signo es necesario para la categorización e identificación final.

Por tanto, al igual que se hizo con el PCA, se analizará distintos enfoques para reducir la resolución de la variable 'score': Sustituir el cálculo cúbico por uno lineal, cuando los predictores se encuentren 'muy alejados' del hiperplano (vectores soporte), enfoque 1. Truncar el cálculo de cada variable intermedia (el productor escalar y el 'score' durante el sumatorio), enfoque 2.

Además, una forma de reducir la resolución del 'score' es emplear el método de desplazamiento de bit como normalización, ya que la resolución de los predictores de entrada sería de 13 bits.

Mediante el enfoque 1, si el predictor está lo suficientemente lejos del hiperplano, el 'score' asociado será significativo, en este caso, tendría un valor muy distinto de 0. Por tanto, al aproximar la función kernel a un cálculo lineal, aunque se introduzca cierto error, el 'score' resultante será capaz de poder clasificar correctamente el predictor de entrada. Por esta misma razón, si los predictores tienen asociados 'score' cercano a 0 (cerca del

hiperplano) será necesario utilizar el kernel polinómico de orden 3 para no perder precisión.

La función kernel seguía la fórmula 3-4, de forma que si se realiza la siguiente sustitución se obtendría la ecuación 3-6:

$$\vec{w} = Q_{SV}$$
$$\vec{x} = Q_{predictores}$$

$$G(\vec{w}, \vec{x}\,) = (\vec{w} \cdot \vec{x}\,)^1 = (\,w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6) \tag{3-6}$$

Para el enfoque 1 se empleará el producto escalar entre el predictor de entrada y los vectores soporte como indicativo de la lejanía respecto al hiperplano. De manera que, si el resultando de éste es superior a 1 se aplicará el cálculo lineal. Se toma este valor como frontera porque, de esta forma, se estaría trabajando con la operación al cubo sólo con los bits asociados con la parte fraccionaría la cual ocupa 42 bits, un 54% menos respecto a la situación inicial

Seguidamente, se simula el comportamiento del SVM con las cuantizaciones de la tabla 3-4 y mediante ese enfoque, y se observa que la precisión cae drásticamente a un 51 %, ya que todos los 'score' calculados tienen el mismo signo.

Con la sospecha de que el valor límite impuesto anteriormente es demasiado bajo se repite la simulación cambiando dicho valor. Además, se comprobará las veces que se cumple la condición de realizar el cálculo lineal.

En la Tabla 3-6 se presentarán el porcentaje de veces que se ejecutó el cálculo lineal frente al total de número de veces que es necesario aplicar la función kernel en este ensayo, junto con la precisión obtenida.

Tabla 3-6. Resultados del cálculo lineal

Valor límite	cálculo lineal [% de las operaciones totales]	Precisión [%]
2	21	58
4	1.9	85.2
8	0.026	94.9
16	0.0036	95.45

Se observa que la linealización de la función Kernel no empeora significativamente la predicción cuando el valor límite es superior a 8. Por tanto, la variable resultante del producto escalar tendrá 54 bits como máximo después de aplicar la operación al cubo en el caso que se implementará como 8 el valor límite.

Mediante el enfoque 2, se mantendrá la operación al cubo, pero se truncará el cálculo del producto escalar y el sumatorio del SVM. El truncamiento se realizará sabiendo que el valor del 'score' durante el sumatorio sólo alcanza valor por debajo de 64 en valor absoluto, y el producto escalar por debajo de 32. De este modo, para la parte entera del 'score' se utilizará 6 bits y para la del resultado del producto escalar se empleará 5 bits.

A continuación, se realizarán simulaciones para analizar la precisión de la identificación para distintas resoluciones, como se verá representado en la Figura 3-18.

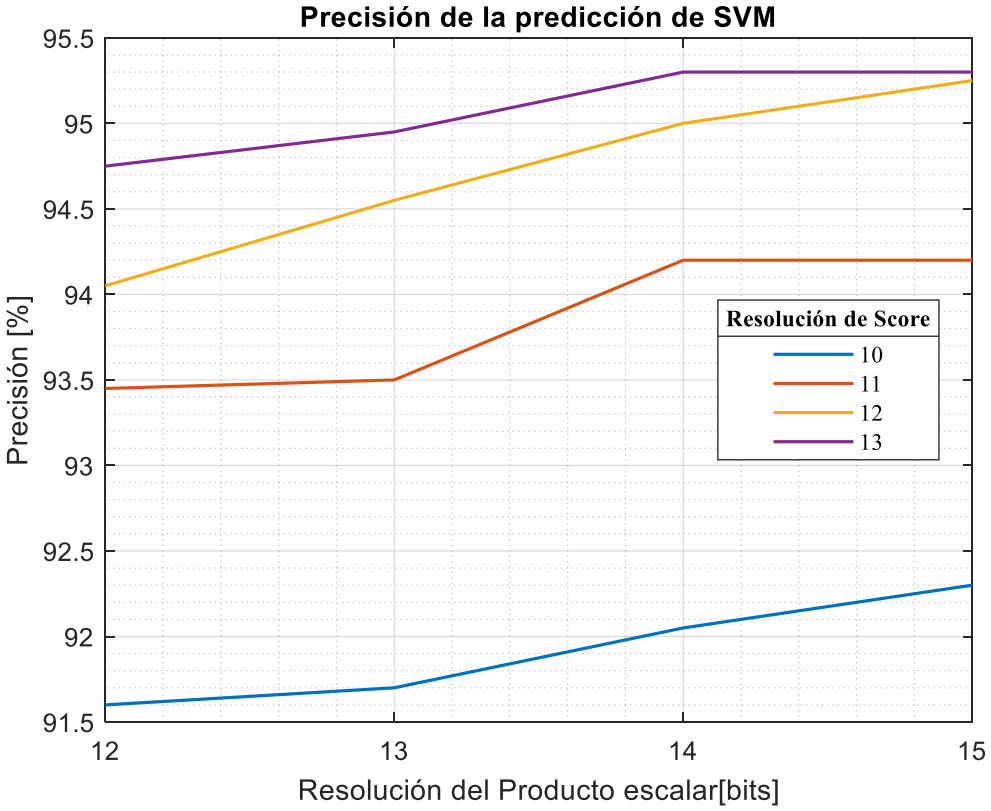

Figura 3-18. Truncamiento en la función Kernel

Como se puede comprobar en la Figura 3-188, utilizando 14 bits como resolución del producto escalar (que supone 42 bits después de la operación al cubo) y 10 bits en la resolución del score, se obtiene una precisión semejante a la obtenida anteriormente, un 95.3 % de precisión en la identificación.

Empleando ambas técnicas se podrá reducir en mayor medida el coste computacional. En las Figura 3-19, Figura 3-20 y Figura 3-21 se mostrará cómo se comporta la combinación de estas técnicas.

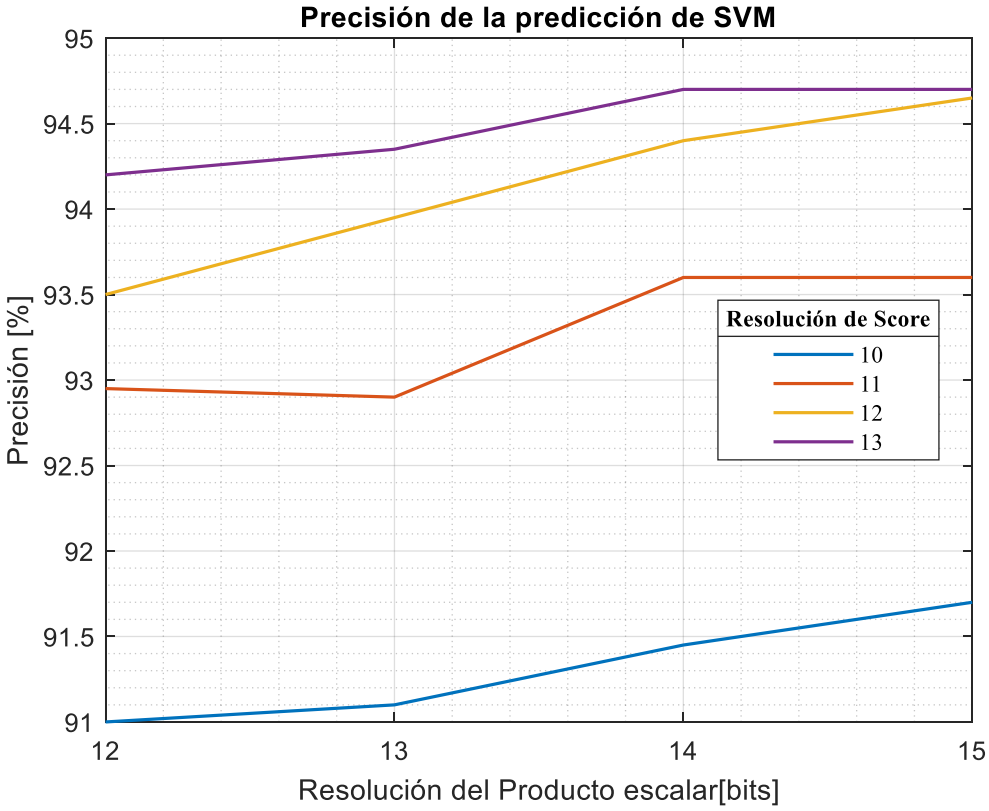

Figura 3-19. Truncamiento y 9 como valor límite

La Figura 3-19 muestra el comportamiento de la fusión de las técnicas mencionadas con anterioridad. Se observa que se obtiene unos datos similares a los mostrados en la Figura 3-18, con la diferencia que existe una reducción generalizada en la precisión de 0.6 puntos.

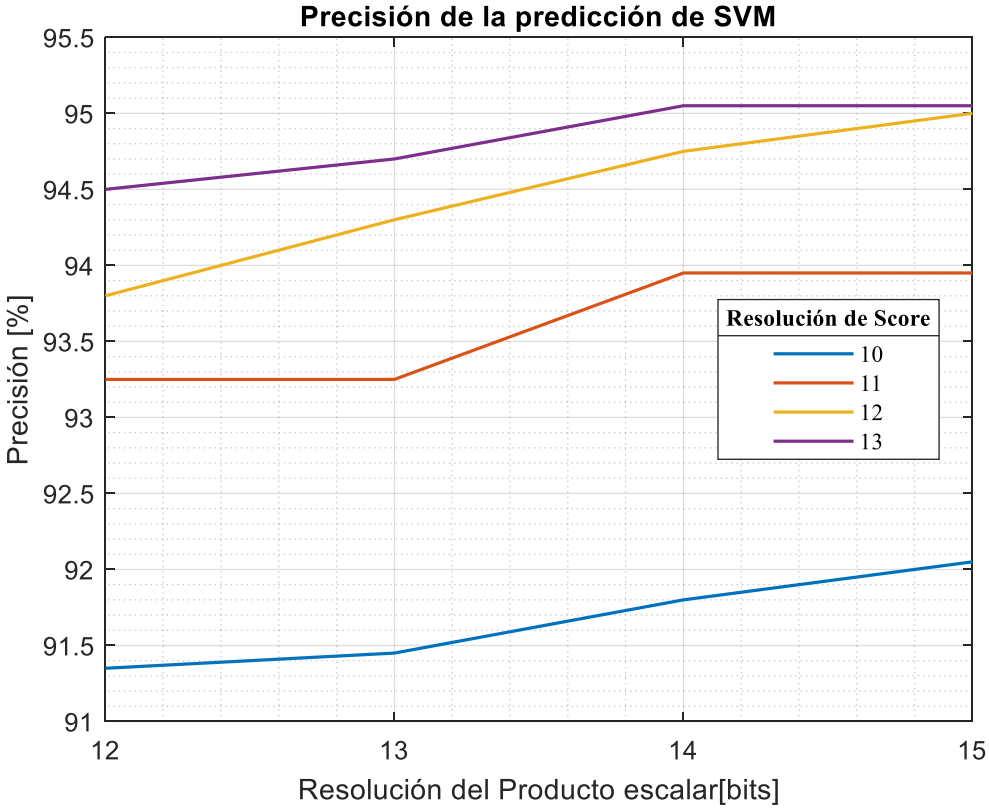

Figura 3-20. Truncamiento y 10 como valor límite

Los datos de la Figura 3-20 presentan una conducta similar a los anteriores, siguiendo la misma tendencia. Sin embargo, en este caso la diferencia de precisión respecto al primer caso es de 0.25 puntos.

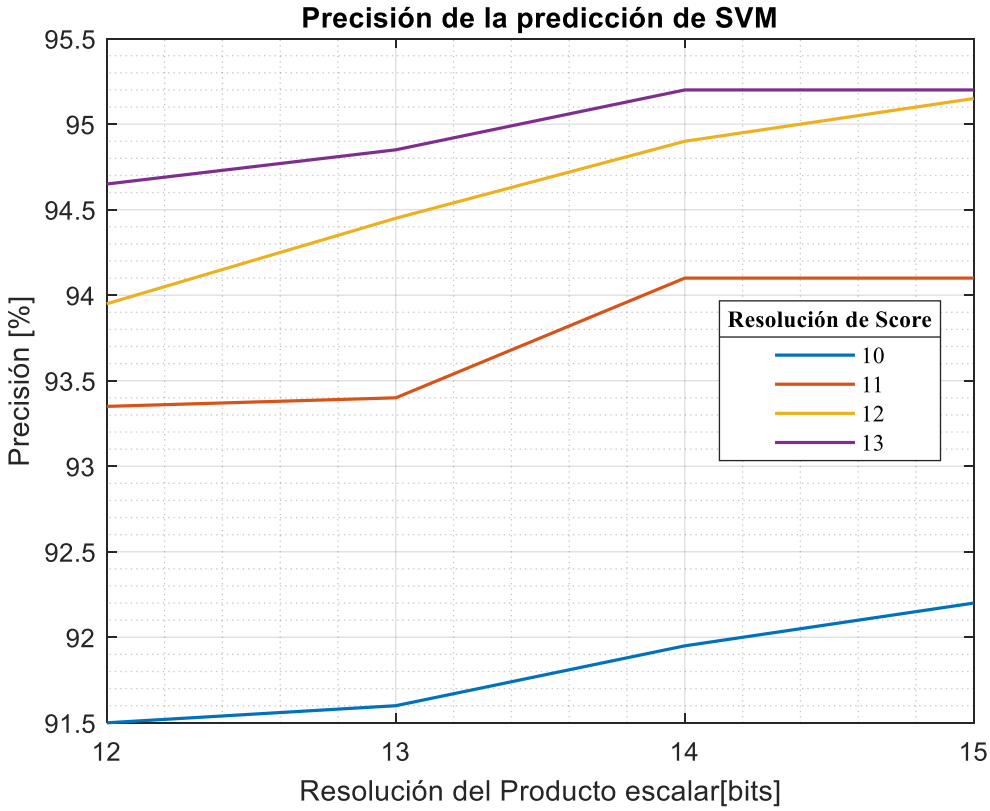

Figura 3-21. Truncamiento y 12 como valor límite

En la situación de la Figura 3-21 la precisión es casi idéntica a la mostrada en la Figura 3-18, ya que la diferencia entre ambas predicciones está por debajo de 0,1 puntos.

A través de los datos representados en las Figura 3-16, Figura 3-17 y Figura 3-18 se puede comprobar que la combinación de ambos enfoques ofrece una posibilidad factible de implementación. En términos generales, el hecho de añadir la linealización de la función kernel, supone, únicamente, una ligera pérdida de la precisión, sin modificar su comportamiento general respecto a la resolución de las variables analizadas.

Empleando ambas técnicas, se podría obtener una precisión del 95.05 % en el caso donde se truncase el resultado del producto escalar a 14 bits y, posteriormente, en el sumatorio se volviese a truncar, de forma que el 'score' tuviera 10 bits de resolución. Además de poner como valor umbral 10 entre utilizar el kernel polinómico de orden 3 y su versión lineal. Sin embargo, para tener un margen de seguridad en la identificación de partículas se preferirá establecer como valor umbral 12, de este modo, se conseguiría una precisión del 95.21 %

3.7 Conclusiones de la pre-Implementación

Como resultado de la pre – implementación, se comprueba que la realización de algoritmos relacionados con técnicas de 'Machine-Learning' mediante aritmética en punto fijo, tiene asociado un cálculo de la precisión de la identificación relativamente alto, con un 95.21 % de aciertos. Además, se ha logrado disminuir el coste computacional, en términos de número de operaciones, gracias a la aplicación de simplificaciones como la sustitución de la normalización de los predictores por el desplazamiento de bits hacia la derecha y la aproximación lineal de la función kernel.

A su vez, gracias a este estudio previo, se ha conseguido reducir tanto el posible espacio en memoria que requerirían los parámetros involucrados, como los recursos necesarios en el cálculo de las técnicas de 'Machine-Learnig',

4 RESULTADOS

Una vez analizados la relación entre la resolución de los datos y algoritmos estudiados frente a la precisión en la identificación de partículas en el capítulo anterior, en éste se estudiará los resultados obtenidos y se comprobará la manera más eficiente de implementarlos.

4.1 Presentación de los datos recopilados

A continuación, en la Tabla 4-1 se presenta la resolución elegida para cada parámetro involucrado en la identificación de partículas.

Tabla 4-1. Cuantización final

Variable	Tamaño total [bits]	Tamaño parte Fraccionaria [bits]
Datos de Entrada	8	3
Coeficientes PCA	8	7
Predictores	17	8
Coeficientes SVM	8	6
Vectores Soporte	8	5
Sesgo	4	1
'Score'	10	3

Con esta resolución se obtiene un 95.21 % de precisión en la identificación, lo cual supone solo una pérdida de 4.54 puntos respecto a usar punto flotante y una resolución de 64 bits para cada variable.

La Tabla 4-2 reúne la memoria ocupada por cada elemento.

Tabla 4-2. Memoria ocupada por las variables

Variable	Tamaño total [bits]	Memoria Ocupada [bits]
Datos de Entrada	8	2400
Coeficientes PCA	8	14400
Predictores	17	102
Coeficientes SVM	8	504
Vectores Soporte	8	3024
Sesgo	4	4
'Score'	10	10

La Tabla 4-3 recoge las operaciones necesarias para la aplicación de cada algoritmo.

Tabla 4-3. Operaciones para la implementación

	Sumas	Multiplicaciones
Cálculo de los Predictores (PCA)	1794	1800
Cálculo del 'score' (SVM)	483	552 (*)
Total	2304	236

(*) Se supone el caso donde todas las aplicaciones se kernel no se linealizan y la que se aplica la operación al cubo.

Por último, la Tabla 4-4 presenta los operadores necesarios para la aplicación de cada algoritmo.

Tabla 4-4. Operadores para la implementación

	Sumadores	Multiplicadores
Cálculo de los Predictores (PCA)	12	12
Cálculo del 'score' (SVM)	7	9
Total	19	21

4.2 Análisis de los resultados

4.2.1 Datos

El mayor gasto de coste computacional reside principalmente, en el cálculo de predictores debido al alto número de coeficientes de la matriz de transformación PCA. Observar que un aumento de un 1 bit más en estos coeficientes implica un gasto 0.1 % de la memoria RAM máxima disponible. Además, este proceso supone más del 77 % de las operaciones implementadas.

En el caso de usar desplazamiento de bits para la normalización, la resolución de los predictores pasaría a ser de 13 bits de longitud total. Esto reduciría significativamente la carga computacional en el cálculo del 'score' en el SVM.

4.2.2 Operadores

4.2.2.1 Multiplicadores

En cuanto a los recursos necesarios por los operadores dependerán del método de implementación. Los multiplicadores pueden ser multiplicador Booth, multiplicador Baugh-Wooley y el multiplicador de Vivado HLS. Existen una mayor tipología de multiplicadores, pero estos son capaces de trabajar con números con signo en complemento a 2.

Además, se considera el doble de operadores en la PCA, ya que en los bloques de almacenamiento en VHDL se puede extraer dos datos a la vez, por tanto, por linealidad, se puede descomponer los sumatorios asociados a los predictores en 2 mitades de igual longitud.

4.2.2.1.1 Multiplicador Booth (paralelo)

El algoritmo para la implementación de este multiplicador se basa en la observación que dicho producto se puede calcular a partir de sumas y restas. De forma que, éste estará formado por celdas de control de adicción,

sustracción y desplazamiento (CASS).

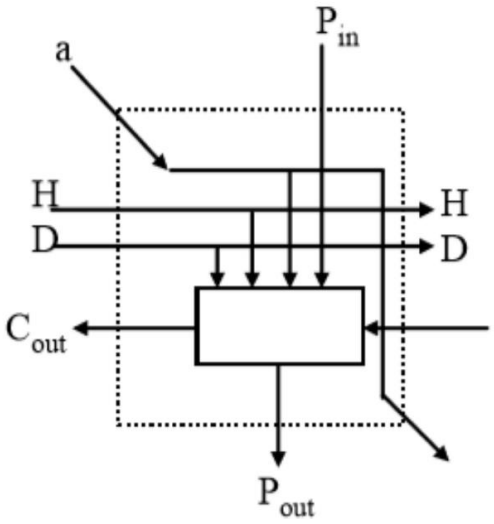

Figura 4-1. celda CASS

El algoritmo necesario para la implementación sería el siguiente:

$$P_{out} = P_{in} \oplus (a * H) \oplus (C_{in} * H) \tag{4-2}$$

$$C_{out} = (P_{in} \oplus D) * (a + C_{in}) + a * C_{in} \tag{4-3}$$

Se puede observar que si $H = 0$ el multiplicador realizaría un desplazamiento. En el caso donde $H = 1$ se realizará una suma, si $D = 0$ o una restra si $D = 1$.

Como ejemplo de implementación se muestra la Figura 4-2 un multiplicador de Baugh-Wooley de 4 bits:

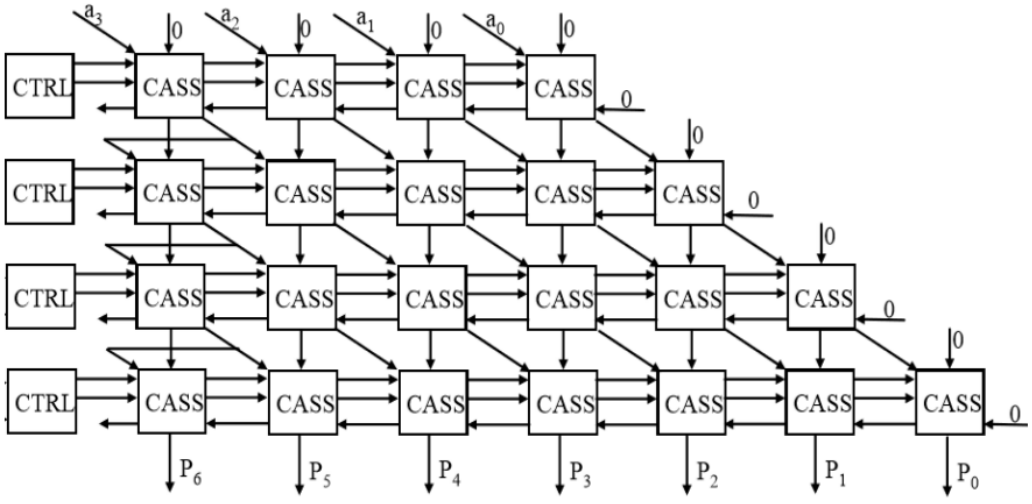

Figura 4-2. Ejemplo de implementación de multiplicador Booth (imagen extraída de [11]).

La Tabla 4-5 muestra los recursos necesarios para la implementación de este multiplicador para distintos bits.

Tabla 4-5. Recursos de un multiplicador Booth

Bits por factor	LUTs	FFs
4 bits	17	15
8 bits	126	31
16 bits	569	64

4.2.2.1.2 Multiplicador Baugh-Wooley

La implementación de multiplicador se basa únicamente en sumadores completos (Full-Adder - FA).

El algoritmo necesario para la implementación sería el siguiente:

$$A \cdot B = \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} 2^{i+j} a_i b_j + 2^{2n-2} a_{n-1} b_{n-1} + 2^{n-1}\sum_{j=0}^{n-2} a_{n-1} b_j 2^j + 2^{n-1}\sum_{i=0}^{n-2} b_{n-1} a_i 2^i + 2^n - 2^{n-1} \quad (4\text{-}4)$$

Como ejemplo de implementación se muestra la Figura 4-3 un multiplicador de Baugh-Wooley de 4bits:

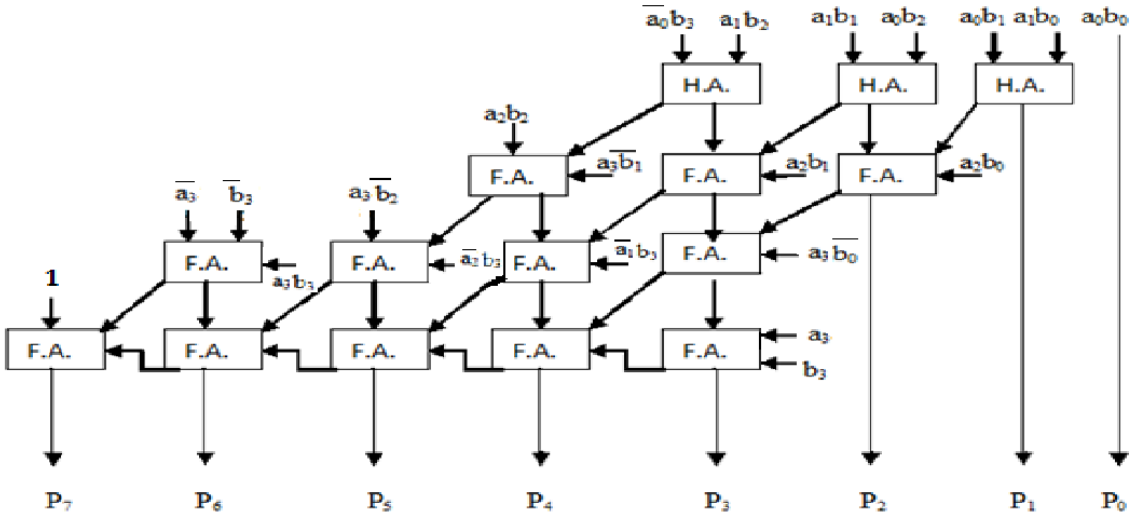

Figura 4-3. Ejemplo de implementación de multiplicador Baugh-Wooley (imagen extraída de [12])

La Tabla 4-6 muestra los recursos necesarios para la implementación de este multiplicador para distintos bits.

Tabla 4-6. Recursos de un multiplicador Baugh-Wooley

Bits por factor	LUTs	FFs
4 bits	18	16
8 bits	75	32
16 bits	393	64

4.2.2.1.3 Multiplicador de Vivado HLS

Vivado cuando detecta una operación de multiplicación entre dos factores es capaz de sintetizar una arquitectura propia para la operación.

La implementación de este multiplicador para distintos bits se recoge en la Tabla 4-7:

Tabla 4-7. Recursos de un multiplicador de Vivado HLS

Bits por factor	LUTs	FFs
4 bits	23	16
8 bits	63	36
16 bits	269	67

Se puede comprobar que el multiplicador de Vivado HLS optimiza el número de LUTS de la FPGA frente al resto de multiplicadores, mientras que el multiplicador de Baugh-Wooley emplea un menor número de flip-flops. Para la implementación final, sabiendo que la Arix-7 tiene un menor número inferior de LUTs que de flip-flops que usarán multiplicadores de Vivado HLS. [11].

Como primera aproximación y suponiendo el peor caso, se considerará que todas las multiplicaciones necesarias para la implementación tendrán el mismo consumo de recursos que un multiplicador de Vivado HLS de 16 bits.

4.2.2.2 Divisor

La operación de normalización, como se comentó con anterioridad, debe dividir los datos de entrada, y en la arquitectura de Vivado HLS no existe una implementación directa de la operación de división. Por tanto, se puede plantear dos enfoques:

1. Sustituir la división por un desplazamiento de bits hacia la derecha, asumiendo el error que pueda introducir.
2. Crear un algoritmo que se asemeje lo máximo a la división, asumiendo el coste de recurso que este requiera.

Para la implementación divisor se ha utilizado el código disponible en el Anexo.

En la Tabla 4-8 se indica la comparativa entre emplear un divisor y el desplazamiento de bits.

Tabla 4-8. Recursos para la normalización

Algoritmo	LUTs	FFs
Desplazamiento	0	0
División	88	210

Como se mencionó con anterioridad la diferencia entre usar desplazamiento de bits y usar una división (en punto flotante) era de un 2 % menos en la precisión en la identificación de la partícula. Pero el elevado consumo de recursos no compensa la utilización de una célula especializada, ya que el desplazamiento de bits supone realizar esta operación sin coste.

4.2.2.3 Sumadores

Para las implementaciones de las sumas se empleará el sumador que sintetiza el propio Vivado HLS el cual utiliza un sumador 'Carry Save' conformado por una etapa de sumadores completos seguido de un sumador 'Ripple Carry'.

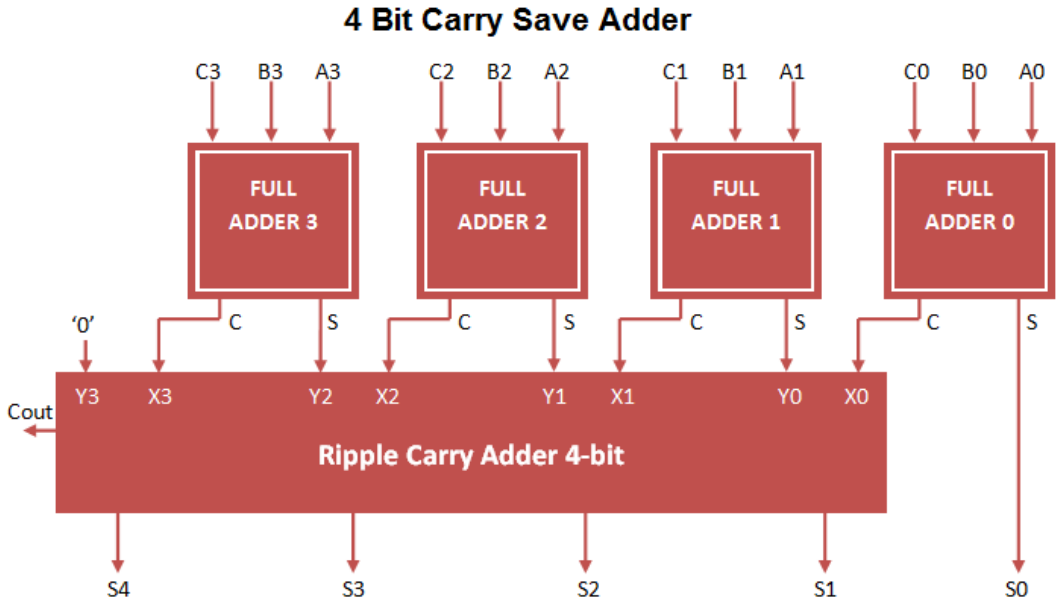

Figura 4-4. Ejemplo de Sumador 'Carry Save' de 4 bits

Vivado crea el sumador de forma que el número de flip-flop sea igual a la resolución de la variable de salida y el número de LUTS igual a la máxima resolución de los términos de la suma. De esta forma, la Tabla 4-9 reúne los recursos estimados para la suma de cada algoritmo.

Tabla 4-9. Recursos consumidos por las sumas

Algoritmo	LUTs	FFs
PCA	306	306
SVM (Kernel+Sumatorio)	255+34	265+10
Total	595	581

A continuación, se comprobará si es posible albergar la demanda de recursos solicitados, sabiendo que la FPGA posee 41600 flip-flops, 20800 LUTs y 1800 Kbits de memoria RAM.

La Tabla 4-10 recogerá los recursos estimados para las operaciones que conforman los algoritmos de 'Machine Learning' y se comprobará si son admisibles.

Tabla 4-10. Recursos consumidos por todos los operadores

	LUTs	FFs
Multiplicaciones	5649	1407
Sumas	595	581
Total requerido	6244	1985
Disponible	20800	41600

Se observa que la implementación, a priori, es completamente viable, ya que los recursos que ofrecen la FPGA satisfacen ampliamente los demandados por los operadores y por el almacenamiento de los datos

5 IMPLEMENTACIÓN EN FPGA

En este capítulo se analizará las características y recursos que ofrece la FPGA empleada en este proyecto, la Artix-7 (XC7a35T-1CPG236C) implementada dentro de la placa Basys 3.

5.1 Características de la FPGA

El dispositivo empleado para la implementación de la identificación de partículas será la placa BASYS 3 basada en la Artix-7 XC7A35T 'Field Programmable Gate Array' (FPGA). Esta presenta las siguientes características:

- 33,280 células/celdas lógicas en 5200 slices (cada slice contiene 4 LUTS de 6 entradas y 8 flip-flops).
- 1,800 kbits de bloques de RAM (BRAM) rápidos.
- Reloj interno con velocidades superiores a 450 MHz.
- 5 bloques de gestión de reloj, cada uno con un 'phase-locked loop' (PLL).
- 90 slice dedicados al procesamiento digital de la señal.

Además, dispone de distintos periféricos implementados en la propia Basys 3 visibles en la Figura 5-1:

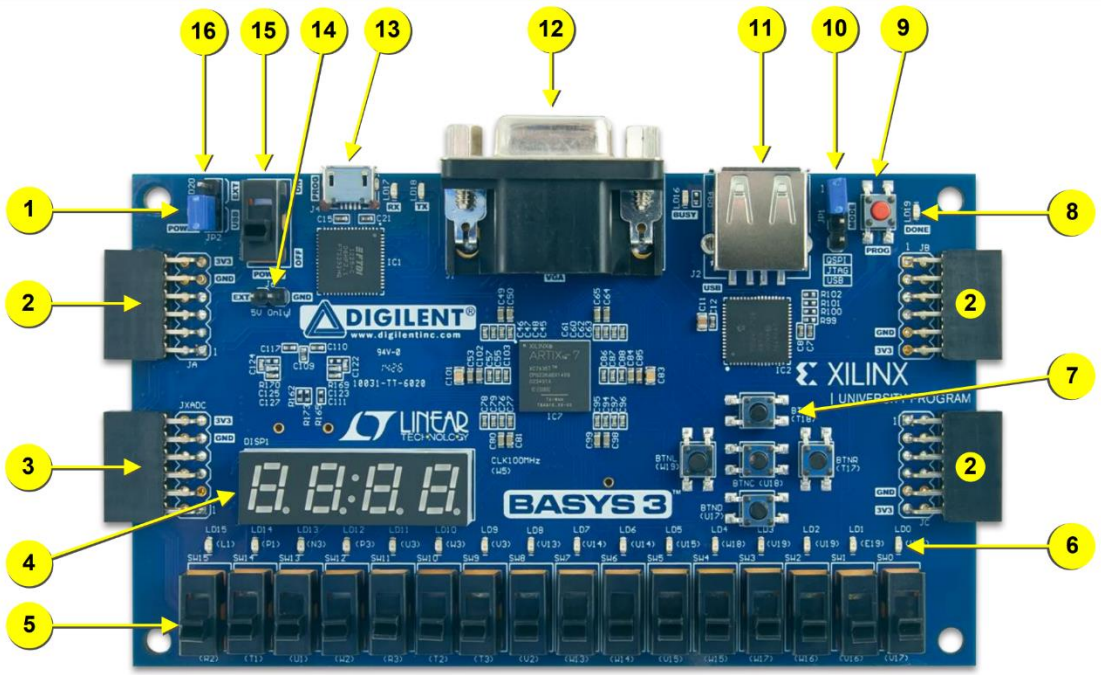

Figura 5-1. Periféricos de la placa Basys 3 (imagen extraída del manual Basys 3).

Tabla 5-1. Periféricos Basys 3

Indicador	Componente	Nº de Comp.
1	LED de alimentación conectada	1
2	Puertos Pmod	3
3	Puertos Pmod para señales analógicas (XDAC)	1
4	Display digital de 7 segmentos	4
5	Interruptores	16
6	LEDs	16
7	Botones de 'push'	5
8	LED de programación de FPGA realizada	1
9	Botón de reset de la configuración del FPGA	1
10	'Jumper' para el modo de programación	1
11	Conector host USB	1
12	Conector VGA	1
13	Puerto USB compartido UART/JTAG	1
14	Conector alimentación externa	1
15	Interruptor de encendido	1
16	'Jumper' selector de alimentación	1

Las características más relevantes para este proyecto son el tamaño de la memoria y el número de celdas lógicas de la FPGA, ya que indicarán la cantidad de información que se podrá almacenar y la capacidad de realizar operaciones lógicas y matemáticas, respectivamente.

5.2 Elementos críticos

En este apartado se analizará las características críticas que pueden existir al implementar las distintas técnicas de 'Machine-Learning' en este dispositivo

5.2.1.1 Memoria Basys 3

Esta placa contiene un dispositivo de tecnología Flash serie no-volátil de 32 Mbits que se enlaza con la FPGA Artix-7 a través de un bus SPI dedicado en modo cuádruple (x4). Este elemento puede ser necesario si se desease tener una mayor resolución de los parámetros y no existiese suficiente capacidad de memoria en la propia FPGA.

5.2.1.2 Celdas Lógicas y Memoria Artix-7 XC7A35T

La Artix-7 posee 33,280 células lógicas en 5200 slices, donde cada uno estos 4 LUTS de 6 entradas y 8 flip-flops. Además de 1,800 kbits de bloque de RAM rápida y 400 kB de RAM distribuida.

Los distintos parámetros involucrados en los cálculos de las técnicas de 'Machine-Learning' se pueden guardan en bloques de RAM (BRAM), lo cuales son partes dedicadas al almacenamiento de datos en una FPGA. Para ello será necesario instanciar 'bloques de almacenamiento ROM/RAM'.

El consumo de LUTs y flip-flops residirá en la realización de las operaciones matemáticas y lógicas necesarias para la ejecución de las técnicas de 'Machine-Learning'. Sin embargo, Vivado ofrece la posibilidad de implementar las multiplicaciones a través de los DSP, bloques que están orientados a tareas de procesamiento de la señal, de este modo, se podría ahorrar en el consumo de lógica en las slices.

5.3 Esquema general de la implementación para un predictor

La implementación de los diferentes algoritmos necesarios para la identificación de partículas se dividirá en diferentes partes, cada una asociado a un proceso en específico:

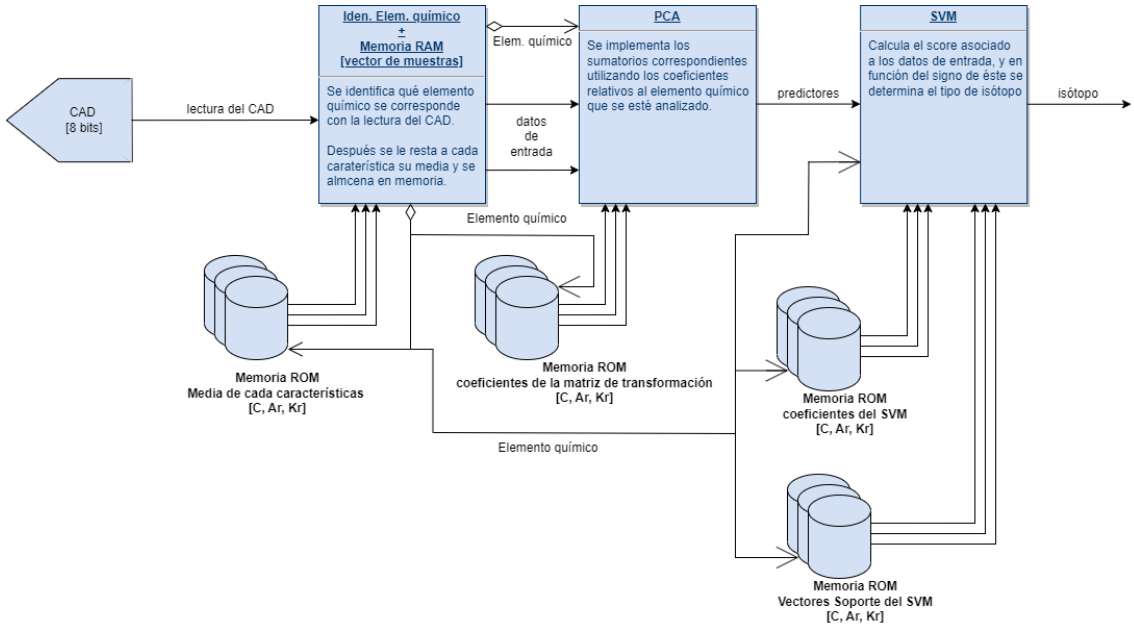

Figura 5-2. Esquema general

Los datos de entrada vienen por la lectura de un convertidor analógico digital externo a la placa Basys 3, ya que el CAD de la placa trabaja a 1 MSPS, cuando sería necesario una tasa de muestreo de 1 GHz.

5.3.1 Identificación del elemento químico y almacenamiento de las muestras

Mediante la medición del tiempo de decaimiento se identifica a qué elemento químico se corresponde los datos de entrada. Posteriormente, se le restará a cada muestra su media y se almacenarán en un bloque memoria RAM.

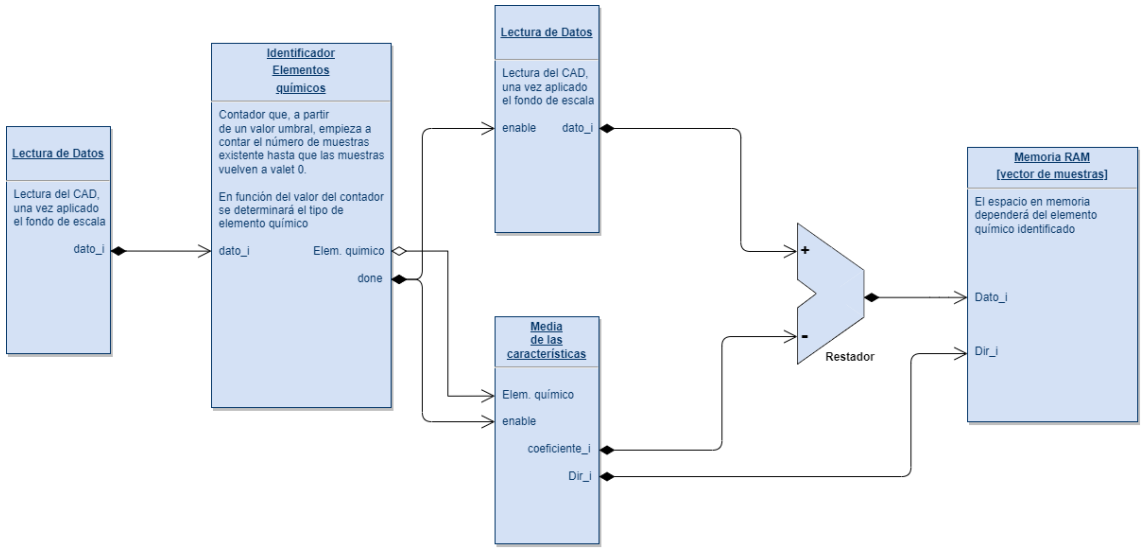

Figura 5-3. Esquema de adaptación de datos de entrada

5.3.2 PCA

Se implementarán tantos sumatorios como predictores para la correcta identificación del isótopo. Los sumatorios tendrán como entrada el producto del dato de entrada (una vez restado la media) por el coeficiente de la matriz de transformación PCA correspondiente. Aquí se aprovechará que los bloques de almacenamiento de memoria pueden tener puertos dobles, de forma que se descompone cada sumatorio en dos mitades iguales y, de este modo, trabajar el doble rápido. Y una vez que terminen ambos sumatorios, se sumarán los resultados de estos para obtener el valor del predictor. Además, será necesario un bloque que administre correctamente el direccionamiento de la memoria.

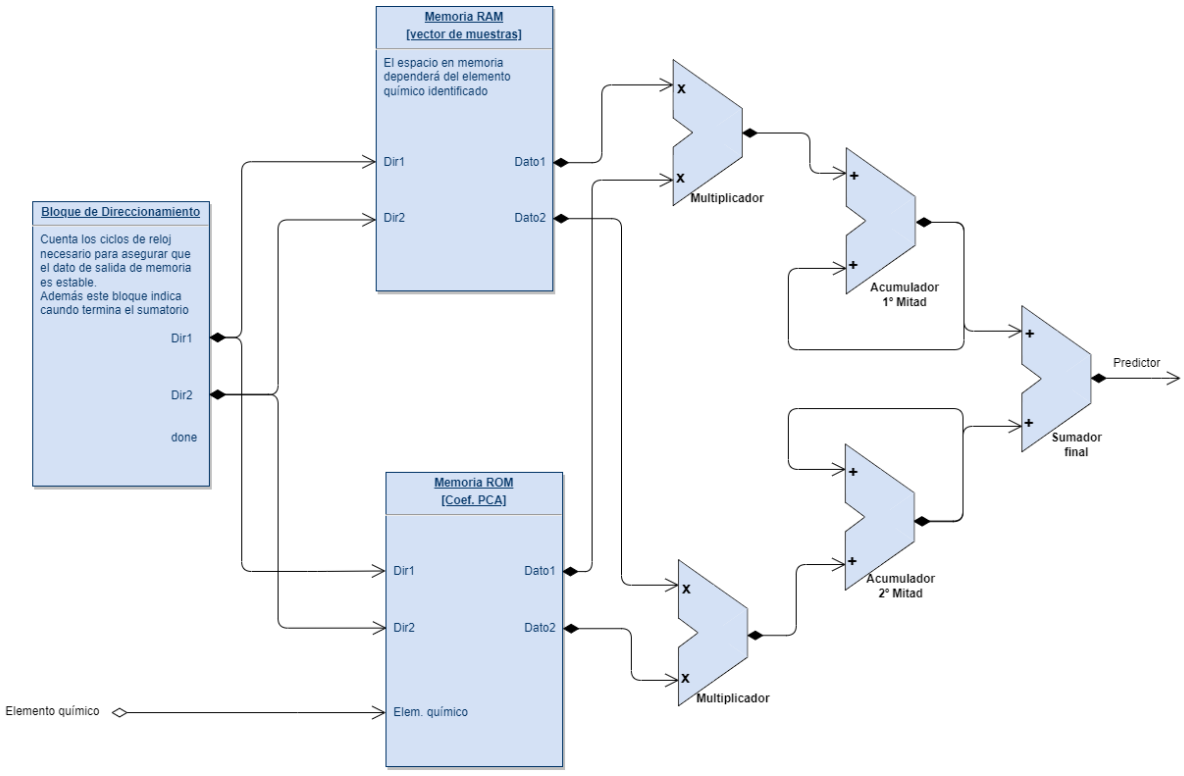

Figura 5-4. Esquema del cálculo de un predictor

5.3.3 SVM

En este bloque confluirán los 6 predictores calculados y se implementarán en un único sumatorio y en una función polinómica de orden 3 para obtener el 'score'. El hiperplano encargado de determinar el isótopo asociado a los datos de entrada es la recta score = 0, es decir, en función del signo de 'score' se conocerá el isótopo asociado a los datos de entrada. A la hora de la aplicación de la función kernel se analizará si el dato de entrada a dicho bloque es superior a un valor límite (igual a 12 como se explica en el capítulo 3), ya que si el dato de entrada supera el límite no será necesario aplicar la operación al cubo y se enviará el dato directamente al multiplicador y posteriormente al sumatorio.

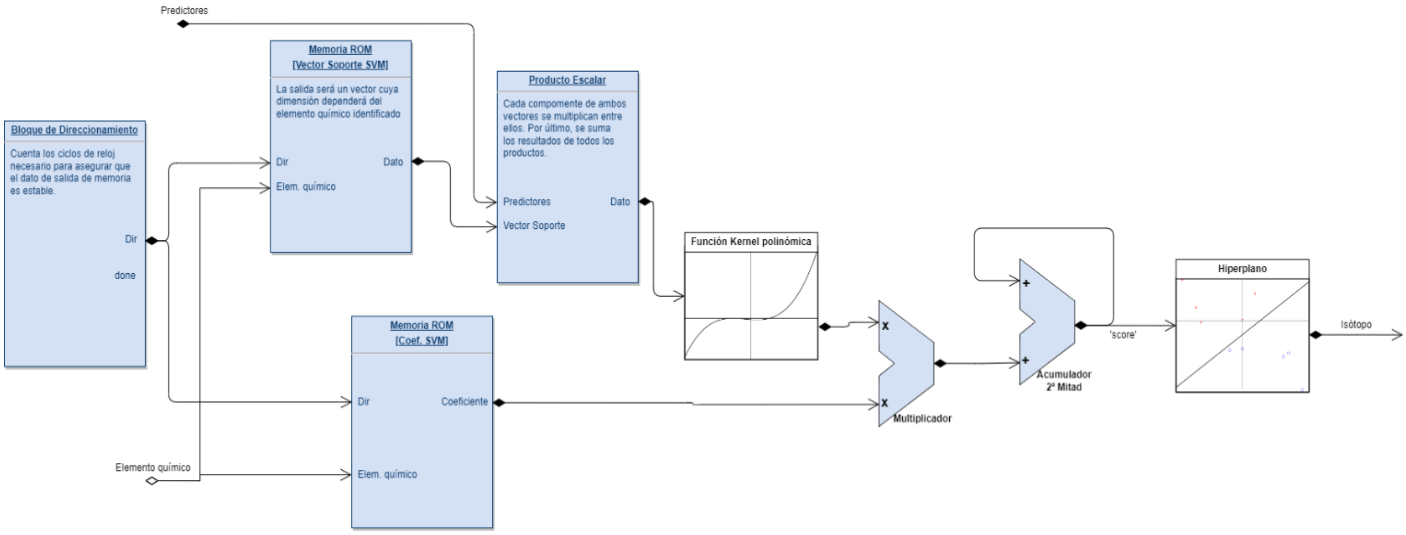

Figura 5-5. Esquema del cálculo de un predictor

5.4 Ejemplo de implementación

En este apartado, se realizará una comprobación del funcionamiento de la implementación del algoritmo del PCA en la FPGA Artix-7. El ejemplo consistirá en calcular el predictor de mayor varianza del Kriptón asociado a un vector de datos dado. Para ello, el sumatorio relativo al cálculo se realizará sin truncamiento y los coeficientes del PCA tendrán 11 bits de resolución para garantizar un cálculo preciso del predictor.

Supondremos que el elemento químico es conocido y que los datos de entrada ya han sido normalizados restando la media.

Además, previamente en el software MATLAB, se calcula el valor de dicho predictor en punto flotante con 64 bits y en punto fijo con la resolución anteriormente indicada. Dichos resultados se mostrarán en la Tabla 5-2.

Tabla 5-2. Valor del 1º Predictor

Resultado Punto Flotante	Resultado Punto Fijo
-26.7665	**-25.7632**

Se puede apreciar que el resultado en punto fijo no difiere significativamente respecto a calcularlo en punto flotante (error absoluto igual a -1.0032 y error relativo igual al 3.75 %). Además, como se observa en la gráfica

3-4 la precisión de la predicción apenas varía, disminuyendo únicamente un 0.1 %.

Mediante un test-bench, mostrado en la Figura 5-6, se simula el funcionamiento del algoritmo de la PCA y se obtiene lo siguiente:

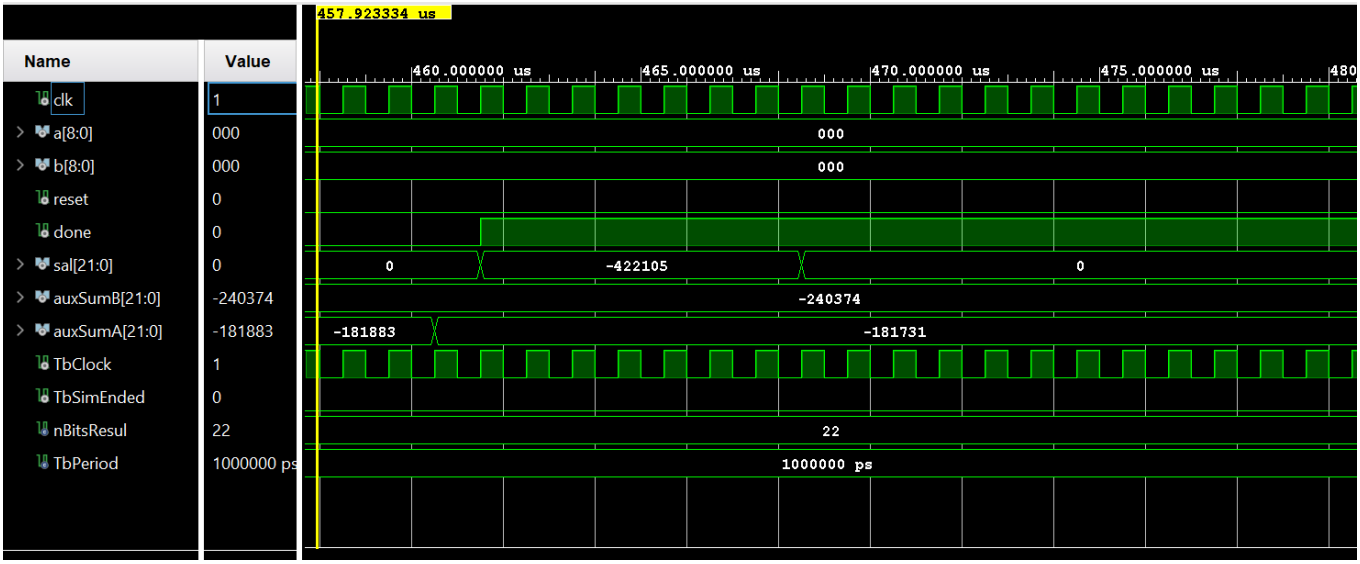

Figura 5-6. Señales resultantes del test-bench

La salida toma el valor -422105, la resolución de la parte fraccionaria de esta variable es de 14 bits de longitud, debido a que la resolución de la parte fraccionaria del dato de entrada es de 3 bits. Por tanto, dicho número dividido por 2^14 es igual a -25.7632, resultado idéntico al calculado a través de MATLAB, como se ve reflejado en la Tabla 5-2.

Seguidamente se comprobará la cantidad de recursos que demanda este ejemplo, y si estos se aproximan a los valores estimados.

Para realizar este ejemplo se ha utilizado 2 multiplicadores y 2 sumadores acumulativos, y se ha considerado que los predictores tendrán una resolución de 22 bits. Además, se ha implementado un divisor de frecuencia, de forma que habilita cada 2 ciclos de reloj los datos procedentes de los bloques de memoria. Por tanto, se estima, basándose en los datos del anterior capítulo, que el número de LUTs será de 582 y 178 flip-flops.

A continuación, se mostrará los recursos implementados a través de la Tabla 5-3, y mediante la Tabla 5-4 se comparará los recursos que se han implementado respecto a los estimados.

Tabla 5-3. Recursos del ejemplo implementado

Resource	Estimation	Available	Utilization %
LUT	166	20800	0.80
FF	144	41600	0.35
DSP	2	90	2.22
IO	25	106	23.58
BUFG	1	32	3.13

Tabla 5-4. Comparativa de recursos estimados frente a implementados en el ejemplo

Recursos	Estimados	Implementados
LUTs	582	166
FFs	178	144

Se observa que los recursos necesarios para el ejemplo son inferiores a los estimados en una primera aproximación. Esto es esperado, ya que la estimación del consumo de recursos de los operadores se sobredimensionó, con el fin de tomar el máximo consumo posible. Además, gracias a la posibilidad de implementar los multiplicadores mediante los bloques DSP se ha podido reducir significativamente los recursos empleados en dichos operadores.

Es necesario, también, considerar que los recursos implementados engloban tanto los necesarios para aplicar las operaciones matemáticas como las operaciones lógicas que concierne a la gestión de los bloques de almacenamiento de memoria. Por tanto, se puede tomar como patrón los recursos implementados en este ejemplo para estimar la implementación global del PCA.

De este modo, los recursos necesarios para implementar del PCA, en su conjunto, tendrán aproximadamente los siguientes gastos de recursos: 996 LUTs, 864 FFs y 12 bloques DSP. Esto supone un 30 % del consumo de LUTs y un 80 % de flip-flops respecto a los valores estimados inicialmente.

6 CONCLUSIONES Y LÍNEAS FUTURAS

En este último capítulo se comentarán distintas conclusiones y líneas futuras del proyecto.

6.1 Comparación con la literatura existente

Actualmente, la implementación de la PSA mediante técnicas de 'Machine-Learning' está en auge, principalmente relacionadas con implementaciones con redes neuronales artificiales (ANN). Esto se debe a que estas técnicas son capaces de conseguir una alta precisión en la identificación de partículas (por encima del 99%). Pero al igual que ocurre en este proyecto, el problema más relevante es la relación de compromiso entre la memoria y recursos que demanda este tipo de técnicas y la posibilidad de aplicarlas de forma paralela, ya que se ha observado que el empleo de varias neuronas por capa (mayor paralelismo) implica en un aumento significativo del coste computacional [13].

La opción que ofrece este proyecto es la capacidad de implementar técnicas de 'Machine-Learning' ampliamente estudiadas como una posibilidad viable para la implementación de la PSA con un menor consumo de recursos.

6.2 Limitaciones del estudio

Este proyecto se ha realizado, en su conjunto, mediante simulaciones en MATLAB, por tanto, los resultados en una implementación real en una FPGA pueden diferir, ya que existen distintas condiciones que no se han considerado, como datos de entrada con ruido debido al ruido de fondo o causado por alguna otra naturaleza.

Además, los recursos considerados son estimaciones, los cuales también pueden diferir de los recursos utilizados en una implementación real, como se ha observado a través del ejemplo del PCA. Sin embargo, las estimaciones se han contemplado suponiendo el peor caso donde el consumo de recursos es máximo, de esta forma, la implementación final tenderá a tener una demanda menor de recursos.

Por otro lado, se ha considerado que los datos que se procesen en el futuro por la FPGA tendrán un comportamiento similar a los datos analizados en la pre – implementación. Por tanto, frente al análisis de otro tipo de partículas o empleo de una tasa muestreo diferente, las técnicas utilizadas en este proyecto no servirían.

6.3 Resumen de los hallazgos

Este proyecto tenía como fin la implementación de técnicas de 'Machine-Learning' en una FPGA para la identificación de partículas a través del análisis de la forma de pulso de la manera más eficiente. Dicho conseguido se alcanzado, consiguiendo una precisión por encima del 95 % minimizando recursos.

Sin embargo, existe el inconveniente de que no se aprovecha el completamente del paralelismo que ofrece la FPGA, ya que, por ejemplo, las 1800 sumas y multiplicaciones del PCA se podría realizar en paralelo. Sin embargo, esto requeriría de tener las 300 muestras a la vez (la FPGA solo dispone de 250 I/O), además de un alto computacional debido a que sería necesario 1800 sumadores.

Una posibilidad de reducir coste computacional es almacenar en la memoria Flash de la placa Basys 3 los parámetros que no varían entre distintos experimentos como son los coeficientes del PCA, del SVM y los

vectores soporte. Pero, el empleo de la memoria Flash requiere trabajar con la comunicación SPI, lo cual relentecería el proceso de identificación.

Además, se podría plantear las mismas técnicas de 'Machine- Learning' con un menor número de parámetros, ya que, por ejemplo, un hiperplano conformado por 69 vectores soporte es una cantidad excesiva y que se podría reducir si se optimizase, en específico, los procesos de entrenamiento y validación del SVM.

6.4 Recomendaciones para futuras investigaciones

Para líneas futuras se podría realizar la implementación dentro de un sistema embebido SoC conformado por un microcontrolador y una FPGA. De forma que, la implementación elaborada en este proyecto estuviera integrada dentro una IP de la FPGA que estaría únicamente dedicada a la identificación de partículas.

El microcontrolador le enviaría los datos de la lectura de una partícula y la FPGA, a modo de periférico, sería capaz de identificar la partícula asociada a dichos datos de forma rápida y fiable.

Además, se puede plantear la integración de otras técnicas de 'Machine-Learning' como son los algoritmos basados en 'Tree Decision'. Estos clasifican los datos de entrada mediante respuestas de verdadero o falso a una serie de preguntas cuya disposición se asemeja a la de un árbol. Estas preguntas están organizadas de forma que, intentan discriminar los datos en el menor número de pasos posibles. Para ello, dichas preguntas se construyen a partir de datos estadísticos [14].

Como ejemplo de 'Tree Decision', se muestra en la Figura 6-1 una cadena de sentencias relativas a una característica estadísticamente significativa de la identificación de partículas, el tiempo de decaimiento. Mediante la negación o afirmación de las distintas comparaciones se puede identificar, con relativa rapidez, el elemento químico.

A través de los datos de entrenamiento, se observa que aquellos experimentos que duran menos de 120 ns están asociados a la medición de Carbono. Mientras que, si dicho tiempo es mayor, en función de si supera 220 ns, puede tratarse de Argón, si no lo supera, o Kriptón, en el caso de que sí lo supere. Por tanto, con el uso dos sentencias se puede clasificar el elemento químico, de modo que se compruebe si una determinada muestra es distinta de cero.

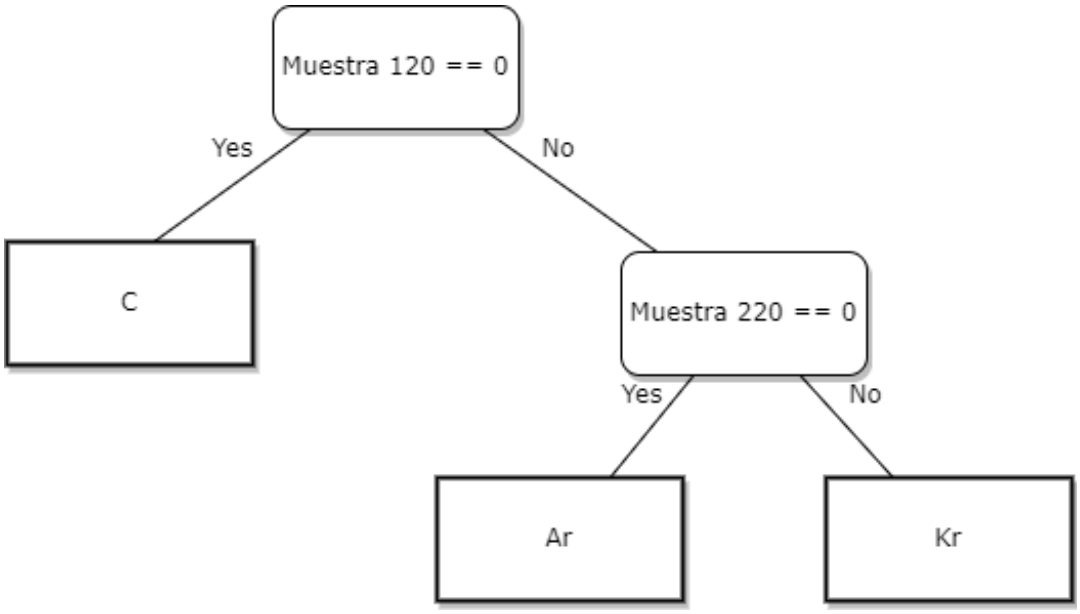

Figura 6-1. Ejemplo de Árbol de decisión

REFERENCIAS

[1] J.L. Flores, I. Martel, R. Jiménez, J. Galán, P. Salmerón (2016) Application of neural networks to digital pulse shape analysis for an array of silicon strip detectors. *ELSEVIER*. *830*, 287-293.

https://www.sciencedirect.com/science/article/pii/S0168900216305095?via%3Dihub

[2] Wikipedia. *Aprendizaje Automático*. Wikipedia. Online:

https://es.wikipedia.org/wiki/Aprendizaje_supervisado#cite_note-Danziger2006-1

[3] Wikipedia. *Física de partículas*. Wikipedia. Online:

https://es.wikipedia.org/wiki/F%C3%ADsica_de_part%C3%ADculas

[4] Wikipedia. *Partícula elemental*. Wikipedia. Online:

https://es.wikipedia.org/wiki/Part%C3%ADcula_elemental

[5] Wikipedia, *Análisis de componentes principales*. Wikipedia. Online:

https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales

[6] Wikipedia, *Máquinas de vectores de soporte*. Wikipedia. Online:

https://es.wikipedia.org/wiki/M%C3%A1quinas_de_vectores_de_soporte

[7] MathWorks. *Understanding SVM.* MATLAB. Online:

https://es.mathworks.com/discovery/support-vector-machine.html#:~:text=Support%20vector%20machine%20(SVM)%20es,reconocimiento%20de%20im%C3%A1genes%20y%20voz

[8] S. Barlini, R. Bougault, Ph. Laborie, O. Lopez, D. Mercier, M. Parlog, B. Tamain, E. Vient, E. Chevallier, A. Chbihi, B. Jacquot, V. L. Kravchuk, (2009) New digital techniques applied to A and Z identification using pulse shape discrimination of silicion detector currente signals. *ELSEVIER*. *600*. 644-650.

https://www.sciencedirect.com/science/article/abs/pii/S0168900209000023

[9] XILINX, *7 Series FPGAs Data Sheet: Overview.* XILINX. Online:

https://datasheet.lcsc.com/lcsc/2003091237_XILINX-XC7A200T-2FBG484I_C494680.pdf

[10] DIGILENT, *Basys 3™ FPGA Board Reference Manual* DIGILENT. Online:

https://digilent.com/reference/_media/basys3:basys3_rm.pdf

[11] Montes Salinero, J. J (2019) *Simulación y medida de consumo en FPGAs para arquitecturas de operadores aritméticos* [Trabajo de Fin de Grado, Universidad Politécnica de Madrid]

https://oa.upm.es/54373/1/TFG_JUAN_JOSE_MONTES_SALINERO.pdf

[12] Allen Institute. *Implementación multiplicador Baugh-Wooley*. Semantic Scholar. Online:

https://www.semanticscholar.org/paper/Design-of-Baugh-wooley-Multiplier-using-Verilog-HDL-Kale-Zade/ec93ef3854154d493dce86b2daf814dd3f30dfd2/figure/0

[13] R. Jiménez, M. Sánchez-Raya. J. A. Gómez-Galán, J.L Flores, J.A Dueñas, I. Martel (2012) Implementation of a neural network for digital pulse shape analysis on a FPGA for on-line identification of heavy ions. *ELSEVIER*. *674*. 99-104. https://www.sciencedirect.com/science/article/abs/pii/S0168900212000745?via%3Dihub

[14] Wikipedia. *Decision tree learning*. Wikipedia. Online:

https://en.wikipedia.org/wiki/Decision_tree_learning#:~:text=Decision%20trees%20are%20among%20the,represent%20decisions%20and%20decision%20making

[15] Baena Alonso, A (2010) *Diseño sobre FPGA de una Unidad Aritmética Decimal* [Trabajo de FIn de Grado, Universidad Robira I Virgili]

http://deeea.urv.cat/public/PROPOSTES/pub/pdf/1453pub.pdf

[16] ResearchGate. *Schematic overview of a beam line*. ResearchGate. Online:

https://www.researchgate.net/figure/Schematic-overview-of-a-beam-line-design-of-a-neutron-imaging-facility-with-the-main_fig1_230998716

[17] Tecnológico de Puerto Rico. *El átomo*. Tecnológico de Puerto Rico. Online:

https://www.tec.ac.cr/atomo

[18] R. A. Winyard, J. E. Lutkin, and G. W. McBeth. (1971) Pulse shape discrimination in inorganic and organic scintillators. I: Nucl. Instruments Methods. *ELSEVIER*, 95, 1, 141–153. doi: 10.1016/0029-554X(71)90054-1.

ANEXO

A.1 Código (VHDL)

Divider.vhd [15]

```
--Entidad del divisor de n dígitos decimales

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL; USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY decimal_divider IS GENERIC (n,m,p,logp:natural:= 16);
PORT(
x:IN STD_LOGIC_VECTOR(4*n-1 downto 0); --dividendo
y:IN STD_LOGIC_VECTOR(4*n-1 downto 0); --divisor

--divisor
start:      IN    STD_LOGIC;              --señal de inicio
clk:  IN    STD_LOGIC;              --señal de reloj
reset:      IN    STD_LOGIC;              --inicializa operación
q:    OUT   STD_LOGIC_VECTOR(4*m-1  DOWNTO      0);   --resultado (cociente)
done: OUT   STD_LOGIC);             --operación realizada
END decimal_divider;


--Arquitectura del divisor de n dígitos decimales
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL; USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ARCHITECTURE architecture_divider OF decimal_divider IS

--Declaración de estados
TYPE states IS RANGE 0 TO 3;
SIGNAL current_state: states;

--Declaración de constantes
CONSTANT initial_zeroes: STD_LOGIC_VECTOR(4*p-9 DOWNTO 0) := ( OTHERS =>
'0');
CONSTANT initial_zeroes_2: STD_LOGIC_VECTOR(4*p-5 DOWNTO 0) := ( OTHERS =>
'0');

--Declaración de señales
SIGNAL r, s1: STD_LOGIC_VECTOR(4*n-1 DOWNTO 0);
SIGNAL rr, rr_y, yy: STD_LOGIC_VECTOR(4*n+3 DOWNTO 0);
SIGNAL qq, ulp, qq_ulp, s2: STD_LOGIC_VECTOR(4*p-1 DOWNTO 0); SIGNAL c_out:
STD_LOGIC;
SIGNAL load, ce, zero: STD_LOGIC;
SIGNAL ulp_by_5: STD_LOGIC_VECTOR(4*p+3 DOWNTO 0); SIGNAL count:
STD_LOGIC_VECTOR(logp-1 DOWNTO 0);
```

```
--Declaración de componentes
COMPONENT n_by_one_multiplier IS    --multiplicador de Nx1 dígitos BCD
GENERIC (n:natural);
PORT(
x: IN STD_LOGIC_VECTOR(4*n-1 DOWNTO 0); --entrada de n dígitos BCD
y: IN STD_LOGIC_VECTOR(3 DOWNTO 0); --entrada de 1 dígito BCD
z: OUT STD_LOGIC_VECTOR(4*n+3 DOWNTO 0)); --salida de n dígitos BCD
END COMPONENT;

COMPONENT n_adder_subs IS     --Sumador/restador utilizado para sumar
GENERIC (n:natural);
PORT(
x:    IN STD_LOGIC_VECTOR(4*n-1 DOWNTO 0); --entrada de n dígitos BCD
y:    IN STD_LOGIC_VECTOR(4*n-1 DOWNTO 0); --entrada de n dígitos BCD
add_sub: IN STD_LOGIC;  --operación: '0' suma y '1' resta
z:    OUT STD_LOGIC_VECTOR(4*n-1 DOWNTO 0); --salida de n dígitos BCD
carry_out: OUT STD_LOGIC);          --acarreo de salida
END COMPONENT;

--Inicio
 BEGIN

yy <= "0000" & y; --Añadimos un 0 decimal a 'y' para igualarla
--en número de dígitos a rr (2r)

multiplier_r_by_2: n_by_one_multiplier    --Multiplicador de r x 2
GENERIC MAP(n => n)
PORT MAP(
x => r,
y => "0010",      --2 en decimal
z => rr);   --resultado 2r
subtraction: n_adder_subs     --Restador para realizar 2r - y
GENERIC MAP(n => n+1)
PORT MAP(
x => rr,    --2r
y => yy,
add_sub => '1',   --Resta como operación a realizar
z => rr_y,  --resultado de 2r-y
carry_out => c_out);

WITH NOT(c_out) SELECT s1 <= rr(4*n-1 DOWNTO 0) WHEN '0', rr_y(4*n-1 DOWNTO
0) WHEN OTHERS;

addition: n_adder_subs  --Sumador para realizar qq + ulp
GENERIC MAP(n => p)
PORT MAP(
x => qq, y => ulp,
add_sub => '0',   --Suma como operación a realizar
z => qq_ulp);

WITH NOT(c_out) SELECT s2 <= qq WHEN '0', qq_ulp WHEN OTHERS;

multiplier_ulp_by_5: n_by_one_multiplier --Multiplicador de ulp x 5
GENERIC MAP(n => p)
PORT MAP(
x => ulp,
y => "0101",      --5 en BCD
z => ulp_by_5);   --resultado 5*ulp

q     <= qq(4*p-1 DOWNTO 4*(p-m));  --Los m dígitos más significativos
--de qq
--Sentencias secuenciales register_r:
```

```
PROCESS(clk) BEGIN
IF clk'EVENT AND clk = '1' THEN
IF load = '1' THEN r <= x;    --El valor inicial de r es x
ELSIF ce = '1' THEN r <= s1;  --Actualiza el valor de r
END IF;
END IF;
END PROCESS;

register_qq: PROCESS(clk) BEGIN
IF clk'EVENT AND clk = '1' THEN
IF load = '1' THEN qq <= (OTHERS => '0');--Valor inicial de qq es 0
ELSIF ce = '1' THEN qq <= s2; --Actualiza el valor de qq
END IF;
END IF; END PROCESS;

register_ulp: PROCESS(clk) BEGIN
IF clk'EVENT AND clk = '1' THEN
IF load = '1' THEN ulp <= "0101" & initial_zeroes_2;
--El valor inicial es 5, que representa 0.5
ELSIF ce = '1' THEN ulp <= ulp_by_5(4*p+3 DOWNTO 4);--Actualiza ulp
END IF;
END IF;
END PROCESS;
--Contador de p estados counter_stages:
PROCESS(clk)
BEGIN
IF clk'EVENT AND clk = '1' THEN
IF load = '1' THEN count <= CONV_STD_LOGIC_VECTOR(p-1, logp); ELSIF ce = '1'
THEN count <= count - 1;
END IF;
END IF;
END PROCESS;

zero <= '1' WHEN count = "0" ELSE '0';

--Unidad de control
control_unit: PROCESS(clk, reset, current_state, zero) BEGIN
CASE current_state IS
WHEN 0 to 1 => load <= '0'; ce <= '0'; done <= '1';
WHEN 2 => load <= '1'; ce <= '0'; done <= '0';
WHEN 3 => load <= '0'; ce <= '1'; done <= '0'; END CASE;
IF reset = '1' THEN current_state <= 0; ELSIF clk'EVENT AND clk = '1' THEN
CASE current_state IS
WHEN 0 => IF start = '0' THEN current_state <= 1; END IF; WHEN 1 => IF start
= '1' THEN current_state <= 2; END IF; WHEN 2 => current_state <= 3;
WHEN 3 => IF zero = '1' THEN current_state <= 0; END IF; END CASE;
END IF;
END PROCESS;

END architecture_divider;
```

Prueba_mul.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
 use IEEE.std_logic_signed.all;
--use IEEE.std_logic_unsigned.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

-- Este ejemplo es una prueba de la implemetación de la técnica de 'Machine-
Learning' PCA

entity prueba_mul is
    generic(    nBits : integer := 8+1; -- +1 por el bit de signo
    );
    Port ( clk: in std_logic;
           a : in STD_LOGIC_VECTOR (nBits-1 downto 0); -- Dato proveniente de
la 1° mitad del vector
           b : in STD_LOGIC_VECTOR (nBits-1 downto 0); -- Dato proveniente de
la 2° mitad del vector
           reset: in std_logic;
           done: out std_logic; -- SeÃ±al que indica si la operación ha
terminado
           sal: out STD_LOGIC_VECTOR (21 downto 0) -- salida

       );
end prueba_mul;


architecture Behavioral of prueba_mul is
-- Componentes
component blk_mem_gen_0 IS -- vector de muestras
  PORT (
   clka : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
    dina : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
    clkb : IN STD_LOGIC;
    enb : IN STD_LOGIC;
    web : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addrb : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
    dinb : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
    doutb : OUT STD_LOGIC_VECTOR(8 DOWNTO 0)
  );
END component;


component cambioDigito is -- Cuando se lea el CAD hay guardarlo en una
variable RAM
```

```
    generic(    nBits : integer := 8
    );
    Port ( clk: in std_logic;
           a : in STD_LOGIC_VECTOR (nBits-1 downto 0); -- Dato proveniente de
la 1° mitad del vector
           b : in STD_LOGIC_VECTOR (nBits-1 downto 0); -- Dato proveniente de
la 2° mitad del vector
           sel:out STD_LOGIC_VECTOR (8 downto 0); -- ¨ dato a (vector de 300
muestras)
           sel2:out STD_LOGIC_VECTOR (8 downto 0); --  del dato b
           reset: in std_logic -- señal de sincronismo que indica que
pertencen al primer índice
        );
end component;


component PCA_1_datos IS
  PORT (
    clka : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(12 DOWNTO 0);
    clkb : IN STD_LOGIC;
    addrb : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
    doutb : OUT STD_LOGIC_VECTOR(12 DOWNTO 0)
  );
END component;



component AvanzarDigito is
  generic(    Length : integer := 9
    );
  Port ( clk: in std_logic;
         reset : in std_logic;
         sal: out std_logic_vector(Length-1 downto 0)
  );
end component;

-- Señales

signal auxSumA,auxSumB,p_auxSumA,p_auxSumB: signed (21 downto 0) :=
(others=>'0'); -- Debe haber estas mismas variables para componente del
vector de salida (es decir 5 más)
signal auxSumA_ant,auxSumB_ant,p_auxSumA_ant,p_auxSumB_ant: signed (21 downto
0) := (others=>'0'); -- Debe haber estas mismas variables para componente del
vector de salida (es decir 5 más)
signal c_1_A,c_1_B,p_c_1_A,p_c_1_B: std_logic_vector (20-1 downto 0):=
(others=>'0');
signal p_sal1: std_logic_vector(21 downto 0) := (others=>'0');
signal auxA,auxB: STD_LOGIC_VECTOR (nBits-1 downto 0) := (others=>'0'); --
Debe haber estas mismas variables para componente del vector de salida (es
decir 5 más)
signal aux1,aux2: STD_LOGIC_VECTOR (nBits-1 downto 0) := (others=>'0'); --
Debe haber estas mismas variables para componente del vector de salida (es
decir 5 más)
signal auxPCA_1_A,auxPCA_1_B: STD_LOGIC_VECTOR (12 downto 0) :=
(others=>'0'); -- Debe haber estas mismas variables para componente del
vector de salida (es decir 5 más)
-- Estas variables se conocen sus tamaños porque la IA ya está entrada y ya
está definida la precisión
signal mulP_1_A,mulP_1_B,p_mulP_1_A,p_mulP_1_B: STD_LOGIC_VECTOR (21 downto
0); -- Producto de muestra por coeficiente de la PCA
```

```
signal sel,sel2,sel2_aux: STD_LOGIC_VECTOR (8 downto 0); -- Indice en la
lectura de los datos del CAD
signal sel_RAM,sel2_RAM: STD_LOGIC_VECTOR (8 downto 0); -- Indice en la
lectura de los datos de RAM
signal pr_sel2: unsigned (8 downto 0):= (others=> '0'); -- valor que se
espera del proximo índice sobre la 2° mitad de la matriz
signal sels2: unsigned (8 downto 0):= (others=> '0');
signal pr_sel: unsigned (8 downto 0):= (others=> '0'); -- valor que se espera
del proximo índice sobre la 1° mitad de la matriz
signal sels: unsigned (8 downto 0):= (others=> '0');
signal p_cont: unsigned (2 downto 0):= (others=> '0');
signal cont: unsigned (2 downto 0):= (others=> '0'); -- contador que indica
qué componente del vector resultante de la operación sale por 'sal'
signal change1:std_logic := '0'; -- Variable que indica si hay que aplicar la
suma parcial sobre la 1° mitad
signal change2:std_logic := '0'; -- Variable que indica si hay que aplicar la
suma parcial sobre la 2° mitad
signal dones,p_done: std_logic := '0';
begin
    sel<=sel_RAM;
    sel2<=sel_RAM + "010010110"; -- seleción de la 2 mitad de los datos

   u1: blk_mem_gen_0 port map( -- vector de muestras
    clka    => clk,
    wea     => (others=>'0'),
    addra   => sel,
    dina    => (others=>'0'),
    douta   => auxA,
    clkb    => clk,
    enb     => '1',
    web     => (others=>'0'),
    addrb   => sel2,
    dinb    => (others=>'0'),
    doutb   => auxB
    );


   u3: PCA_1_datos port map(
    clka   =>clk,
    addra  =>sel,
    douta  =>auxPCA_1_A,
    clkb   =>clk,
    addrb  =>sel2,
    doutb  =>auxPCA_1_B
   );

   u4: AvanzarDigito generic map( 9)
   Port map (
    clk    => clk,
    reset  => reset,
    sal    => sel_RAM
   );

   -- Lógica
    mulParcail: process(reset,auxA,auxB,auxPCA_1_A,auxPCA_1_B,c_1_A,c_1_B)
    begin
         if (reset = '1') then
             p_mulP_1_A <=(others=>'0');
             p_mulP_1_B <=(others=>'0');

         else
             p_mulP_1_A <= auxA*auxPCA_1_A;
             p_mulP_1_B <= auxB*auxPCA_1_B ;
```

```
        end if;
    end process;

    PCA:
process(reset,auxSumA,auxSumB,auxA,auxB,auxPCA_1_A,auxPCA_1_B,auxSumA_ant,aux
SumB_ant,mulP_1_A,mulP_1_B)
    begin
        if (reset = '1') then
          p_auxSumA <= (others=>'0');
          p_auxSumB <= (others=>'0');
          p_auxSumA_ant <= (others=>'0');
          p_auxSumB_ant <= (others=>'0');
        else
          p_auxSumA <= auxSumA + signed(mulP_1_A);--(16 downto (16-(11))));-
- - signed(c_1_A(12 downto 0));
          p_auxSumB <= auxSumB + signed(mulP_1_B);--signed(mulP_1_B(16
downto (16-(11)) ));-- - signed(c_1_B(12 downto 0));

        end if;

    end process;

    comb: process(sels2,sel2,reset,auxSumA,auxSumB,cont,sels,sel)
    begin
        if (reset = '1') then -- reseteo de las señales
            p_sal1 <= (others=>'0');
            p_cont<= (others=>'0');
            pr_sel<= (others=>'0');
            pr_sel2<=  "010010110"; -- indice 150
            p_done<='0';
            change1<='0';
            change2<='0';
        else
            if(sels2-150 >= 151 AND sels >= 151)then
                p_done<='1'; -- La operación ya ha terminado
                change1<='0';
                change2<='0';
                if(cont<7)then
                    p_sal1 <= std_logic_vector((auxSumA+auxSumB));-- se envia
la componente "cont"
                    p_cont<=cont+1;
                    pr_sel<=sels;
                    pr_sel2<=sels2;
                 else -- Después de enviar todas las componentes del
resultado la salida toma el valor 0
                    p_sal1 <= (others=>'0');
                    p_cont<=cont;
                    pr_sel<=sels;
                    pr_sel2<=sels2;
                 end if;
            else
                p_sal1 <= (others=>'0');
                p_cont<=cont;
                p_done<='0';
                if(std_logic_vector(sels)=sel)then     -- si 'sel' coincide
con el índice esperado (='sels'), entonces se cambia el índice esperado y que
aplica la suma parcial
                    pr_sel<=sels+1;
                    change1<='1';
                else
                    pr_sel<=sels;
                    change1<='0';
```

```
                end if;
                if(std_logic_vector(sels2)=sel2)then -- si 'sel2' coincide
con el índice esperado (='sel2s')
                    pr_sel2<=sels2+1;
                    change2<='1';
                else
                    pr_sel2<=sels2;
                    change2<='0';
                end if;
            end if;
        end if;
    end process;

   sync: process(clk,reset)
    begin
        if (reset = '1') then -- reseteo de las señales
            sal<=(others=>'0');
            auxSumA<=(others =>'0');
            auxSumB<=(others =>'0');
            cont<=(others=>'0');
            sels<=(others=>'0');
            sels2<= "010010110"; -- indice 150
            done<='0';
            auxSumA_ant <= (others=>'0');
            auxSumB_ant <= (others=>'0');
            mulP_1_A <= (others=>'0');
            mulP_1_B <= (others=>'0');
        elsif(rising_edge(clk)) then
               sal<=p_sal1;
               cont<=p_cont;
               sels<=pr_sel;
               sels2<=pr_sel2;
               done<=p_done;
               mulP_1_A <= p_mulP_1_A;
               mulP_1_B <= p_mulP_1_B;
                    if(change1='1')then -- se aplica la suma parcial cuando
se avanza de índice de fila
                        auxSumA <= p_auxSumA;
                        auxSumA_ant <= p_auxSumA_ant;
                    end if;
                    if(change2='1')then
                        auxSumB<=p_auxSumB;
                        auxSumB_ant <= p_auxSumB_ant;
                    end if;
        end if;
    end process;
end Behavioral;
```

A.2 Código (MATLAB)

simularScore.m

```
longitud = 2000; % nº de muestras para el análisis
indice=linspace(1,2199,longitud);
indice=fix(indice);
% Precisión del resultado
    bitsFracPCA=7; % Resolución de los datos de la PCA (parte entera tiene 1 bit)
    bitsSumTt=9; % Bits de la fración del sumatorio (parte entera tiene 8 bits)

% Predictores
Predictores_Double = zeros(longitud,6);
Predictores_PuntoFijo = zeros(longitud,6);
Predictores_puntoFijo_Truncamiento = zeros(longitud,6);

% Predicción
pred_Double = cell(longitud,1);
pred_PuntoFijo = cell(longitud,1);
pred_PuntoFijo_Truncamiento =  cell(longitud,1);
score_Double = zeros(longitud,2);
score_PuntoFijo = zeros(longitud,2);
score_puntoFijo_Truncamiento = zeros(longitud,2);

for jj=1:longitud

% Selección de Datos
   ii=indice(jj);
   datos=T.feat(ii,:)-mediaDatos; % Datos normalizados
   resultado=T.label(ii);

   % Pasar a binario los datos
   ensayo1=datos.*PCA_Kr(:,1)';
   [~,bin1]=datos2bin(datos,1,4,3); % pasa a binario únicamente "datos" % (8 bits del CAD [5|3]; 5 la parte entera debido a que el mayor número posible puede llegar a necesitar 5 bits)

   % Pasar a binario las PCA
   [~,binPCA_1]=datos2bin(PCA_Kr(:,1),1,2,bitsFracPCA); % pasa a binario únicamente "PCA_Kr(:,i)". datos2bin da problemas si la parte entera es inferior a 2
   binPCA_1=adaptarPCA(binPCA_1);
```

```
[~,binPCA_2]=datos2bin(PCA_Kr(:,2),1,2,bitsFracPCA); % pasa a binario únicamente "PCA_Kr(:,i)". datos2bin da problemas si la parte entera es inferior a 2
binPCA_2=adaptarPCA(binPCA_2);

[~,binPCA_3]=datos2bin(PCA_Kr(:,3),1,2,bitsFracPCA); % pasa a binario únicamente "PCA_Kr(:,i)". datos2bin da problemas si la parte entera es inferior a 2
binPCA_3=adaptarPCA(binPCA_3);

[~,binPCA_4]=datos2bin(PCA_Kr(:,4),1,2,bitsFracPCA); % pasa a binario únicamente "PCA_Kr(:,i)". datos2bin da problemas si la parte entera es inferior a 2
binPCA_4=adaptarPCA(binPCA_4);

[~,binPCA_5]=datos2bin(PCA_Kr(:,5),1,2,bitsFracPCA); % pasa a binario únicamente "PCA_Kr(:,i)". datos2bin da problemas si la parte entera es inferior a 2
binPCA_5=adaptarPCA(binPCA_5);

[~,binPCA_6]=datos2bin(PCA_Kr(:,6),1,2,bitsFracPCA); % pasa a binario únicamente "PCA_Kr(:,i)". datos2bin da problemas si la parte entera es inferior a 2
binPCA_6=adaptarPCA(binPCA_6);

% Cálculo del producto muestra por coeficiente en Punto Fijo
reg_Mul=zeros(6,length(datos));
for ind = 1:length(datos)
    reg_Mul(1,ind)=multiplicarBinariosConSigno(bin1(ind),binPCA_1(ind),3,bitsFracPCA);
    reg_Mul(2,ind)=multiplicarBinariosConSigno(bin1(ind),binPCA_2(ind),3,bitsFracPCA);
    reg_Mul(3,ind)=multiplicarBinariosConSigno(bin1(ind),binPCA_3(ind),3,bitsFracPCA);
    reg_Mul(4,ind)=multiplicarBinariosConSigno(bin1(ind),binPCA_4(ind),3,bitsFracPCA);
    reg_Mul(5,ind)=multiplicarBinariosConSigno(bin1(ind),binPCA_5(ind),3,bitsFracPCA);
    reg_Mul(6,ind)=multiplicarBinariosConSigno(bin1(ind),binPCA_6(ind),3,bitsFracPCA);
end
[ensayo1' reg_Mul'];
[sum(reg_Mul,2) (datos*PCA_Kr)']; % [Sumatorio sin Truncamiento Sumatorio en punto Flotante]

datosPCA_puntoFijo=sum(reg_Mul,2);

[~,elem1]=datos2bin(reg_Mul(1,:),1,3,(bitsFracPCA+3)); % Más 3 debido a que el desplazamiento en bits del punto ha sido igual al número totales de decimales que los números multiplicados
[~,elem2]=datos2bin(reg_Mul(2,:),1,3,(bitsFracPCA+3));
[~,elem3]=datos2bin(reg_Mul(3,:),1,3,(bitsFracPCA+3));
[~,elem4]=datos2bin(reg_Mul(4,:),1,3,(bitsFracPCA+3));
[~,elem5]=datos2bin(reg_Mul(5,:),1,3,(bitsFracPCA+3));
[~,elem6]=datos2bin(reg_Mul(6,:),1,3,(bitsFracPCA+3));
```

```
    % Usando Truncamiento
    sumsumarTruncamiento1=sumarTruncamiento(elem1,(bitsFracPCA+3),bitsSumTt)/2^(bitsFracPCA+3);
    sumsumarTruncamiento2=sumarTruncamiento(elem2,(bitsFracPCA+3),bitsSumTt)/2^(bitsFracPCA+3);
    sumsumarTruncamiento3=sumarTruncamiento(elem3,(bitsFracPCA+3),bitsSumTt)/2^(bitsFracPCA+3);
    sumsumarTruncamiento4=sumarTruncamiento(elem4,(bitsFracPCA+3),bitsSumTt)/2^(bitsFracPCA+3);
    sumsumarTruncamiento5=sumarTruncamiento(elem5,(bitsFracPCA+3),bitsSumTt)/2^(bitsFracPCA+3);
    sumsumarTruncamiento6=sumarTruncamiento(elem6,(bitsFracPCA+3),bitsSumTt)/2^(bitsFracPCA+3);
    sumsumarTruncamiento=[sumsumarTruncamiento1 sumsumarTruncamiento2 sumsumarTruncamiento3
sumsumarTruncamiento4 sumsumarTruncamiento5 sumsumarTruncamiento6];


    %% Predictores
    Predictores_Double(jj,:) = datos*trainedClassifier.PCACoefficients;
    Predictores_PuntoFijo(jj,:) =sum(reg_Mul,2);
    Predictores_puntoFijo_Truncamiento(jj,:) = sumsumarTruncamiento;


    %% Prediciones
    [pred_Double(jj),score_Double(jj,:)] = predict(trainedClassifier.ClassificationSVM,
datos*trainedClassifier.PCACoefficients);
    [pred_PuntoFijo(jj),score_PuntoFijo(jj,:)] = predict(trainedClassifier.ClassificationSVM, sum(reg_Mul,2)');
    [pred_PuntoFijo_Truncamiento(jj),score_puntoFijo_Truncamiento(jj,:)] = predict(trainedClassifier.ClassificationSVM,
sumsumarTruncamiento);


end
function bin=adaptarPCA(bin2)
    aux=char(bin2);
    aux=[aux(:,1) aux(:,3:end)]; % Se deja una 1 bit en la parte entera
    bin=string(aux);
end
```