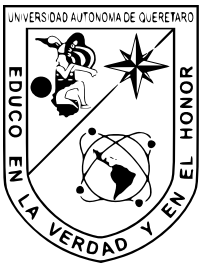


Universidad Autónoma de Querétaro

Facultad de Ingeniería

Maestría en Ciencias en Control en Sistemas Embebidos

Desarrollo de sistema en FPGA para control de dispositivo mediante señales EMG e inteligencia artificial

TESIS

Que como parte de los requisitos para obtener el grado de
Maestro en Ciencias en Control en Sistemas Embebidos

Presenta:

José Félix Castruita López

Dirigido por:

Dr. Marcos Romo Avilés

SINODALES

Dr. Marcos Romo Avilés
Presidente

Dr. Juvenal Rodríguez Reséndiz
Codirector

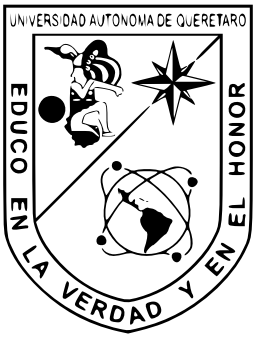
Dr. José Manuel Álvarez Alvarado
Vocal

Dr. Edson Eduardo Cruz Miguel
Suplente

Dr. Suresh Thenozhi
Suplente

Centro Universitario
Querétaro, QRO
México.
Julio 2025

Universidad Autónoma de Querétaro



Facultad de Ingeniería
Maestría en Ciencias en Control en Sistemas Embebidos

Desarrollo de sistema en FPGA para control de dispositivo mediante señales EMG e inteligencia artificial

Tesis

Que como parte de los requisitos para obtener el grado de
Maestro en Ciencias en Control en Sistemas Embebidos

Presenta:
José Félix Castruita López
Dirigido por:
Dr. Marcos Romo Avilés
Co-Director:
Dr. Juvenal Rodríguez Reséndiz
Vocal:
Dr. José Manuel Álvarez Alvarado
Suplente 1
Dr. Edson Eduardo Cruz Miguel
Suplente 2
Dr. Suresh Thenozhi

Querétaro, Qro. a Julio de 2025

A mi familia.

Agradecimientos

Agradezco a mi familia, amigos y a todas las personas que me apoyaron de alguna forma para completar esta meta. A todos por escucharme, acompañarme, aconsejarme y compartir su tiempo en esta etapa de mi vida. A la Universidad Autónoma de Querétaro por la formación y a los profesores por sus enseñanzas. Finalmente, al CONAHCYT ahora SECIHTI por el apoyo económico que me permitió estar en el posgrado.

Resumen

Las señales de electromiografía (EMG) son bipotenciales que registran la actividad muscular, como contracción, relajación, fuerza, movimiento y fatiga. Estas señales, al proporcionar información sobre el comportamiento muscular son muy utilizadas en campos de investigación, como la biomédica, la medicina, la robótica, y la rehabilitación. Sin embargo, las señales de EMG al ser muy complejas requieren un considerable nivel computacional para su clasificación. Tradicionalmente, este procesamiento se realiza en dispositivos de cómputo secuencial, lo que genera tiempos de respuesta elevados y un alto consumo de energía, limitando su uso en aplicaciones embebidas o portátiles. En este trabajo se diseñó e implementó un sistema de control basado en la clasificación de señales EMG mediante una red neuronal tipo perceptrón multicapa (MLP), implementada en un dispositivo lógico programable tipo FPGA. El objetivo fue realizar la inferencia del modelo MLP directamente en hardware para clasificar señales EMG procesadas y controlar en lazo cerrado un actuador que simula los movimientos de la mano predichos por la red. El sistema implementado en FPGA alcanzó una latencia de 843 ciclos de reloj para tener una predicción, equivalente a 33.72 μs por muestra a una frecuencia de 25 MHz, en contraste con los 52.03 ms requeridos por el modelo ejecutado en Python en una computadora con procesador Intel® Core™ i7-1255U. La eficiencia de clasificación fue del 97.42 %, y el consumo de potencia del diseño en el FPGA fue de 0.153 W. Estos resultados demuestran una mejora significativa en el tiempo de respuesta y el consumo energético respecto a los sistemas secuenciales tradicionales, validando la viabilidad del uso de FPGA para la implementación de modelos de inteligencia artificial embebidos en aplicaciones de control basadas en señales EMG.

Abstract

Electromyography (EMG) signals are biopotentials that record muscle activity such as contraction, relaxation, strength, movement, and fatigue. By providing information about muscle behavior, these signals are widely used in research fields such as biomedical engineering, medicine, robotics, and rehabilitation. However, due to their complexity, EMG signals require a considerable computational load for accurate classification. Traditionally, this processing is carried out on sequential computing devices, which leads to high response times and increased power consumption, limiting their use in embedded or portable applications. In this work, a control system based on the classification of EMG signals was designed and implemented using a multilayer perceptron (MLP) neural network deployed on a Field Programmable Gate Array (FPGA). The objective was to perform the inference of the MLP model directly in hardware to classify processed EMG signals and to control, in a closed loop, an actuator that simulates the hand movements predicted by the network. The FPGA-based system achieved a latency of 843 clock cycles per prediction, equivalent to 33.72 μs per sample at a frequency of 25 MHz, in contrast to the 52.03 ms required by the Python-based model running on an Intel® Core™ i7-1255U processor. The classification accuracy reached 97.42 %, and the FPGA design consumed only 0.153 W of power. These results demonstrate a significant improvement in response time and energy efficiency compared to traditional sequential systems, validating the feasibility of using FPGAs for the implementation of embedded artificial intelligence models in control applications based on EMG signals.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

En la actualidad, existen una gran cantidad de estudios con el uso de señales EMG, las cuales son señales bioeléctricas que se generan al contraer y relajar los músculos [1]. Estas señales proporcionan información valiosa sobre la actividad muscular y son utilizadas para desarrollar o mejorar tecnologías enfocadas en su análisis e interpretación, aplicadas a procesos o modelos que pueden predecir o clasificar movimientos [2].

Las señales EMG son muy complejas al provenir del sistema nervioso, por lo que su clasificación suele utilizan algoritmos de inteligencia artificial, como algoritmos de aprendizaje supervisado. Diversos autores han empleado distintos clasificadores, entre ellos redes neuronales [3], máquinas de soporte vectorial (SVM, por las siglas en inglés)[4] y algoritmos de K vecinos más cercanos (KNN, por las siglas en inglés) [5].

Sin embargo, al utilizar algoritmos de inteligencia artificial complejos, se dificulta la portabilidad de los modelos, ya que al aumentar la complejidad también aumenta la capacidad necesaria de procesamiento del equipo de cómputo donde se implementa [6]. Por lo que esto genera un desafío al integrar clasificadores de señales EMG a sistemas portátiles, ya que no es posible utilizar equipos de escritorio (PC) que son difíciles de transportar y requieren de una gran fuente de energía constantemente.

Ante esta problemática han surgido soluciones donde se implementan los clasificadores EMG en dispositivos y procesadores embebidos, como en [7, 8], donde se utilizan microcontroladores que destacan por su bajo consumo energético en comparación con otros dispositivos empleados en el estado del arte, pero tienen el inconveniente de estar limitados en su capacidad para ejecutar algoritmos complejos. Otros trabajos reportados, como [9, 10], emplean sistemas en chip (SoCs) como Raspberry® o Jetson Nano®, que permiten la ejecución de algoritmos más complejos que los microcontroladores, pero consumen una mayor cantidad de energía.

Por otro lado, en los trabajos [11, 12] se utilizan FPGAs, los cuales se presentan como una buena opción para embeber clasificadores EMG por sus características de procesamiento en paralelo y su eficiencia energética adecuada para aplicaciones portátiles, aunque su implementación resulta más desafiante que en otros dispositivos.

Estos trabajos presentan alternativas viables para integrar clasificadores EMG en sistemas portátiles; sin embargo, existen áreas de oportunidad, ya que la mayoría no utiliza clasificadores complejos, lo que limita la precisión de los modelos. Por otro lado, aquellos que ejecutan algoritmos más complejos, como redes neuronales, en su mayoría utilizan SoCs, lo que conlleva un consumo de energía considerable. Por ejemplo, en [13] se reporta que se requiere hasta 10 W de potencia, en

comparación con [11], uno de los pocos trabajos reportados de una red neuronal implementada en FPGA, donde el consumo fue de 91.81 mW.

Por lo que se propone este trabajo en el cual se diseña e implementa un sistema basado en un dispositivo FPGA para la clasificación de señales EMG utilizando una red neuronal tipo MLP, con arquitectura 67-100-100-10 (entradas - neuronas en capas ocultas - salidas), donde las entradas corresponden a características extraídas de las señales EMG y las salidas corresponden a los gestos que se predicen. Este sistema permite identificar 10 gestos de la mano, a partir de las señales EMG y posteriormente se controla un actuador que simula el movimiento.

1.1. Descripción del problema

Los algoritmos de inteligencia artificial, como las redes neuronales artificiales (RNA), se han vuelto extremadamente populares para la clasificación de señales EMG, con la posibilidad de aplicarlas en prótesis inteligentes, rehabilitación o seguimiento de la actividad muscular [14]. Por lo tanto, es necesario implementar estas RNA en campo para integrarlas en la vida diaria [15].

La mayoría de los clasificadores EMG reportados en la literatura se han implementado en *software*, ya que esta metodología resulta más sencilla y rápida de desarrollar. Sin embargo, su rendimiento suele ser limitado para aplicaciones en tiempo real o portátiles. Esto se debe a que los procesadores utilizados en dichas implementaciones como las computadoras de propósito general o los microprocesadores secuenciales ejecutan las operaciones de manera serial, aunque intenten paralelizar los cálculos mediante hilos. En consecuencia, el procesamiento de las neuronas se realiza de forma seudoparalela, lo que provoca tiempos de respuesta elevados y una menor eficiencia en tareas que requieren alta simultaneidad de operaciones [16]. Además, no resulta práctico para aplicaciones en campo utilizar procesadores de gran tamaño o incluso unidades de procesamiento gráfico, debido al espacio requerido y al alto consumo energético [17].

Por otro lado, las implementaciones en *hardware* ofrecen un mejor rendimiento en tiempo real, menor latencia y bajo consumo de energía, aunque su diseño es más complejo y requiere una optimización cuidadosa de recursos como memoria, bloques lógicos y arquitectura del modelo. Esta complejidad técnica ha limitado la cantidad de desarrollos reportados, convirtiéndose en un área de oportunidad, ya que las implementaciones en hardware ofrecen ventajas como alta flexibilidad, mayor precisión, mejor replicabilidad, alta capacidad de prueba y menor consumo energético [18].

El tipo de implementación en *hardware* que se plantea en este trabajo es mediante el uso de un dispositivo lógico programable tipo FPGA, el cual permite crear varios módulos lógicos simples que pueden interconectarse para formar sistemas más complejos. Esto resulta adecuado para diseñar el modelo en *hardware* de una neurona básica, con el fin de modularla y construir una red completa [16]. De esta forma, los FPGA se presentan como una mejor opción para implementar RNA, debido a su verdadero procesamiento en paralelo, bajo consumo energético y flexibilidad en su programación.

En la figura 1.1 se muestra un esquema simplificado que representa el uso de algoritmos de inteligencia artificial en aplicaciones útiles.

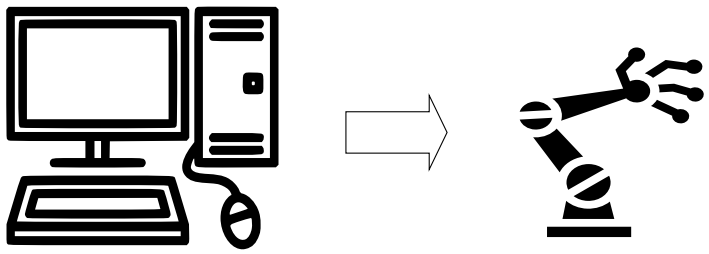


Figura 1.1: Transición del modelo en PC hacia una aplicación embebida de control.

1.2. Justificación

La Organización Mundial de la Salud estima que aproximadamente el 15 % de la población mundial vive con alguna forma de discapacidad. En este contexto, la discapacidad motriz a nivel global representa el 52 %, lo que indica un alto número de personas que experimentan problemas en los músculos y el sistema esquelético [19]. En México, según datos del Instituto Nacional de Estadística y Geografía de 2015, el número de personas con discapacidad debido a un miembro amputado se acercaba a las 780 mil. De estos casos, el 82 % se debía a enfermedades vasculares derivadas de la Diabetes Mellitus Tipo 2, el 16 % era el resultado de amputaciones traumáticas y el 2 % restante correspondía a amputaciones causadas por tumores malignos o enfermedades congénitas [20]. En la figura 1.2 se presenta una gráfica con estos datos. Así, esta información destaca la necesidad en México de desarrollar tecnología y soluciones para personas que han perdido alguna de sus extremidades. Además, es importante señalar que los individuos en esta situación pueden enfrentar dificultades económicas al no poder cumplir con las demandas de diversos entornos laborales, por lo que es de importancia presentar alternativas más accesibles para todos.

Figura 1.2: Distribución porcentual de las causas de amputación en México, según INEGI 2015.

De manera local, en el estado de Querétaro se tienen algunas instituciones que ofrecen soluciones a personas con alguna amputación, tal como el CRIMAL IAP Querétaro, como institución privada, fabrica prótesis para aproximadamente 130 pacientes al año, registro al año 2016. Los Centros de Rehabilitación Infantil Teleton también colaboran con el sector privado para proporcionar prótesis a alrededor de 775 pacientes al año [21]. Estas instituciones realizan una gran contribución al atender personas amputadas, pero según los registro en instituciones hospitalarias del sector salud en México, tan solo en 2014, se registraron 2179 amputaciones traumáticas por causa accidental

o intencional [21]. Lo que indica que aún hay un gran sector de la población que sufre de este problema. De manera significativa, se tiene un registro que el 70.9 % de las afectaciones fue en las extremidades torácicas (principalmente mano y dedos de la mano) con base en la clasificación de las amputaciones traumáticas [20]. Y considerando que la mano es uno de los órganos principales en la extremidad superior del ser humano, ya que desempeña un papel fundamental al permitir una manipulación física delicada y precisa. La amputación de la mano genera graves repercusiones físicas y psicológicas [22]. Debido a estas necesidades el estudio de la electromiografía ha cobrado gran importancia en diversas áreas de investigación, como la biomédica, la medicina, la robótica, la rehabilitación y la inteligencia artificial [14]. Esto ha llevado al desarrollo de dispositivos que pueden simular eficazmente el movimiento de una extremidad. En este contexto, es de suma importancia el estudio y aplicación de las señales EMG ya que son una fuente valiosa, ya que proporcionan datos precisos sobre el funcionamiento de los músculos y nervios, registrando las señales eléctricas generadas por los músculos durante el movimiento[23].

1.3. Hipótesis y Objetivos

En esta sección se presenta la hipótesis, objetivo general y los objetivos específicos del proyecto.

1.3.1. Hipótesis

La implementación de una red neuronal perceptrón multicapa en un dispositivo FPGA para la clasificación de señales EMG asociadas a 10 movimientos de mano logrará una eficiencia igual o mayor al 93 %, disminuyendo el consumo energético y el tiempo de procesamiento en contraste con implementaciones en software.

1.3.2. Objetivo general

Desarrollar un sistema de control en FPGA de un dispositivo que simule el comportamiento de la mano, basado en un perceptrón multicapa para la detección de movimientos mediante señales EMG.

1.3.3. Objetivos particulares

- Elaborar un dispositivo actuador para simular los movimientos de mano.
- Extraer características de las señales EMG y seleccionar las que presenten mejor rendimiento por medio de algoritmo genético al evaluar modelo perceptrón multicapa de prueba.
- Diseñar e implementar la fase de entrenamiento del modelo perceptrón multicapa en software.
- Diseñar e implementar la fase de ejecución del perceptrón en FPGA con lenguaje descriptor de hardware VHDL.
- Diseñar e implementar el control del actuador que simule movimientos de la mano en FPGA con lenguaje descriptor de hardware VHDL.
- Validar la implementación mediante el análisis de resultados al compararlos con la implementación en software.

1.4. Antecedentes

Actualmente, la evolución de los algoritmos de *Deep Learning* (DL) ha marcado un hito en las técnicas de inteligencia artificial. Destacando porque la información se procesa en capas jerárquicas que permiten realizar representaciones de los datos y comprender sus características en niveles cada vez más complejos [24]. En la práctica, la mayoria de algoritmos de DL se basan en redes neuronales artificiales, que comparten propiedades básicas comunes. Estas redes consisten en neuronas interconectadas dispuestas en capas, diferenciándose principalmente en la arquitectura de la red y, ocasionalmente, en su proceso de entrenamiento [14, 25].

Una de las aplicaciones más empleadas del DL es el procesamiento de señales, para la clasificación, análisis e interpretación de las mismas, con el fin de desarrollar u optimizar procesos o modelos [24]. De manera relevante, la clasificación de señales de electromiografía ha sido estudiada en los recientes años en ámbitos de ingeniería biomédica. Pero el problema con estas señales recae en que son muy complejas, ya que provienen de la actividad muscular, que a su vez es controlada por el sistema nervioso. Además, dependen de las propiedades anatómicas y fisiológicas de los músculos. Así que, las señales EMG son consideradas un bipotencial que registra corrientes eléctricas generadas en los músculos durante su contracción/relajación [26]. Por lo que se suelen emplear métodos de DL como el perceptrón multicapa o *multilayer perceptron* (MLP) para la clasificación de este tipo de señales. El MLP consiste en una arquitectura de RNA donde se interconectan neuronas, activadas por una función no lineal [14]. En comparación con otras arquitecturas más complejas, como las redes neuronales convolucionales (CNN), el MLP presenta una estructura más simple y regular que puede representarse mediante operaciones basicas de multiplicacion y suma, lo que facilita su implementación directa en hardware reconfigurable como FPGA. El desafío con el uso de este método es conseguir un desempeño adecuado por medio de los hiperparámetros en su proceso de aprendizaje, lo que puede resultar en la necesidad de altos recursos computacionales.

Debido a la alta complejidad de algunas tareas de procesamiento, como la clasificación de señales EMG, los algoritmos de DL también se han vuelto más complejos desde el punto de vista computacional. Además, el consumo energético y tiempo de ejecución en dichos algoritmos también se ha elevado considerablemente. Por ello, encontrar la tecnología que permita una implementación eficiente en tiempos y consumo energético es fundamental en la clasificación. Recientemente, se han utilizado unidades de procesamiento central o *central process units* (CPUs) y unidades de procesamiento gráfico o *graphics processing unit* (GPUs) para implementar estos algoritmos [27]. A pesar de ello, las CPUs suelen requerir mucho espacio y funcionan con procesamiento secuencial. Y en el caso de las GPUs utilizan un consumo eléctrico elevado. Como alternativa para eliminar estas desventajas, los dispositivos de procesamiento FPGA han demostrado ser prometedores en términos de consumo de energía y rendimiento. Además, son adecuadas para la implementación de algoritmos de DL por su ejecución en paralelo [16]. Las FPGA ofrecen un diseño específico de hardware que garantiza un mayor rendimiento, menor consumo de energía y costos reducidos en comparación con una implementación en CPUs y GPUs [28].

En este contexto, esta tesis se enfoca en la investigación e implementación de una RNA tipo MLP utilizando un dispositivo FPGA con el fin de clasificar señales de EMG provenientes de movimientos de mano. Ya que, se desarrollará un prototipo de dispositivo, que simule movimientos de una mano. Con el fin de realizar funciones como agarre, posicionamiento de dedos, entre otras.

se busca abandonar la implementación en *software* de la red, que suele ser una implementación de algoritmos y librerías ya definidas en forma de instrucciones que se ejecutan en un procesador. Para pasar a una implementación a nivel de *hardware* con el objetivo de reducir los costos y aumentar la velocidad de procesamiento con menor consumo de energía. Gracias a las características propias de las FPGA que resultan más adecuadas para aplicaciones portátiles y que requieran procesamiento en paralelo como el problema que se abordará.

Principales referencias en la literatura

En esta subsección, se encontrarán las investigaciones, proyectos y publicaciones más relevantes a nivel global acerca la implementación de algoritmos para clasificar señales de EMG en dispositivos portátiles.

Tabla 1.1: Referencias de las investigaciones y publicaciones más importantes.

Ref.	**Año**	**Descripción**	**Precisión %**	**Movs.**	**Tecnología**
[11]	2023	Reconocimiento de gestos basado en un sensor EMG de tipo seco y redes neuronales binarizadas implementadas en FPGA.	95.4	9	FPGA
[29]	2024	Diseño e implementación de un modelo para autenticar usuarios mediante señales EMG, arquitectura optimizada para implementarse en FPGA.	99.0	12	FPGA
[17]	2020	Comparación de MLP y CNN en FPGA en términos de retardo y consumo.	96.0	5	FPGA
[30]	2024	Se implementa una red neuronal de picos para reconocer movimientos mediante señales EMG, se hace la inferencia del modelo en FPGA.	83.1	12	FPGA
[31]	2022	Combina datos de sensores EMG y visuales para clasificar gestos, mediante una red neuronal recurrente de picos, se hace inferencia en FPGA.	63.9 (solo EMG)	5	FPGA
[32]	2023	Clasificación de señales EMG utilizando una red neuronal ligera implementada en FPGA.	95.0	6	FPGA
[12]	2023	Autenticación de usuarios basado en señales EMG utilizando redes neuronales *Siamese* y transformada MODWT. Implementado en FPGA.	90.0	1	FPGA

Continúa en la siguiente página

Tabla 1.1 – continuación de la página anterior

Ref.	Año	Descripción	Precisión (%)	Movs.	Tecnología
[33]	2023	Reconocimiento de gestos de la mano utilizando señales EMG. Incluye extracción de características en el dominio tiempo y aprendizaje supervisado (DT, KNN, SVM) para controlar una prótesis.	94.0	4	Raspberry Pi®
[34]	2024	Clasificación de movimientos de la mano usando señales EMG, diseñado específicamente para pruebas en prótesis de mano subactuadas.	92.5	5	Jetson Nano®
[35]	2024	Describe el diseño de una prótesis de rodilla controlada en tiempo real por señales EMG, utiliza varios algoritmos implementados en SOC.	80.0	20	Raspberry Pi®
[36]	2024	Algoritmos kNN implementados en el microcontrolador para clasificar señales EMG.	94.7	6	ARM Cortex-M4®
[37]	2024	Diseño de un sistema en tiempo real para el análisis y clasificación de señales EMG en una plataforma embebida, se utiliza una red neuronal MLP.	95.3	5	ARM Cortex-M4®
[38]	2024	Sistema portátil basado en EMG y algoritmo de inteligencia artificial para reconocer seis expresiones faciales.	90.0	6	Arduino Nano IoT®
[39]	2023	Sistema de reconocimiento de gestos en plataforma Zynq (XC7Z020).	97.7	6	FPGA

De acuerdo con la Tabla 1.1, se observa que la mayoría de los trabajos recientes (2023–2024) donde se embeben modelos de clasificación EMG en FPGA priorizan el uso de redes neuronales ligeras, como MLP optimizados o variantes binarizadas, con resultados superiores al 90 % de precisión. Esto evidencia el potencial de esta arquitectura para ejecutar inferencias en tiempo real de modelos de redes neuronales.

Trabajos previos en la UAQ

En esta subsección se encontrarán los trabajos realizadas en la universidad autónoma de Querétaro relacionados a el proyecto, lo cual se puede observar en el Cuadro.

Tabla 1.2: Artículos e investigaciones utilizados para los antecedentes a nivel local.

Ref.	Año	Descripción	Precisión %	Movs.	Tecnología
[40]	2023	El trabajo se centra en utilizar algoritmos genéticos para la selección de características de señales EMG.	90.0	7 (pierna derecha)	PC
[41]	2013	Diseño en FPGA de una red neuronal artificial (RNA) para la identificación en línea de un sistema.	99.9 (identificación de sistema, no clasificación EMG)	-	FPGA
[42]	2019	Análisis de señales EMG mediante IA; abarca recolección, tratamiento y clasificación.	-	-	PC
[43]	2021	Clasificación de señales EMG con máquinas de soporte vectorial y tres kernels.	92–98	10 (mano)	PC

Según la Tabla 1.2 los trabajos de la UAQ han explorado más la clasificación en PC, sin llegar aún a implementaciones prácticas en FPGA, lo que justifica el presente trabajo que se alinea con la tendencia internacional, proponiendo una implementación práctica en FPGA que busca equilibrar precisión, velocidad y consumo de recursos, aportando un paso más hacia la traslación del modelo en PC a una aplicación embebida funcional.

Tabla de patentes

En esta subsección se encontrarán patentes relevantes con el uso de señales EMG.

Tabla 1.3: Patentes relacionadas con sistemas de control basados en EMG.

Año	Referencia	Título	Descripción
2020	[44]	*EMG control systems and methods for instructing extracorporeal prosthesis users*	Se describen sistemas y métodos, tanto en hardware como en software, para la obtención y análisis de señales EMG, con el fin de realizar la calibración, funcionamiento y control de prótesis de brazo.
2016	[45]	*Electromyography with prosthetic or orthotic devices*	Se describen sistemas, métodos y dispositivos para el funcionamiento y control de prótesis mediante señales EMG. Pueden ser para extremidades superiores o inferiores, y usar sensores externos, subcutáneos, entre otros.

Continúa en la siguiente página

Tabla 1.3 – continuación de la página anterior

Año	**Referencia**	**Título**	**Descripción**
2013	[46]	*Method, system and apparatus for real-time classification of muscle signals from self-selected intentional movements*	Se propone un nuevo método para asignar señales EMG a contracciones musculares que corresponden a funciones específicas de una prótesis. Las señales son clasificadas en tiempo real a partir de movimientos intencionales.

En la Tabla 1.3 se presentan algunas patentes relevantes en el área de control de prótesis mediante señales EMG. Estas patentes, registradas principalmente entre 2013 y 2020, describen sistemas y métodos orientados al control de extremidades protésicas. Sin embargo, a diferencia de estos desarrollos patentados, la presente tesis no busca diseñar una prótesis comercial, sino optimizar la etapa de clasificación mediante una red neuronal MLP implementada en FPGA. Por tanto, el análisis de estas patentes sirve únicamente para contextualizar la evolución tecnológica del uso de EMG en aplicaciones biomédicas.

1.5. Estructura de la tesis

Esta tesis está estructurada de la siguiente manera: en el capítulo 1 se presenta la introducción donde se explica cómo surge este proyecto, sobre el uso de señales EMG para controlar dispositivos como prótesis, el uso reciente de algoritmos de inteligencia artificial y la importancia de implementar los modelos en dispositivos portátiles como FPGA. También, en este capítulo se presenta la problemática, justificación, hipótesis, objetivos y los antecedentes encontrados en la revisión de literatura, relacionadas con la implementación de clasificadores EMG en dispositivos FPGA. En el capítulo 2 se muestra el marco teórico donde se describen las señales EMG, así como las técnicas de procesamiento de ellas. Después se describe el algoritmo genético. Posteriormente, se describe la red neuronal tipo perceptrón multicapa y sus hiperparámetros y, por último, se describe que es un FPGA y su arquitectura general. En el capítulo 3 se encuentra la metodología para el desarrollo del proyecto. En el capítulo 4 se muestran los resultados obtenidos del trabajo y su discusión. Por último, en el capítulo 5 se presentan las conclusiones que se obtuvieron en el proyecto.

CAPÍTULO 2

Fundamentación teórica

En este capítulo se describe el marco teórico correspondiente a los temas más importantes para comprender las bases de esta tesis. Describe las señales EMG, sus características y algunas técnicas de procesamiento. También, se describe la arquitectura de la red neuronal MLP, se describe el algoritmo genético como herramienta de optimización y, finalmente, se presentan los fundamentos de los dispositivos FPGA.

2.1. Señales EMG

Las señales EMG son señales eléctricas musculares o bioeléctricas que registran la actividad muscular, revelando información sobre la fuerza, el movimiento y la fatiga. Esta actividad muscular se registra mediante electrodos, ya sean invasivos (como agujas en el músculo) o no invasivos (colocados en la piel). Por lo que, la señal EMG es una representación de los potenciales de acción de las fibras musculares, que ocurren a intervalos aleatorios [47]. Aunque, la detección de estas señales presenta dos problemas clave. El primero es la relación señal-ruido, que se refiere a la proporción entre la energía de las señales de EMG y la energía del ruido no deseado. El segundo problema en la detección es la distorsión de la señal, la cual implica que la contribución relativa de las frecuencias en la señal no debe alterarse [47]. Por ello, estas señales requieren preprocesamiento para eliminar el ruido y mejorar su amplitud, ya que, típicamente, su amplitud está entre 0.1 mV y 10 mV [14, 48, 47]. Además, hay que tener en cuenta la colocación precisa de los electrodos en la superficie muscular para obtener mediciones exactas. Así como, evitar movimientos no deseados de los electrodos e interferencias electromagnéticas que puedan comprometer la precisión de la señal [14, 48, 49].

2.1.1. Adquisición de señal EMG

La adquisición de la señal es una etapa que requiere mucha atención, ya que los procesos subsiguientes y la precisión de la estimación dependen de la calidad de la señal [49]. Según [50] esta etapa de adquisición consiste principalmente en :

- El método utilizado para registrar la señal.
- El dispositivo de adquisición.

- El número de canales y la posición de los músculos.
- El diseño del amplificador y del filtro.
- La frecuencia de muestreo.

2.1.2. Filtrado y tasa de muestreo

Las señales EMG se preprocesan antes de realizar la extracción de sus características. El cual consiste en amplificar y filtrar la señal [50, 48]. Típicamente, se utiliza un filtro pasa-banda, ya que la información importante de las señales EMG se encuentra en la banda de frecuencia de 0 Hz a 500 Hz, y principalmente la energía se concentra en 20 Hz a 150 Hz [49, 50]. Por lo que, comúnmente, la frecuencia de corte inferior en el filtro es entre 5 y 20 Hz, con el fin de eliminar variaciones causadas por el movimiento de electrodos o cables, las cuales suelen estar en el rango de 0 a 20 Hz [48, 49]. Mientras que la frecuencia de corte superior comúnmente es de 500 Hz, ya que por encima de esta frecuencia se considera ruido en la señal [50, 48]. De manera adicional, se suele utilizar un filtro tipo Notch a 50 o 60 Hz para eliminar el ruido que pueda interferir de la frecuencia de alimentación [48, 49]. Además, como se registra una gran cantidad de información, se realiza un muestro de la señal. Que comúnmente, con un filtro pasa bajas a 500 Hz, la tasa de muestreo es de 1 KHz. Ya que, según la regla de Nyquist, la tasa de muestreo debe ser igual al doble de la frecuencia más alta de interés. Esto con el fin de evitar que la señal sea indistinguible al momento de digitalizarla [50]. De manera adicional, otro punto importante es la amplificación de la señal, la cual suele ser entre 100 y 5 mil veces. Esto ya que la señal tienen una amplitud máxima por debajo de 10 mV, lo que la hace propensa a interferencias. Sin embargo, la mayoría de sistemas de adquisición ya realizan la amplificación de la señal automáticamente [50].

2.1.3. Segmentación

El análisis de las señales EMG se suele hacer en segmentos de tiempo o también llamados, épocas o ventanas. ya que la propiedad no estacionaria de la señal complica que los datos se analizan en tiempo real [50, 51]. En la segmentación, es importante elegir la técnica que se utilizara, el tamaño del segmento y el estado de la señal, para conseguir una buena precisión en la clasificación de estas [49]. Existen dos técnicas de segmentación, la adyacente y la superpuesta. En la segmentación adyacente se segmenta de manera consecutiva con una longitud personalizada. El problema con este tipo de segmentación es que no se aprovecha el tiempo de procesamiento, ya que el tiempo de procesamiento suele ser menor al tiempo de segmentación. En cambio, en la segmentación superpuesta se aprovecha mejor el tiempo de procesamiento, ya que se utiliza el tiempo inactivo del procesador para adquirir más datos [50]. En la figura 2.1 se muestra un ejemplo de las técnicas de segmentación.

El tamaño de los segmentos tiene relación en el tiempo de procesamiento de este. Un segmento muy grande puede aumentar la presión de la clasificación, pero el tiempo de procesamiento será mayor [50]. Así que se busca un equilibrio entre precisión y tiempo de procesamiento.

Para obtener el número de ventanas o segmentos en que se divide la señal se emplea la ecuación 2.1. Donde N es el número de ventanas resultantes, L es la longitud de la señal, w es el tamaño del segmento propuesto y t es el empalme entre los segmentos propuestos, toso representado en numero de puntos o muestras de la señal.

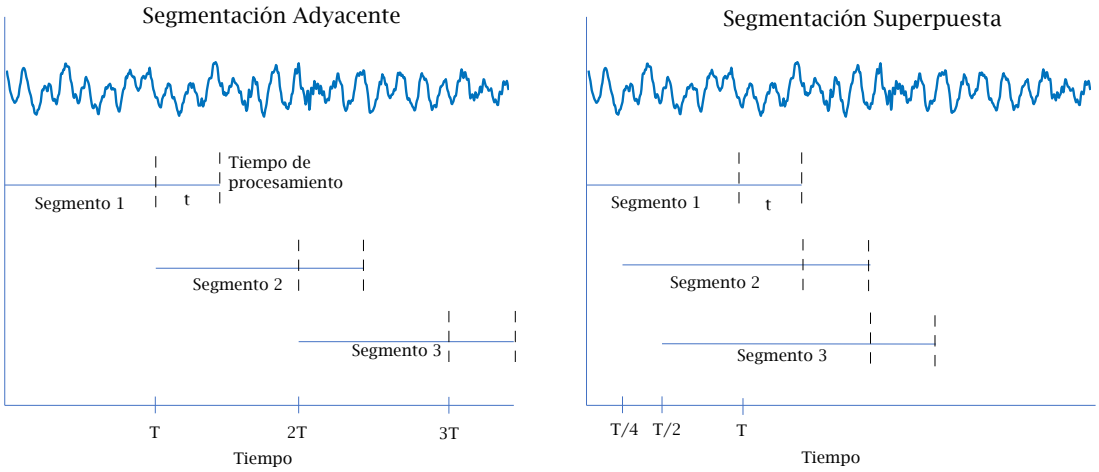


Figura 2.1: Técnicas de segmentación.

$$N = \left\lfloor \frac{L - w}{w - t} \right\rfloor + 1 \tag{2.1}$$

2.1.4. Extracción de características

Después de que la señal EMG fue adquirida y pre-procesada, se realiza la extracción de características [48]. Se asignan vectores de características de menor dimensión que la señal cruda, ya que las características describen mejor la información de la señal [50]. Las características se pueden agrupar dependiendo el dominio donde se calculan [14, 49, 48, 50, 51].

- Características en el dominio de tiempo.
- Características en el dominio de frecuencia.
- Características en el dominio tiempo-frecuencia.
- Características en el dominio espacial.

Los vectores de características, al estar formados de diferentes parámetros de la señal, son de suma importancia. Ya que la elección de los parámetros adecuados determina el éxito de la clasificación de la señal [48].

Características en el dominio de tiempo: Las características en el dominó del tiempo son utilizadas con mayor frecuencia debido a su menor complejidad computacional comparada con las características de otro tipo [14, 51, 49, 52]. Además, estas son extraídas directamente de la señal cruda, por lo que no es necesario aplicar algún tipo de transformada. El resultado es una función en el tiempo que gracias a su alta velocidad computacional son ampliamente utilizadas en modelos de clasificación y regresión [49, 14]. Estas características se utilizan principalmente para analizar el esfuerzo en el músculo o el nivel activo [52]. En el Cuadro 2.1 se muestran las características temporales comúnmente utilizadas.

Tabla 2.1: Cuadro de las características temporales comúnmente utilizadas.

Nombre	Fórmula	Función	Descripción
Valor absoluto medio	$\frac{1}{N}\sum_{k=1}^{N}\|x_k\|$	Amplitud promedio sin polaridad	Promedio de los valores absolutos de la señal EMG [48, 14].
Electromiograma integrado	$\sum_{k=1}^{N}\|x_k\|$	Suma de magnitudes absolutas	Tasa de detección inicial; indica activación muscular [48, 51].
Raíz cuadrada media	$\sqrt{\frac{1}{N}\sum_{k=1}^{N}x_k^2}$	Amplitud efectiva	Raíz cuadrada del promedio de los valores al cuadrado [48, 51].
Cruce por cero	$\sum_{k=1}^{N-1}f(x)$	Cambios de polaridad	Número de veces que la señal cruza por cero. $f(x)=1$ si $x_kx_{k+1}<0$ y $\|x_k-x_{k+1}\|\geq L$; de otro modo es cero [48, 14].
Cambios de signo de pendiente	$\sum_{k=2}^{N-1}f(x)$	Número de cambios de pendiente	$f(x)=1$ si x_k es un punto de inflexión. Representa cambios en la dirección de la señal [14].
Longitud de onda	$\sum_{k=2}^{N}\|x_k-x_{k-1}\|$	Actividad total de la señal	Mide la variación acumulada de la señal [14].
Varianza	$\frac{1}{N-1}\sum_{k=1}^{N}(x_k-\bar{x})^2$	Dispersión de la señal	Mide cuán dispersos están los valores de la señal respecto a la media [48].
Desviación estándar	$\sqrt{\frac{1}{N}\sum_{k=1}^{N}(x_k-\bar{x})^2}$	Medida de variabilidad	Raíz cuadrada de la varianza; útil para conocer la amplitud promedio de variación [48].
Amplitud Willison	$\sum_{k=1}^{N-1}f(\|x_{k+1}-x_k\|)$	Cantidad de cambios mayores a un umbral	Cuenta el número de veces que la diferencia entre muestras consecutivas supera un umbral dado [48].

Características en el dominio de frecuencia: las características en dominio de la frecuencia se utilizan principalmente para calcular la fatiga muscular [48, 52]. También se utilizan para complementar las características en el dominio del tiempo. Se extraen de la densidad espectral de potencia y se calculan mediante métodos paramétricos. En comparación Con las características en el dominio del tiempo, estas requieren un mayor costo computacional [49]. En el Cuadro 2.2 se muestran las características en el dominio de la frecuencia comúnmente utilizadas.

Tabla 2.2: Características en frecuencia comúnmente utilizadas en señales EMG.

Nombre	**Fórmula matemática**	**Descripción**
Frecuencia media	$MNF=\sum_{l=1}^{M}f_lP_l/\sum_{l=1}^{M}P_l$	Se conoce como frecuencia espectral promedio o frecuencia promedio de potencia. Donde f_l y P_l son la frecuencia y el espectro de potencia en el l-ésimo segmento en el dominio de la frecuencia o (bin). Y M es la longitud del bin [48, 51].

Continúa en la siguiente página

Tabla 2.2 – continuación de la página anterior

Nombre	**Fórmula matemática**	**Descripción**
Frecuencia mediana	$\sum_{l=1}^{MDF} P_l = \sum_{MDF}^{M} P_l$	MDF es la mitad de la potencia total de la señal en el *bin* actual. Donde P_l es el espectro de potencia en el l-ésimo *bin*, y M es la longitud del *bin* [48, 51].
Relación de frecuencias	$FR = \sum_{i=LLC}^{ULC} P_l / \sum_{i=LHC}^{UHC} P_l$	Se utiliza para distinguir entre contracción y relajación del músculo. Donde ULC y LLC representan el límite superior e inferior de corte de baja frecuencia de la señal EMG. Y UHC y LHC representan el límite superior e inferior de corte de alta frecuencia del de la señal EMG [48].
Frecuencia pico	$PKF = \text{máx}(P_l)$	Indica la ubicación de la frecuencia donde está la máxima potencia [51].

Características en el dominio tiempo-frecuencia: Las características en el dominio tiempo-frecuencia pueden localizar la energía de la señal en los dos dominios. Pero, tienen una mayor complejidad computacional debido a la transformación que requieren[52, 49, 48]. Por lo que hay pocos estudios que se basan en estas características, adicional, que es más compleja su interpretación [52]. Algunos algoritmos rápidos que incluyen las características en el dominio tiempo-frecuencia son la transformada de Fourier de tiempo corto, y la transformada wavelet. Estos logran cumplir los requisitos de tiempo real necesarios para la clasificación de la señal [48].

Características en el dominio espacial: Las características en el dominio espacial, han tenido relevancia al emplear la técnica de electromiografía de alta densidad [50]. Esta técnica consiste en colocar una rejilla densa de electrodos de superficie. Lo cual permite variar el uso de estos sobre un área de piel restringida. Lo que llevó a demostrar que regiones distintas del músculo se activan de manera variante dependiendo de la posición de la articulación [53]. Por lo que estas mediciones han permitido extraer características espaciales de las señales EMG registradas. Las cuales permiten diferenciar entre las posiciones y el nivel de fuerza, dependiendo la distribución de unidades motoras de acción potencial en los músculos. Lo cual, permite visualizar cómo se realiza la distribución de carga en los músculos [48, 50]. Aunque estas características tiene muy poca investigación y diseño, ya que la electromiografía de alta densidad se ha adaptado recientemente [54].

2.2. Redes neuronales artificiales

Las RNAs, son algoritmos que se inspiran en el funcionamiento de las redes neuronales biológicas del sistema nervioso. El desempeño de las neuronas en el cerebro se explica mediante el paradigma conexionista, cuyo principio fundamental es que los fenómenos mentales pueden ser explicados mediante redes de unidades simples interconectadas [55]. La neurona constituye la unidad fundamental del sistema nervioso, especialmente en el cerebro. Su función principal radica en procesar y combinar señales para luego transmitirlas a otras neuronas interconectadas. Las RNAs siguen un

esquema similar, donde la unidad principal es la neurona o perceptrón simple. En la figura 2.2, cada nodo corresponde a una neurona artificial, y las flechas simbolizan las conexiones que existen entre ellas, desde la salida de una hasta la entrada de la siguiente [56].

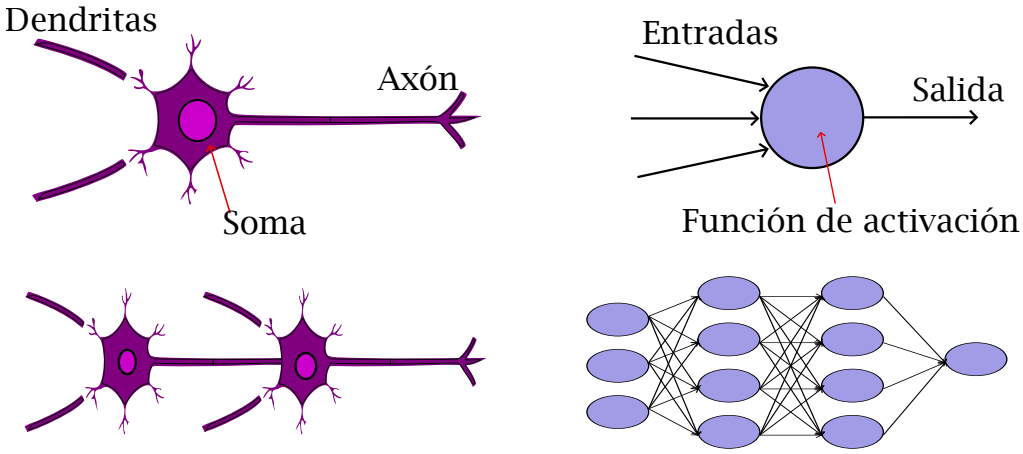


Figura 2.2: Esquema analogía entre neuronas biológicas y RNA.

En un principio, las redes neuronales tomaron su inspiración del cerebro. Sin embargo, con el tiempo, se dejó de intentar replicar cómo funciona el cerebro y en su lugar se busca encontrar las configuraciones adecuadas para tareas específicas [57].

Una red neuronal no es un programa fijo, sino más bien un modelo, un sistema que procesa información o entradas. Las características de una red neuronal de acuerdo a [24] son las siguientes :

- El procesamiento de la información ocurre en su forma más simple, a través de elementos simples llamados neuronas.
- Las neuronas están conectadas y se intercambian señales entre ellas a través de enlaces de conexión.
- Los enlaces de conexión entre las neuronas pueden ser más fuertes o más débiles, y esto determina cómo se procesa la información.
- Cada neurona tiene un estado interno que es determinado por todas las conexiones entrantes de otras neuronas.
- Cada neurona tiene una función de activación diferente que se calcula en función de su estado y determina su señal de salida.

Podemos identificar dos características principales para una red neuronal:

- La arquitectura de la red neuronal: Esto describe el conjunto de conexiones, es decir, feedforward, recurrente, de capa única o múltiple, y el número de neuronas en cada capa.
- El aprendizaje: Esto describe lo que comúnmente se define como el entrenamiento. La forma más común pero no exclusiva de entrenar una red neuronal es mediante el descenso de gradiente y la retropropagación (backpropagation).

Una neurona es una función matemática que toma uno o más valores de entrada y produce un único valor numérico como salida:

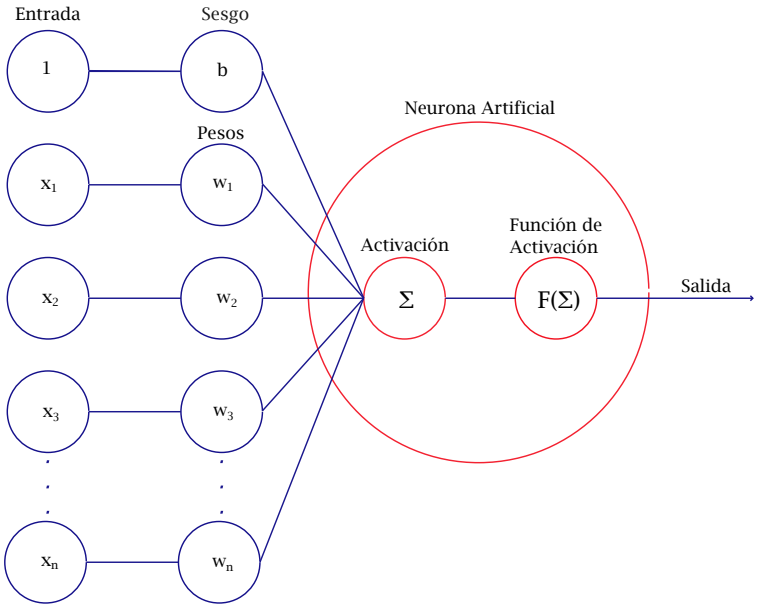


Figura 2.3: Modelo de neurona artificial.

Por lo que el modelo general sería el expuesto en la ecuación 2.2:

$$y = f\left(\sum_{i=1}^{n} x_i w_i + b\right) \tag{2.2}$$

1. Primero, se tiene la suma ponderada de las entradas x_i y los pesos w_i también conocida como valor de activación. En este caso, x_i puede ser valores numéricos que representan los datos de entrada o las salidas de otras neuronas es decir, si la neurona forma parte de una red neuronal. Los pesos w_i son valores numéricos que representan tanto la fuerza de las entradas como, alternativamente, la fuerza de las conexiones entre las neuronas. El peso b es un valor especial llamado sesgo cuya entrada siempre es 1 [58].

2. Luego, se utiliza el resultado de la suma ponderada como entrada para la función de activación f, que también se conoce como función de transferencia. Existen muchos tipos de funciones de activación, pero todas deben cumplir con el requisito de ser no lineales [58].

2.2.1. Perceptrón multicapa

La arquitectura del MPL es una de las redes neuronales artificiales más utilizadas [59]. En la figura 2.4 se presenta un MPL genérico que consta de las siguientes capas:

- Capa de entrada: Esta capa posee una o más entradas, cuya cantidad depende de la aplicación. Cada entrada se conecta y multiplica por el peso de cada neurona en la primera capa oculta. Se consideran como capas pasivas ya que solo transfieren el valor x a la siguiente capa.

- Capas ocultas: Estas capas comprenden una o más capas de neuronas, donde cada neurona está conectada a todas las neuronas en la capa siguiente mediante un peso. Se consideran capas activas ya que operan sobre los datos de entrada de las capas anteriores.

- Capa de salida: En esta capa se encuentran las neuronas que coinciden con el número de salidas de la red.

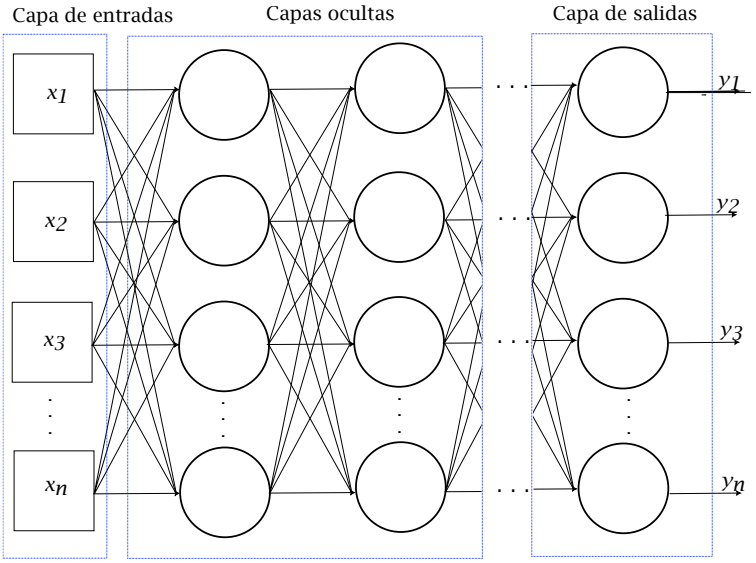


Figura 2.4: Red Neuronal Perceptrón Multicapa.

El empleo del modelo de red neuronal perceptrón multicapa tiene dos fases:

- Fase de entrenamiento, donde la red aprende a llevar a cabo una tarea específica.

- Fase de operación, en la cual la red ejecuta la tarea para la cual fue entrenada.

En la fase de entrenamiento, se destacan dos pasos fundamentales: la etapa de avance (*Forward stage*) y la etapa de retropropagación (*Back-Propagation*).

En la etapa de avance, se calcula la salida de la red a partir de los valores de entrada. Es decir, los valores de entrada pasan a través de las capas ocultas hasta obtener un valor de salida.

En la etapa de retropropagación, se calcula un error entre el valor que se espera y la salida obtenida en la etapa de avance. Este error se propaga a las neuronas dentro de la red mediante los pesos. Se calcula un error correspondiente a cada neurona, y se actualizan los pesos y el sesgo de la red [59].

2.2.2. Etapa de avance en el entrenamiento del MLP

Durante la etapa de avance, se realiza el cálculo de la salida de la red. Suponiendo que $xi = [x_1, x_2...x_n]$ representa las entradas de la red, w_{ij} es el peso de la neurona j asociado con la entrada

i, y_{hj} es la salida de la neurona j ($j = 1, 2...q$) en la capa h ($h = 1, 2...m$); b_{hj} es el sesgo. La entrada total de una neurona de la primera capa oculta ($h = 1$) se expresa mediante la siguiente ecuación:

$$s_{1j} = \sum_{i=1}^{n} (w_{ij} * x_i + b_{1j}) \tag{2.3}$$

El procesamiento de la acumulación en la entrada de una neurona se realiza a través de su función de activación no lineal. La expresión para la salida de una neurona en la primera capa oculta se describe mediante la ecuación:

$$y_{1j} = f(s_{1j}) = f\left(\sum_{i=1}^{n} (w_{ij} * y_i + b_{1j})\right) \tag{2.4}$$

Los cálculos efectuados por las neuronas en las capas ocultas siguientes y la capa de salida están representados por las ecuaciones:

$$s_{hj} = \sum_{k=1}^{m} (w_{kj} * y_k + b_{hj}) \tag{2.5}$$

$$y_{hj} = f(s_{hj}) = f\left(\sum_{k=1}^{m} (w_{kj} * y_k + b_{hj})\right) \tag{2.6}$$

Donde $k = (h-1)j$, representando a la neurona j en la capa anterior y w_{kj} es el peso entre las neuronas k y j.

2.2.3. Etapa de retropropagación en el entrenamiento del MLP

Durante esta etapa, el error en la salida se propaga hacia atrás dentro de la red y se utiliza el algoritmo de gradiente descendente para actualizar los pesos y el sesgo [59]. La actualización se lleva a cabo en tres pasos principales:

1. El error para las neuronas en la capa de salida, y el gradiente del error, se calculan por las siguientes ecuaciones.

$$\varepsilon_{salida,j} = (y_{deseada} - y_{salida,j}) \tag{2.7}$$

$$\delta_{salida,j} = \varepsilon_{salida,j} * f'(y_{salida,j}) \tag{2.8}$$

Donde $\varepsilon_{salida,j}$ es el error entre el valor deseado y la salida real de la neurona j en la capa de salida; $\delta_{salida,j}$ es el gradiente del error que se propaga a las neuronas en la capa oculta a través de los pesos.

Entonces el error propagado y el gradiente local están expresados por:

$$\varepsilon_{hj} = \delta_{(h+1)j} * w_{j,(h+1)j} \tag{2.9}$$

$$\delta_{hj} = \varepsilon_{hj} * f'(y_{h,j}) \tag{2.10}$$

Donde $w_{j,(h+1)j}$ es el peso entre la neurona j y la neurona siguiente.

2. La variación en los pesos y sesgo en la primera capa:

$$\Delta w_{ij} = \alpha * \delta_{1j} * xi \tag{2.11}$$

$$\Delta b_{1j} = \alpha * \delta_{1j} \tag{2.12}$$

Y la variación de los pesos y sesgo en las capas ocultas:

$$\Delta w_{kj} = \alpha * \delta_{hj} * y_{hj} \tag{2.13}$$

$$\Delta b_{hj} = \alpha * \delta_{hj} \tag{2.14}$$

Donde α es la tasa de aprendizaje, que determina cuanto se ajusta los pesos durante el entrenamiento. Si es muy alto, el entrenamiento es rápido, pero el modelo puede no ser muy preciso.

3. Actualización de los pesos y sesgo:

$$w_{kj}(siguiente) = w_{kj}(actual) + \Delta w_{kj}(actual) \tag{2.15}$$

$$b_{hj}(siguiente) = b_{hj}(actual) + \Delta b_{hj}(actual) \tag{2.16}$$

2.2.4. Funciones de Activación

Si las neuronas carecen de funciones de activación, su salida sería la suma ponderada de las entradas, lo que constituye una función lineal. En consecuencia, la red neuronal en su totalidad, que es una composición de estas neuronas, se convierte en una composición de funciones lineales, manteniendo así su naturaleza lineal. Esto implica que incluso al agregar capas ocultas, la red seguirá siendo equivalente a un modelo simple de regresión lineal, con todas sus limitaciones. Para introducir la no linealidad en la red, se utilizan funciones de activación no lineales en las neuronas. Por lo general, todas las neuronas en una misma capa comparten la misma función de activación, pero distintas capas pueden emplear funciones de activación diferentes [60]. Las funciones de activación más comunes son las siguientes:

- Función identidad. Esta función permite que el valor de activación pase a través de ella:

$$f(a) = a \tag{2.17}$$

- Función de actividad de umbral. Esta función activa la neurona; si la activación está por encima de cierto valor:

$$f(a) = \begin{cases} 1, \, si \; a \geq 0 \\ 0, \, si \; a < 0 \end{cases} \tag{2.18}$$

- Función logística o la sigmoidal logística. Esta función es una de las más comúnmente utilizadas, ya que su salida está acotada entre 0 y 1, y puede interpretarse de manera estocástica como la probabilidad de activación de la neurona:

$$f(a) = \frac{1}{1 + e^{-a}} \tag{2.19}$$

- Función sigmoidal bipolar. Es simplemente una sigmoidal logística redimensionada y desplazada para tener un rango en $(-1, 1)$:

$$f(a) = \frac{2}{1 + e^{-a}} - 1 = \frac{1 - e^{-a}}{1 + e^{-a}} \tag{2.20}$$

- Función Tangente hiperbólica:

$$f(a) = \frac{1 - e^{-2a}}{1 + e^{-2a}} \tag{2.21}$$

- Función ReLU o Unidad Lineal Rectificada. Esta función de activación es probablemente la que más se asemeja a su contraparte biológica. Es una mezcla de la función identidad y la función de umbral:

$$f(a) = \begin{cases} a, \ si \ a \geq 0 \\ 0, \ si \ a < 0 \end{cases} \tag{2.22}$$

Las funciones de activación más utilizadas son la sigmoidal logística, la tangente hiperbólica y la ReLU. Las tres funciones de activación difieren en los siguientes aspectos:

- Su rango es diferente.
- Sus derivadas se comportan de manera diferente durante el entrenamiento.

2.2.5. Validación del modelo MLP

Para validar el desempeño de una red MLP se hace una evaluación por medio de métricas que permitan cuantificar la capacidad de clasificación ante datos nuevos [40]. El análisis de métricas a partir de la matriz de confusion resultan útiles en casos de modelos de clasificación multiclase.

Por ello, la eficiencia del modelo proporciona una métrica que valida el desempeño de la red. La cual, se calcula a partir del promedio de distintos factores resultantes de k numero de pruebas. En cada prueba, se registran el numero de verdaderos positivos (VP), verdaderos negativos (VN), falsos positivos (FP) y falsos negativos (FN). A partir de estos se determinan tres métricas para calcular la eficiencia: exactitud, sensibilidad y especificidad. Las formulas de estas métricas según [61] son las siguientes:

$$Exactitud = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.23}$$

$$Sensibilidad = \frac{VP}{VP + FN} \tag{2.24}$$

$$Especificidad = \frac{VN}{VN + FP} \tag{2.25}$$

$$Eficiencia = \frac{Exactitud + Sensibilidad + Especificidad}{3} \tag{2.26}$$

Donde:

- VP: casos correctamente clasificados como positivos.

- VN: casos correctamente clasificados como negativos.
- FP: casos incorrectamente clasificados como positivos.
- FN: casos incorrectamente clasificados como negativos.

La exactitud nos proporciona un porcentaje toral de predicciones correctas con respecto al total de las pruebas. La sensibilidad solo nos proporciona un promedio de la capacidad del modelo para detectar correctamente casos positivos. Por otro lado, la especificidad muestra la capacidad del modelo para detectar correctamente casos negativos. Finalmente, la eficiencia se define como el promedio de estas tres métricas, lo que proporciona una valoración balanceada del desempeño del modelo.

2.3. Algoritmo genético

El algoritmo genético (GA, por sus siglas en inglés) es una técnica o algoritmo de optimización que está inspirada en el principio de la evolución natural. Se basa en los mecanismos de selección, reproducción y mutación [62]. Se empleó por primera vez en 1970 por John Holland al buscar resolver problemas complejos por medio de una estrategia adaptativa de la población [63]. Los GA trabajan sobre un grupo de soluciones potenciales, el cual se conoce como población. La cual se evalúa y evoluciona durante varias generaciones o iteraciones hasta encontrar la solución deseada o satisfactoria [62, 63].

Operadores genéticos El mecanismo evolutivo del GA se basa en los siguientes principios:

- Evaluación en función objetivo: a cada individuo del grupo de soluciones se le asigna un valor de aptitud que muestra qué tan buena es con respecto al problema planteado o función objetivo. La evaluación le indica al algoritmo qué soluciones ir seleccionando al ser las más aptas en las siguientes generaciones [62].
- Selección: se eligen qué individuos se van a reproducir. Las soluciones con mejor aptitud en la evaluación son las seleccionadas, lo que hace referencia a la "supervivencia del más apto"[62].
- Cruzamiento: combina los genes de dos individuos padres para formar nuevos individuos hijos que explorarán el espacio de búsqueda al ser nuevas combinaciones [62].
- Mutación: modifica aleatoriamente genes de un individuo, con respecto a una probabilidad. Esto introduce nuevas características genéticas en la población, que ayudan a abarcar mayor espacio de búsqueda y evita los óptimos locales [62].

Ciclo evolutivo El algoritmo genético funciona con este ciclo básico:

- Aleatoriamente, se genera una población inicial.
- Se evalúa la aptitud de cada individuo.
- Se seleccionan los individuos más aptos para el cruzamiento.
- Se aplican operadores de mutación para formar una nueva generación a partir del cruzamiento.

- Se repite el proceso hasta alcanzar un criterio de parada o hasta un número máximo de generaciones propuesto.

En la figura 2.5 se muestra un diagrama generar del ciclo evolutivo en el GA, donde i es la población inicial aleatoria, $f(x)$ es la evaluación en la función objetivo, Se es la selección de los individuos más aptos, Cr es el cruzamiento o reproducción de los individuos seleccionados, Mu es la mutación de los individuos hijos, y X^* es la solución más apta al final del ciclo.

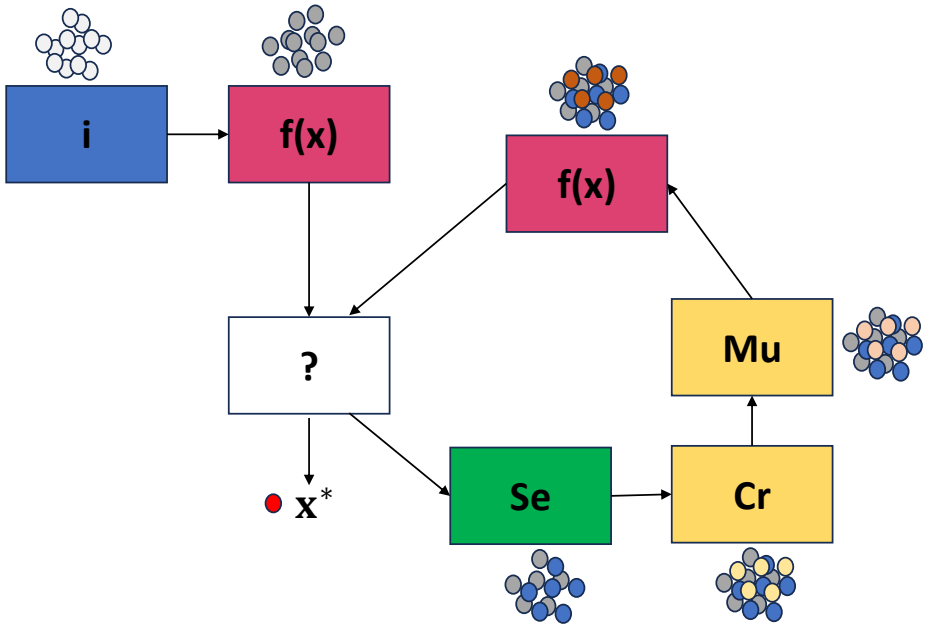


Figura 2.5: Diagrama de flujo genérico algoritmo genético.

2.4. FPGA

Las matrices de puertas lógicas programables en campo o *Field Programmable Gate Arrays* (FPGA) son circuitos integrados basados en bloques lógicos programables o bloques lógicos configurables (CLB, por sus siglas en inglés). Los cuales están conectados entre sí mediante interconexiones programables. En la figura 2.6 se muestra una vista general de la arquitectura de una FPGA [64].

Los CLB se componen de múltiples celdas que pueden emular cualquier compuerta lógica o combinación de estas. Estas celdas están conectadas mediante buses de interconexión configurables [64]. En la figura 2.7 se muestra una celda lógica general simplificada según el fabricante Xilinx®.

Donde la LUT o tabla de busqueda, se pueden considerarse una colección de celdas de SRAM las que forman tablas de verdad de n entradas. Esta arquitectura se utiliza para implementar cualquier función lógica combinacional que tenga un número n de variables. Las n entradas dependen del fabricante, pero comúnmente suelen ser 3, 4, 5 o 6, y estas proporcionan la dirección a la que se reflejara la salida de la LUT [65].

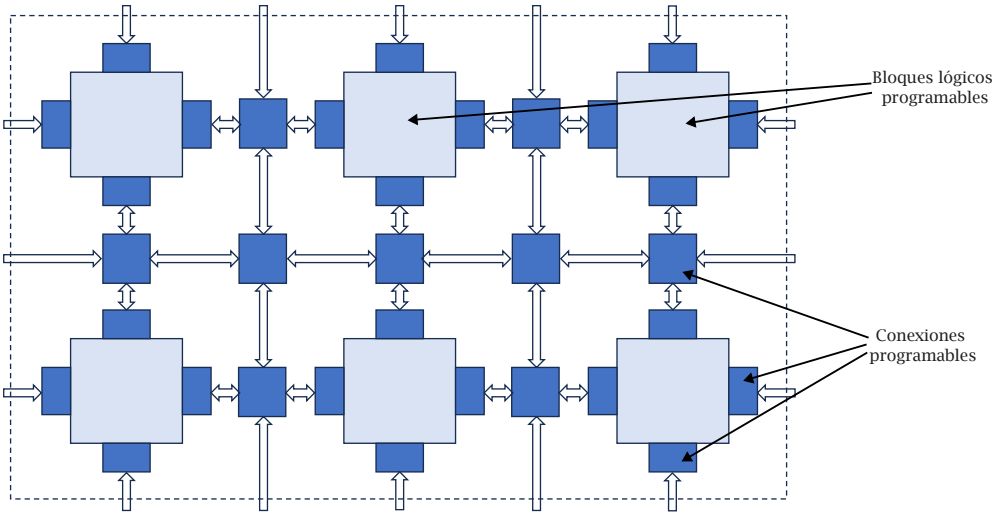


Figura 2.6: Vista superior simple de arquitectura general de FPGA.

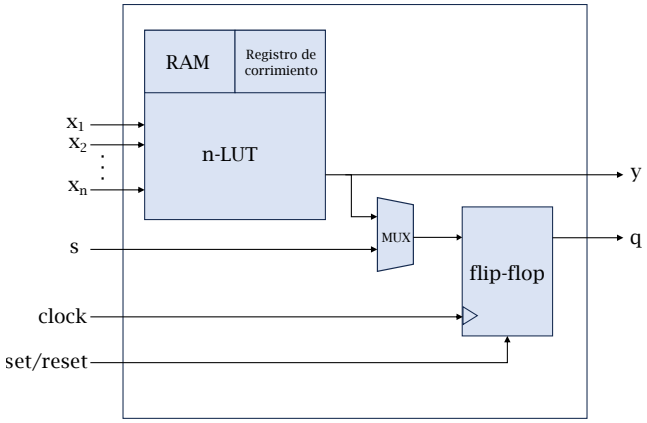


Figura 2.7: Celda lógica simplificada en FPGA Xilinx®.

La arquitectura de la FPGA también incluye bloques lógicos de entrada/salida que facilitan la comunicación con el entorno externo. Cuentan con pin de reloj y gestores de reloj, que proporcionan la sincronización en circuitos secuenciales. Además de bloques DSP para realizar funciones aritméticas con mayor facilidad y bloques de memoria RAM para almacenar miles de bits [64].

2.4.1. Diseño en FPGA

Los FPGA son una tecnología de cómputo reconfigurable, lo cual se refiere a un procesador que puede ser programado con un diseño y luego poder cambiarlo por otro de acuerdo a las necesidades del diseñador. Los FPGA permiten hacer diseños en paralelo dentro del mismo circuito y a nivel compuertas, por lo que estos pueden ser exportados a Circuitos Integrados de Aplicación Específica (ASIC, por sus siglas en inglés) convirtiéndose en diseños de hardware destinados a una aplicación específica [64].

Los ingenieros comúnmente utilizan lenguajes de descripción de hardware (HDL, por sus siglas en inglés) como VHDL o Verilog para hacer los diseños en FPGA, lo que permite la implementación

de algoritmos utilizando una metodología similar al de un diseño en software [66].

Circuitos Integrados de muy Alta Velocidad HDL

El lenguaje descriptivo de hardware Circuitos integrados de muy alta velocidad (VHSIC-HDL o VHDL, por sus siglas en inglés) es un lenguaje desarrollado por el Departamento de Defensa de los Estados Unidos en la década de los 80 [64]. Con el propósito de describir, simular y sintetizar circuitos digitales complejos mediante una representación textual [64, 66].

VHDL está diseñado para el modelado de sistemas concurrentes, como ocurre en circuitos digitales reales, a diferencia de lenguajes de programación tradicionales donde se modela un comportamiento secuencial de algoritmos [66]. Además, VHDL es un estándar abierto definido por el IEEE, lo cual le otorga compatibilidad con una amplia gama de herramientas de diseño[64]. Esto permite que los proyectos desarrollados en VHDL sean portables entre distintos entornos y fabricantes, siempre que se respete el cumplimiento del estándar.

En el contexto de esta tesis, la descripción del funcionamiento interno FPGA y el uso de VHDL sirven como base teórica para comprender cómo pueden implementarse algoritmos como un perceptrón multicapa en hardware reconfigurable. La traducción del modelo MLP a componentes de hardware tales como bloques lógicos configurables para los módulos de decisión en la red neuronal, memorias internas (BRAM) para almacenar los pesos y sesgos de la red, y unidades aritméticas (DSP) para realizar las operaciones de suma y multiplicación en las neuronas se detalla en el capitulo 3 correspondiente a la metodología, donde se aborda la implementación específica del clasificador en la FPGA.

CAPÍTULO 3

Metodología

En este capítulo se presentan los métodos y las etapas que se realizaron para el desarrollo del sistema de control de un actuador diseñado para simular movimientos de la mano, mediante la implementación de una red neuronal tipo MLP en un dispositivo FPGA.

El procesamiento de la base de datos de señales EMG se llevó a cabo por medio del software *MATLAB® R2020a* . La red neuronal inicialmente se construyó y entrenó en una laptop de 16 GB de memoria RAM, disco duro de 500 GB y procesador Intel® Core™ i7 de 12ª generación. Se utilizó *Jupyter Notebook 6.5.4* como entorno; empleando el lenguaje Python 3.12.6, para identificar las características e hiperparámetros requeridos para la clasificación.

El modelo entrenado fue posteriormente implementado en una tarjeta de desarrollo Basys 3®, que incorpora un FPGA Artix 7® de la marca Xilinx®. La laptop, programas y el FPGA seleccionado fueron utilizados para el logro de todos los objetivos. En la figura 3.1 se muestra el *graphical abstract* que representa el sistema que se desarrolló.

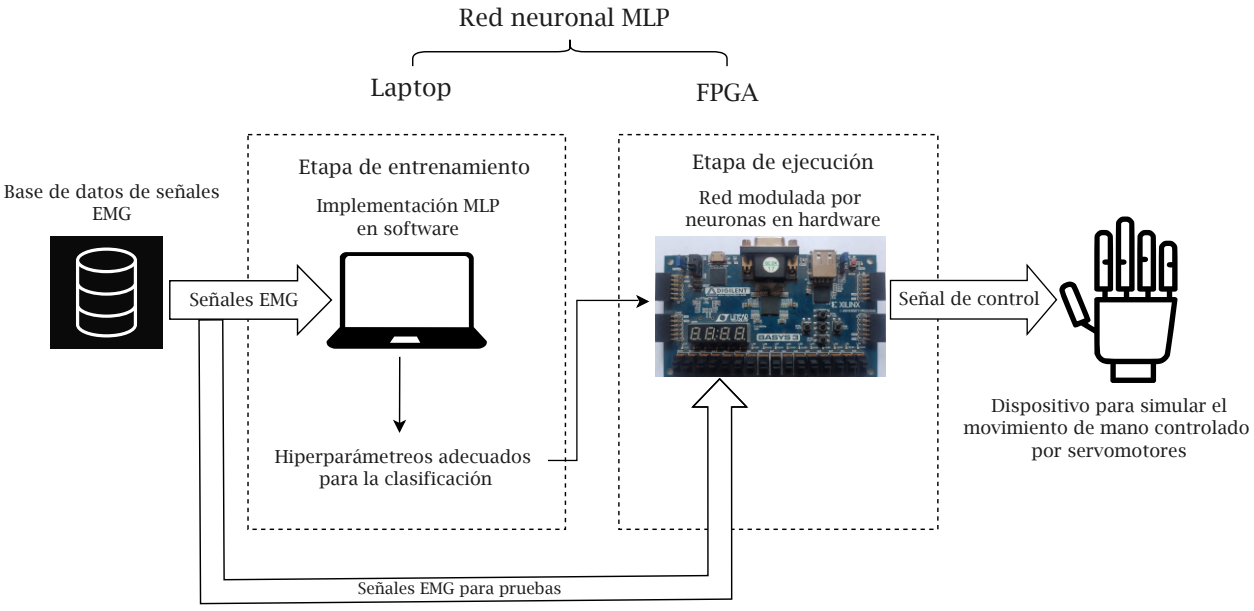


Figura 3.1: *Graphical abstract* del sistema.

La Figura 3.2 muestra el diagrama de flujo general de la metodología para llevar a cabo el trabajo.

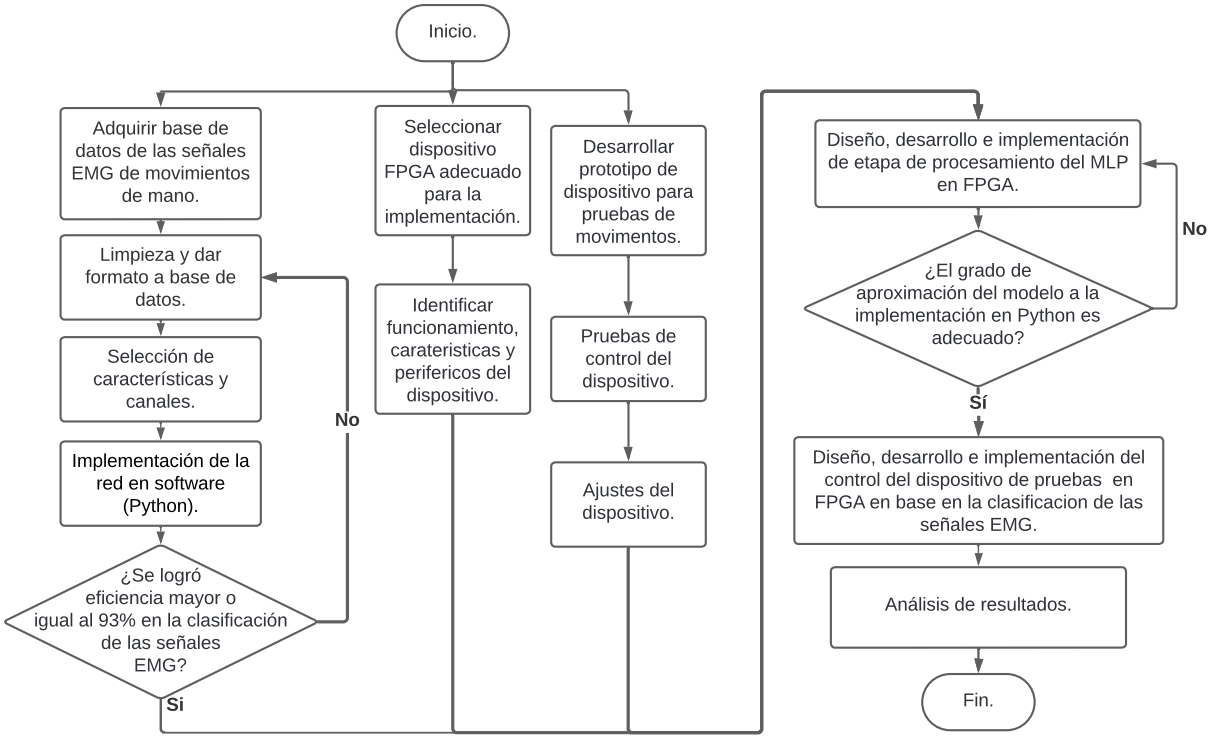


Figura 3.2: Diagrama de flujo para metodología general aplicada.

A continuación, se presentan a detalle las etapas del desarrollo del proyecto.

3.1. Base de datos de señales EMG

Inicialmente, se buscó una base de datos de registros de señales EMG con el propósito de utilizar dicha información para el entrenamiento y pruebas de la red MLP para clasificar. Se priorizó la selección de una base de datos de acceso público que incluyera gestos específicos de la mano, particularmente aquellos con movimiento de los dedos. Esto con la finalidad de que, una vez implementado el sistema de control, dichos gestos pudieran ser simulados posteriormente mediante el actuador desarrollado.

La base de datos que fue escogida contiene registros electromiográficos recopilados de los músculos de la muñeca y el antebrazo mientras se realizan gestos con la mano. La recopilación de datos se llevó a cabo con la participación de 43 individuos sanos (edad: 24-35 años) durante tres días distintos, mientras ejecutaban 16 gestos manuales y de los dedos en sesiones experimentales idénticas cada día. Para fines de este estudio solo se emplearon las muestras del primer día, descartando las muestras recopiladas en la sesiones dos y tres. En el experimento original, se colocaron 28 electrodos de superficie distribuidos sobre los principales músculos del antebrazo, utilizando una configuración monopolar con un electrodo de referencia en el codo. Las señales fueron adquiridas mediante

un sistema de registro multicanal *EMG-USB2+®* con una frecuencia de muestreo de 2048 Hz y ganancia de 500, además se aplicaron filtros analógicos pasa altas y pasa bajas a frecuencias de 10 Hz y 500 Hz para eliminar el ruido de movimiento y la interferencia eléctrica.

El protocolo de adquisición consistió en la ejecución de gestos de prensión y movimientos individuales de los dedos, realizados con la mano dominante en una posición anatómica neutral. Los participantes recibieron instrucciones visuales sobre el gesto a realizar y contaron con periodos de descanso entre repeticiones para evitar la fatiga muscular. La adquisición se realizó en condiciones controladas de temperatura y sin esfuerzo excesivo. Este protocolo garantiza la calidad de las señales y la repetibilidad de los gestos, permitiendo el uso de la base de datos para el entrenamiento de clasificadores de EMG.

La base de datos lleva por nombre: *Gesture Recognition and Biometrics ElectroMyogram* (GRABMyo) (version 1.0.2), Con identificación RRID: SCR_007345. Se encuentra disponible en [67]. Las características principales de esta se presentan en la tabla 3.1:

Tabla 3.1: Características base de datos EMG GRABMyo.

Tipo de movimientos	Manuales y de los dedos
Ubicación de electrodos	Muñeca y antebrazo
Número de electrodos	28 electrodos de superficie
Número de participantes	43 individuos sanos (24–35 años)
Número de gestos	16 gestos manuales y de los dedos, más reposo
Número de repeticiones	7 repeticiones por movimiento
Total de registros	429,828 señales
Frecuencia de muestreo	2048 Hz
Tiempo de captura	4 s

Los 16 gestos o movimientos que representan las señales EMG de la base de datos. Estos se muestran en la figura 3.3. Donde se observan diversas configuraciones de dedos y algunos gestos con desplazamientos de muñeca.

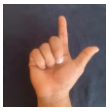
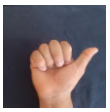
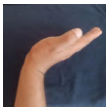
Gesto	Descripción	Gesto	Descripción	Gesto	Descripción	Gesto	Descripción
	Prehensión lateral		Extensión de los dedos pulgar e índice		Extensión del dedo índice		Rotación interna del antebrazo
	Aducción del pulgar		Extensión del pulgar y el meñique		Extensión del dedo pulgar		Rotación externa del antebrazo
	Oposición del pulgar y el meñique		Extensión de los dedos índice y medio		Extensión de muñeca		Mano abierta
	Oposición de los dedos pulgar e índice		Extensión del dedo meñique		Flexión de muñeca		Mano cerrada

Figura 3.3: Movimientos registrados en la base de datos.

Cabe señalar que, originalmente, la base de datos contenía 16 clases de movimientos y registros del estado en reposo que se consideraron como una clase adicional. Sin embargo, se eliminaron 6 de estas clases por no cumplir con los criterios definidos en este trabajo: algunas requerían movimientos de muñeca, y otras eran gestos repetitivos que no aportaban valor adicional al modelo. Por tanto, se conservaron únicamente los gestos correspondientes a movimientos de los dedos.

En la Figura 3.4 se presentan los 10 gestos seleccionados, junto con la etiqueta asignada a cada uno.

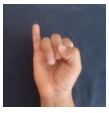
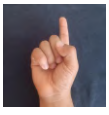
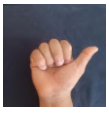
	Gesto	Descripción		Gesto	Descripción		Gesto	Descripción
0		Aducción del pulgar	4		Extensión de los dedos índice y medio	8		Mano cerrada
1		Oposición de los dedos pulgar e índice	5		Extensión del dedo meñique	9		Mano en reposo
2		Extensión de los dedos pulgar e índice	6		Extensión del dedo índice			
3		Extensión del pulgar y el meñique	7		Extensión del dedo pulgar			

Figura 3.4: Movimientos utilizados en el trabajo con su etiqueta numérica asignada.

Al conservar solo 10 movimientos y solo emplear los registros del primer día de las tres sesiones,

da un total de 84,280 señales crudas utilizadas en este trabajo, de las 429,828 disponibles en la base de datos original.

En la Figura 3.5 se muestra una representación gráfica de una de las señales EMG contenidas en la base de datos. Cada señal está compuesta por dos variables: amplitud y tiempo. Como puede observarse, las señales presentan un comportamiento complejo y de difícil interpretación de forma directa. Por ello, en etapas posteriores, se realiza un preprocesamiento para acondicionar la información antes de utilizarla en el entrenamiento y validación del clasificador.

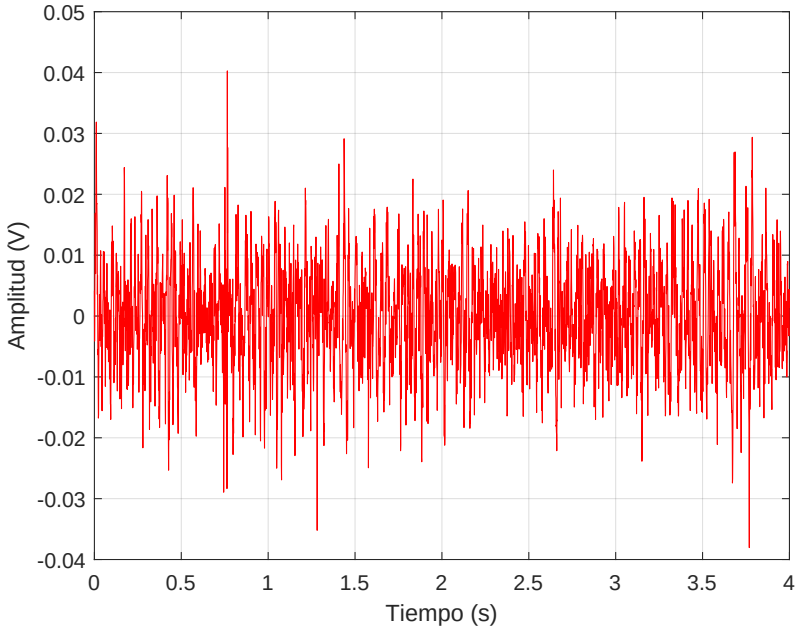


Figura 3.5: Muestra de señal EMG correspondiente al gesto de mano abierta.

3.1.1. Preprocesamiento de los datos

Las señales EMG crudas de la base de datos al ser complejas con difícil interperetacion, se le aplicaron varios etapas para acondicionar la información para ser exportada como una base de datos estructurada y etiquetada en un archivo Excel® con formato CSV. el procesamiento de las señales se realizo en *MATLAB*® por medio de funciones incorporadas en este software. a continuación se presentan las fases de procesamiento.

Cambio de tasa de muestreo

Inicialmente, las señales EMG contenidas en la base de datos estaban registradas a una tasa de muestreo de $2048Hz$. Sin embargo, para facilitar el procesamiento y reducir la carga computacional en el equipo de cómputo sin afectar la información relevante de las señales, se aplicó un resampleo o cambio de muestreo a 1000 Hz. Se optó por utilizar esta tasa de muestreo, ya que en la literatura de trabajos procesando este tipo de señales, como en [68] se emplea una tasa de 1 KHz, la cual resulta adecuada para representar la información útil de las señales electromiografías. Esto favorece la compatibilidad con otros trabajos de referencia y estándares de experimentos con buenos resultados al aplicar algoritmos de clasificación. Para realizar el cambio de muestreo se utilizó la función

resample(x,p,q) de *MATLAB®*, donde cambia la frecuencia de muestreo de la señal de entrada x, multiplicándola por la razón $\frac{p}{q}$. En este caso, la razón $\frac{1000}{2048}$ se aproximó racionalmente con la función *rat()*, resultando en $p = 125$ y $q = 256$, es decir, $\frac{p}{q} = \frac{125}{256}$. La función implementa internamente un filtro *antialiasing FIR* con ventana *Kaiser* para prevenir *aliasing* (contaminación de la señal) durante el proceso de cambio de muestreo. Dicho filtro presenta fase lineal, lo que garantiza que no se introduzca distorsión de fase en la señal remuestreada. Además, la función compensa automáticamente el retardo introducido por el filtro, manteniendo la alineación temporal entre la señal original y la señal resultante.

Filtrado de las señales

Después de realizar el cambio en la frecuencia de muestreo, se aplicó un filtro pasa bandas *Butterworth* de segundo orden, con frecuencias de corte de 10 Hz y 500 Hz, implementado mediante las funciones *butter* y *filtfilt* de *MATLAB®*. Esta última se empleó para lograr una respuesta de fase cero y evitar el desplazamiento temporal de la señal.

El propósito de este filtrado fue mantener la coherencia metodológica con trabajos previos y eliminar posibles componentes residuales fuera del rango de interés. No obstante, cabe señalar que las señales EMG originales ya habían sido prefiltradas durante la adquisición con un sistema *EMG-USB2+*, el cual incorpora un filtro pasa banda analógico entre 10 Hz y 500 Hz. Por ello, la aplicación del filtro digital no produjo cambios significativos en las señales.

De acuerdo con estudios previos [69, 70, 71], la mayor parte de la información útil en las señales electromiografías de superficie se encuentra dentro de este rango de frecuencias. Por lo que al realizar este filtrado dentro de esta banda, se asegura un mejor desempeño en el análisis de las señales en etapas posteriores. No se aplicó un filtro *notch*, dado que el filtrado analógico del dispositivo ya atenúa la interferencia de la red eléctrica (50/60 Hz).

Segmentación de las señales

Se aplica una segmentación o ventaneo superpuesto a cada muestra para aumentar el número de datos disponibles y para capturar con mejor precisión las transiciones entre contracciones musculares. Este enfoque de ventaneo superpuesto permite que cada nuevo segmento comience solo pocos milisegundos después del anterior y comparte un mayor número de puntos de la señal con los segmentos vecinos.

Previamente a este proceso de ventaneo, las muestras fueron divididas en los conjuntos de entrenamiento, validación y prueba, con el fin de evitar cualquier traslape de información entre ellos y asegurar una evaluación independiente del modelo.

Para este trabajo, se utilizaron ventanas de 250 ms con un empalme de 200 ms, por lo que implica, que cada 50 ms empieza un nuevo segmento. Esta técnica produce múltiples subconjuntos de datos a partir de cada muestra original, mejorando la robustez del modelo clasificador durante la etapa de entrenamiento al tener mayor número de datos.

No se aplicó un umbral de energía o detección de *EMG onset*, debido a que la base de datos utilizada ya se encuentra segmentada en periodos donde ocurren las contracciones musculares, presentando pocos intervalos de reposo.

Utilizando la fórmula 2.1 se obtuvieron que cada muestra fue dividida en 76 ventas nuevas, como se describe en 3.1. Donde cada ventana comparte la misma etiqueta de movimiento de la señal que se extrajo.

$$N = \left\lfloor \frac{4000 - 250}{250 - 200} \right\rfloor + 1 = \left\lfloor \frac{3750}{50} \right\rfloor + 1 = \lfloor 75 \rfloor + 1 = 76 \tag{3.1}$$

Al dividir cada señal en 76 segmentos, en esta etapa se tienen 6,405,280 registros EMG (84,280 registros x 76 ventanas).

Extracción de características a cada segmento

Después de la segmentación de las señales EMG, se procedió a cada nuevo segmento a la extracción de características. A cada segmento se le calcularon nueve características en el dominio del tiempo, las cuales representan una métrica que permite cuantificar varios aspectos de la señal. Estas métricas permiten identificar patrones como la forma, la amplitud, y variabilidad de cada segmento, lo que facilita la identificación de patrones discriminativos entre movimientos al aplicar un modelo de clasificación. En resumen, la extracción de características transforma cada señal en un conjunto reducido de valores numéricos representativos. En este trabajo, a cada segmento de señal EMG se le extrajeron nueve características. Dado que cada movimiento fue registrado simultáneamente por 28 sensores, se calcularon las mismas nueve características para cada sensor de forma individual. Como resultado, cada segmento quedó representado por un vector de 252 características (9 carateristicas x 28 sensores = 252) que describen un movimiento de la mano, teniendo en esta etapa, 228760 vectores que corresponden cada uno a una muestra EMG.

Este proceso de extracción de características se implementó mediante un algoritmo desarrollado en *MATLAB®*, que permitió automatizar el cálculo y organización de los vectores de características. La lista completa de las características extraídas se presenta en la Tabla 3.2.

Tabla 3.2: Características de señales EMG extraídas en este trabajo

Característica	**Fórmula matemática**
Valor absoluto medio	$MAV = \frac{1}{N}\sum_{k=1}^{N} \lvert x_k \rvert$
Cruce por cero (umbral = 0.01)	$ZC = \sum_{k=1}^{N-1} f_{ZC}(x_k)$
Cambio de signo de pendiente (umbral = 0.01)	$SSC = \sum_i f_{SS}(x_i)$
Longitud de forma de onda	$WL = \sum_{k=1}^{N} \lvert x_k - x_{k-1} \rvert$
Raíz cuadrada media	$RMS = \sqrt{\frac{1}{N}\sum_{k=1}^{N} x_k^2}$
Varianza	$VAR = \frac{1}{N-1}\sum_{k=1}^{N}(x_k - \bar{x})^2$
Integral EMG	$IEMG = \sum_{k=1}^{N} \lvert x_k \rvert$
Desviación estándar	$STD = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(x_k - \bar{x})^2}$
Amplitud Willison (umbral = 0.5 x STD)	$WAMP = \sum_{k=1}^{N-1} f(\lvert x_{k+1} - x_k \rvert)$

3.1.2. Exportación de base de datos limpiada y preprocesada

Una vez finalizado el preprocesamiento de las señales EMG, la información obtenida fue exportada en formato .CSV (Comma Separated Values), con el objetivo de facilitar su análisis posterior en el entorno de programación *Jupyter Notebook®* utilizando *Python versión 6.5.4*. Este formato permite organizar los datos de manera tabular, lo cual es compatible con múltiples bibliotecas de análisis y facilita su manipulación.

La base de datos resultante quedó estructurada y etiquetada, lista para ser utilizada en las siguientes etapas de clasificación. Las principales propiedades de la base de datos son:

- Estructura tabular con datos organizados por filas (muestras) y columnas (características).
- 252 características extraídas por segmento de señal.
- Un total de 228,760 muestras, 22,876 por clase.
- 10 clases o etiquetas, correspondientes a diferentes tipos de movimiento muscular.
- Datos no normalizados, es decir, en su escala original.
- Datos no homogéneos, dado que pueden variar en distribución y escala entre características.

Esta representación estructurada facilita el entrenamiento y evaluación del modelo de clasificación, al tiempo que permite aplicar técnicas estadísticas o de aprendizaje automático de forma eficiente. Cabe aclarar que el archivo .CSV se conserva sin normalizar con el fin de mantener la amplitud original de las señales; sin embargo, durante la fase de entrenamiento del modelo, los datos se normalizan mediante el uso de *StandardScaler*. El ajuste (*fit*) del normalizador se realiza exclusivamente con el conjunto de entrenamiento, y posteriormente la misma transformación se aplica a los conjuntos de validación y prueba.

3.2. Elección de entradas en el modelo de red MLP

Al exportar la base de datos, cada movimiento queda representado por un vector de 252 características, lo cual constituye una cantidad considerable de información, especialmente al ser utilizadas como entradas en el modelo de red neuronal MLP empleado en este trabajo. Debido a esta alta dimensionalidad, se consideró necesario identificar qué características aportan mayor información relevante al modelo, tanto en función del tipo de métrica extraída como del sensor de origen. Por lo que se optó por usar un algoritmo Genético de optimización, para determinar qué combinación de entradas en el modelo proporcionaba la mayor precisión en el modelo sin comprometer su rendimiento, y al mismo tiempo, mejorar la eficiencia computacional.

3.2.1. Reducción del número de características mediante algoritmo genético

Se optó a utilizar un algoritmo genético para reducir la dimensión del vector de entrada. Este algoritmo es una técnica de búsqueda y optimización que se inspira en el proceso de selección natural. En este caso, se utilizó para maximizar la precisión del modelo de clasificación por medio de identificar el subconjunto con menor número de características.

La elección del GA se debe a que, a diferencia de métodos lineales como PCA o LDA, permite explorar de manera más flexible combinaciones no lineales de características. Asimismo, frente a métodos basados en relevancia individual de características, como *Mutual Information*, *ReliefF* o *Boruta*, el GA ofrece la ventaja de optimizar de forma directa el desempeño del modelo final, considerando la interacción entre múltiples variables simultáneamente [63].

El objetivo del algoritmo fue maximizar la precisión (*accuracy*) del modelo MLP sobre el conjunto de validación con el menor número posible de características, buscando un equilibrio entre rendimiento y simplicidad del modelo. se utilizó una partición fija de los datos entre entrenamiento

y validación durante todo el proceso evolutivo, con el fin de asegurar que la evaluación de cada individuo fuera directamente comparable entre generaciones. No se aplicó estratificación en la división de los datos, dado que la base de datos contiene más de 200 mil muestras con una distribución de clases proporcional, lo que garantiza una representación equilibrada de las clases sin necesidad de aplicar este procedimiento adicional.

La búsqueda de reducción de características contribuye a disminuir la latencia y el consumo de recursos en la implementación final del modelo dentro del FPGA, donde la complejidad computacional debe mantenerse baja.

El GA se implementó en *Jupyter Notebook®* y utiliza una codificación binaria, donde cada individuo representa un conjunto específico de características seleccionadas (1 para seleccionada, 0 para descartada). La evaluación de cada individuo se realizó entrenando un modelo MLP y calculando su precisión sobre el conjunto de validación. Las pruebas se hicieron en un modelo MLP de prueba con dos capas ocultas con 100 neuronas cada una, funciones de activación en las capas ocultas Relu y softmax en la salida, y se utilizaron 5 épocas de entrenamiento.

A continuación, se muestra el pseudocódigo general del algoritmo:

Pseudocódigo 1 Pseudocódigo del algoritmo genético para la selección de características

Inicializar parámetros: generaciones $G = 25$, tamaño de población $\mu = 10$, desviación estándar de mutación $\sigma = 0.5$
Inicializar población padre X_p como una matriz de ceros de tamaño $(\mu \times 252)$
Inicializar población cruzada X_c y población mutada X_h con el mismo tamaño
Inicializar vectores de evaluación Y_p y Y_h como vectores de ceros de longitud μ
Inicializar lista vacía para almacenar la mejor solución
$X_p \leftarrow$ `POBLACION_INICIAL(`X_p`)`
for $i = 1$ to G **do**
    $X_c \leftarrow$ `OPERADOR_CRUCE(`X_p`)`
    $X_h \leftarrow$ `OPERADOR_MUTACION(`X_c`,` σ`)`
    $Y_p \leftarrow$ `OBJETIVO_V1(`X_p`)`
    $Y_h \leftarrow$ `OBJETIVO_V1(`X_h`)`
    $X_p \leftarrow$ `SELECCION_V1(`X_p`,` X_h`,` Y_p`,` Y_h`)`
    Almacenar `MAX(`Y_p`)` en la lista de mejores soluciones
end for
return mejor solución encontrada

Se utilizó un cruce de tipo uniforme, donde cada gen se obtiene del promedio entre dos genes padres de la población que son seleccionados de forma aleatoria. En este caso, para garantizar que todos los individuos participen en el cruce en cada generación, se asumió una probabilidad de cruce igual a 1.

El operador de mutación consistió en la adición de ruido gaussiano de media cero y desviación estándar $\sigma = 0.05$ a cada gen del individuo, seguido de una binarización donde los valores mayores o iguales a 0.5 se asignan a 1 y los menores a 0. Dado que la mutación se aplica sobre toda la población, la probabilidad de mutación es alta, mientras que su intensidad está determinada por el valor de σ.

Se implementó un esquema de preservación de élite, en el cual un hijo sustituye a su padre únicamente si alcanza una precisión superior en la función objetivo. Esto evita la pérdida de soluciones

prometedoras y acelera la convergencia del proceso.

El algoritmo se ejecutó con una población inicial de $\mu = 10$ individuos y $G = 25$ generaciones. El proceso es estocástico debido al uso de operadores aleatorios.

La gráfica de convergencia del algoritmo, mostrada en la Figura 3.6, muestra cómo la precisión del mejor individuo va variando mientras pasan las generaciones. Se observa una tendencia general de mejora en las primeras iteraciones, seguida de pequeñas fluctuaciones en generaciones posteriores. Estas variaciones se deben a que el proceso de evaluación incluye el entrenamiento de un modelo de red neuronal, cuyo desempeño puede presentar ligeras diferencias en cada ejecución, debido a factores como la aleatoriedad en la inicialización de pesos.

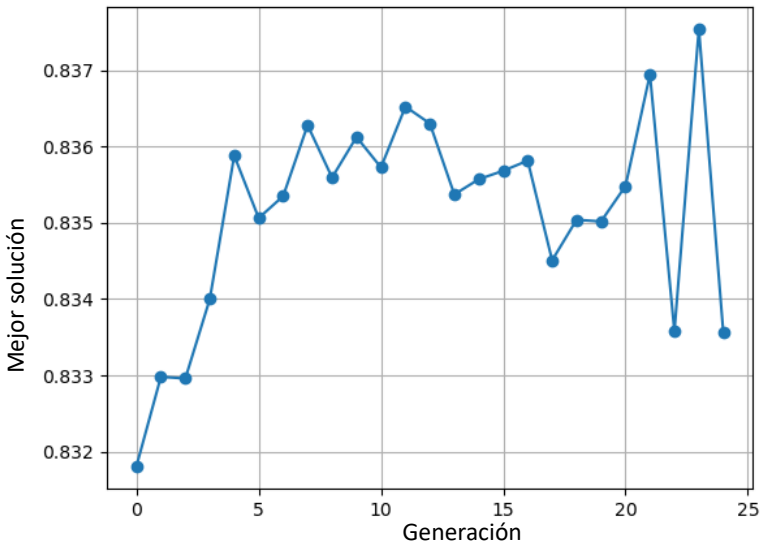


Figura 3.6: Gráfica convergencia de algoritmo genético.

Tras completar la ejecución del algoritmo genético, se obtuvo un subconjunto de características que presentó el mejor desempeño en términos de precisión de clasificación a lo largo de las 25 generaciones. Estas características fueron seleccionadas según su desempeño a lo largo de la ejecución del algoritmo, el cual evaluó distintas combinaciones de entradas.

La distribución de las 252 características originales se presenta en la Figura 3.7. En esta matriz visual, las filas representan las métricas extraídas, y las columnas corresponden a los sensores EMG numerados del 1 al 28. Las características seleccionadas por el algoritmo genético se encuentran marcadas, mientras que aquellas descartadas se muestran sin resaltar.

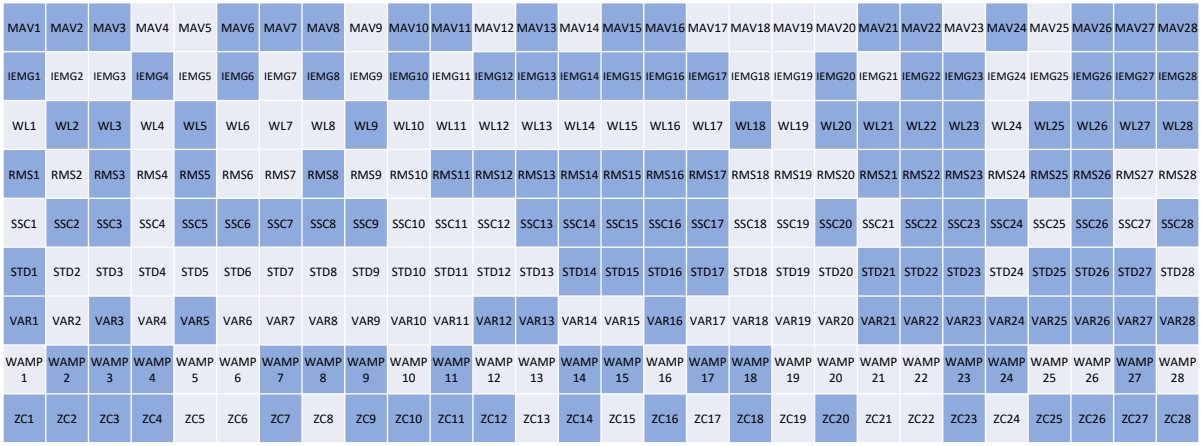


Figura 3.7: Características seleccionadas por el algoritmo genético.

El algoritmo seleccionó como mejor solución al final de las iteraciones a un individuo con 137 características de las 252 disponibles, alcanzando un valor de precisión de 93.51 % en datos de validación. Este subconjunto reducido fue empleado como entrada en el modelo MLP, logrando una precisión comparable a la obtenida utilizando el conjunto completo, pero con una reducción significativa en la dimensionalidad de entrada, lo cual favorece la eficiencia del sistema en términos de procesamiento y complejidad.

Dado que aún se conservaba una cantidad considerable de características y el modelo presenta una presión muy por arriba de la planteada en la hipótesis, se propuso una etapa adicional de reducción, la cual se detalla en la siguiente sección.

3.2.2. Reducción del número de características según la participación de sensores

Después de aplicar la primera etapa de selección utilizando el GA, se observó que aún era posible reducir la cantidad de características sin afectar significativamente la precisión del modelo. Para ello, se implementó una segunda estrategia de reducción, basada en la participación de los sensores en el subconjunto seleccionado previamente.

Esta etapa consistió en identificar los sensores con mayor contribución en el vector resultante del GA. Como criterio de filtrado, se descartaron todos los sensores que presentaban menos de seis métricas seleccionadas. De esta manera, se priorizó la información proveniente de los sensores con mayor relevancia en el proceso de clasificación.

La Figura 3.8 muestra la matriz de características, donde se resaltan las métricas seleccionadas de manera final para ser utilizadas como entradas en el modelo MLP.

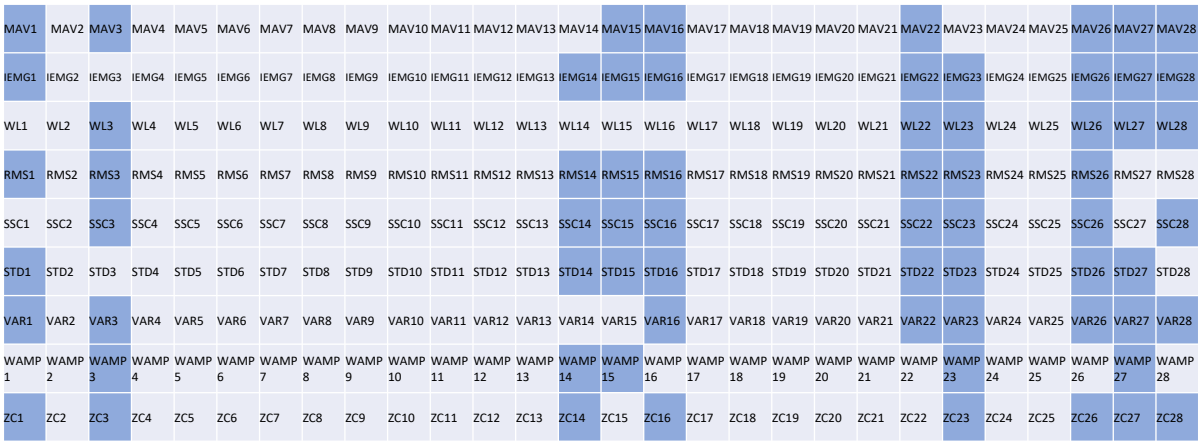


Figura 3.8: Selección final de características, considerando los sensores con mayor participación.

De acuerdo con este criterio, se conservaron únicamente las características correspondientes a los sensores 1, 3, 14, 15, 16, 22, 23, 26, 27 y 28, utilizando únicamente las métricas previamente seleccionadas por el algoritmo genético para cada uno de ellos.

Esta selección final resultó en un conjunto de 67 características de las 252 iniciales.

3.3. Modelo red neuronal en software

Una vez seleccionadas las características con mayor impacto para distinguir los diferentes movimientos musculares, se implementó el modelo de red neuronal MLP, incrementando el número de épocas de entrenamiento con el objetivo de obtener un modelo más robusto y preciso.

La implementación se realizó en el entorno de desarrollo *Jupyter Notebook®* con Python, empleando como entradas el vector de características reducido, obtenido mediante las técnicas de selección descritas en secciones previas. A las cuales, antes de ser usadas en el modelo, se les aplicó un proceso de normalización mediante la librería *StandardScaler* de la biblioteca *sklearn.preprocessing* con el propósito de normalizar la escala de los datos y mejorar los resultados en el proceso de aprendizaje. El ajuste (*fit*) del normalizador se realiza exclusivamente con el conjunto de entrenamiento, y posteriormente la misma transformación se aplica a los conjuntos de validación y prueba.

El modelo fue construido y entrenado utilizando la biblioteca *Keras* integrada en *TensorFlow®*, lo que facilita el uso de redes neuronales en este entorno, ya que son arquitecturas previamente diseñadas, a las cuales solo se definen su configuración.

La arquitectura final del modelo fue definida tras realizar múltiples pruebas con distintas configuraciones, tomando como referencia modelos del estado del arte en clasificación de señales EMG, como los criterios propuestos en [14]. La configuración elegida mostró un equilibrio adecuado entre precisión y complejidad computacional, siendo además viable para su posterior implementación en hardware embebido.

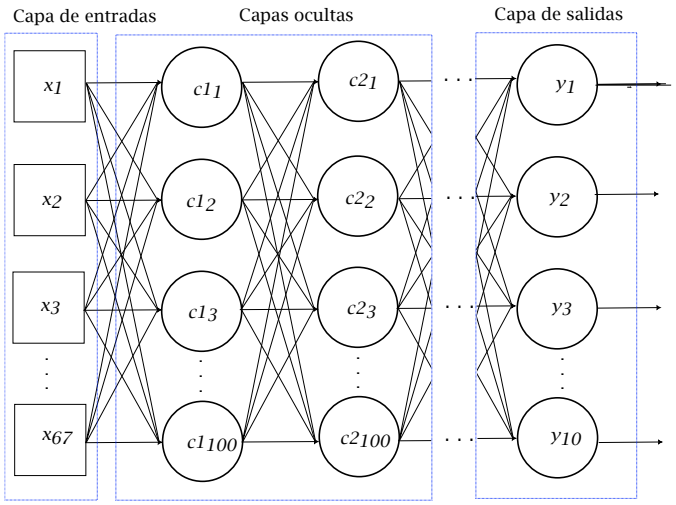


Figura 3.9: Modelo red neuronal perceptrón multicapa.

Las características del modelo se muestran en la tabla 3.3:

Tabla 3.3: Configuración de la red neuronal utilizada.

Neuronas capa de entrada	67
Capas ocultas	2
Neuronas por capa	100
Funciones de activación	`ReLU` en capas ocultas y `Softmax` en capa de salida
Neuronas capa de salida	10
Optimizador	`Adam`
Tasa de aprendizaje (LR)	0.001
Función de pérdida	`sparse_categorical_crossentropy`
Métrica de evaluación	`Accuracy`
Inicialización de pesos	Por defecto de *Keras* (Glorot Uniform)
Regularización	Restricción de norma máxima = 3
Épocas de entrenamiento	30
Tamaño de lote	20
Datos de entrenamiento	80 %
Datos de validación	10 %
Datos de prueba	10 %
Semilla aleatoria	42 (para reproducibilidad)
Estrategia de barajado	Aleatoria sin estratificación
Callback utilizado	`ModelCheckpoint` (almacenamiento de pesos durante el entrenamiento)

Se eligió la función de activación ReLU para las capas ocultas por su simplicidad y bajo costo

computacional, lo cual facilita su futura implementación en hardware digital. Por su parte, la capa de salida utiliza la función Softmax, que permite interpretar las salidas como una distribución de probabilidad sobre las 10 clases, indicando el grado de certeza del modelo respecto a cada clase posible. Cabe señalar que la función Softmax no fue implementada en hardware, ya que en dicha etapa no es necesario obtener la distribución completa de probabilidades sobre las clases. Basta con identificar la salida con el valor más alto para determinar el gesto predicho por la red. Por ello, en la implementación en hardware se sustituyó Softmax por la operación Argmax, la cual selecciona directamente la neurona con el mayor valor de activación en la capa de salida.

Durante el entrenamiento, el modelo alcanzó una precisión de clasificación del **96.19 %** con los datos de validación en el entrenamiento y **95.94** con los datos de pruebas. Teniendo un comportamiento estable del error de entrenamiento, como se observa en la Figura 3.10. Esta precisión fue considerada adecuada para ser tomada como base para la implementación posterior en hardware.

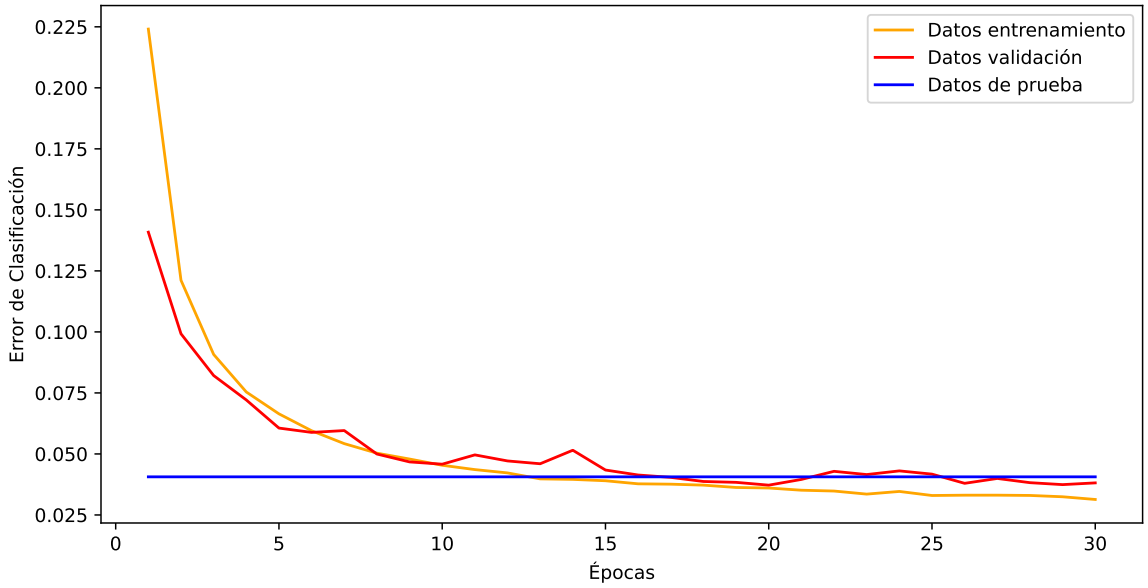


Figura 3.10: Gráficas de error en la clasificación.

3.4. Exportación de parámetros del modelo MLP para implementación en hardware

Para poder realizar la inferencia del modelo MLP en hardware utilizando un dispositivo FPGA, fue necesario adaptar y exportar los parámetros del modelo entrenado, específicamente, los pesos, sesgos y datos de prueba que funcionen como entradas en el modelo. Estos elementos se convierten en un formato compatible con la arquitectura del diseño digital. Este proceso involucró varias etapas, como la extracción de los pesos y sesgos del modelo entrenado en software, la conversión binaria en representación a dos con punto fijo y la codificación final en archivos compatibles para su almacenamiento y uso dentro de las memorias ROM del FPGA.

3.4.1. Extracción de pesos y sesgos desde el modelo entrenado

Después de completar el modelo en software con los resultados requeridos, se extrajeron los pesos y sesgos de cada capa utilizando las funciones disponibles en la biblioteca *Keras* de *TensorFlow®*. Estos valores se almacenaron inicialmente en archivos CSV utilizando la biblioteca *pandas*, lo que facilitó su análisis y procesamiento posterior.

3.4.2. Análisis de rangos y decisión del formato de punto fijo

Para determinar la representación binaria más adecuada para el diseño en hardware, se realizó un análisis de los pesos y sesgos, al identificar los valores mínimos y máximos presentes en cada capa del modelo. La tabla 3.4 muestra estos rangos de valores de cada capa. Al analizar los valores, es posible interpretar la parte entera con pocos bits al tratarse de valores relativamente pequeños; aun así, se optó por utilizar formatos más amplios para conservar una precisión adecuada en la etapa de inferencia en hardware.

Tabla 3.4: Rangos de valores mínimos y máximos de pesos y sesgos por capa.

Min capa oculta 1	**Max capa oculta 1**	**Min capa oculta 2**	**Max capa oculta 2**	**Min capa salida**	**Max capa salida**
-2.02119	1.81856	-1.67922	2.67355	-1.39727	0.76537

Adicionalmente, los valores de entrada correspondientes a las muestras de prueba también se almacenaron en memoria ROM del FPGA, por lo que se optó por utilizar el mismo formato de representación que los pesos. Dado que estas entradas fueron previamente normalizadas, sus valores se mantienen en un rango reducido, justificando el uso de una representación con pocos bits enteros.

Con base en este análisis se definieron los siguientes formatos de la representación numérica en complemento a dos con punto fijo:

- Pesos: representados en formato 8e.8f (8 bits para parte entera, 8 bits para parte fraccionaria), con un total de 16 bits por valor.
- Sesgos: representados en formato 16e.16f, ocupando 32 bits por valor, ya que deben sumarse al resultado de productos entre valores de 16 bits, lo que incrementa el rango y la precisión requeridos.
- Valores de entradas para pruebas: representados en formato 8e.8f, ocupando 16 bits por valor.

3.4.3. Conversión a binario y generación de archivos .COE

Una vez los valores de pesos, sesgos y vectores de entradas fueron cuantizados a formato punto fijo, se realizo la conversión binaria en complemento a dos. Este proceso fue automatizado mediante algoritmos en *Python* que hacia la conversion de los valores desde los archivos CSV a vectores binarios de longitud fija.

Posteriormente, los vectores binarios que representaban los parámetros del modelo fueron convertidos a formato .COE (*Coefficient File*), el cual es compatible con el entorno *Xilinx Vivado®* para su uso en memorias ROM dentro del FPGA.

3.5. Modelo red neuronal en hardware

Una vez establecidos los hiperparámetros (número de capas, número de neuronas, funciones de activación, pesos y sesgos) del modelo entrenado en software. Al igual, que la exportación en una representación binaria de los pesos y sesgos, para su almacenamiento en memorias ROM. Se diseñó la arquitectura digital para la ejecución de la inferencia del modelo MLP directamente en hardware. Se utilizó una tarjeta *Basys 3®* con un FPGA de la familia *Xilinix Artix-7®*.

El diseño fue desarrollado en el entorno *Vivado 2021.1®*, empleando el lenguaje de descripción de hardware VHDL para desarrollar los módulos que componen la arquitectura. También, se empleó el IP Core prediseñado para configurar y utilizar memorias ROM del FPGA por medio de la herramienta *Generate Memory*.

La arquitectura general está basada en composición modular para el funcionamiento del sistema, donde cada bloque o módulo cumple una tarea específica para lograr la inferencia del modelo. Algunas de las tareas son lectura de datos, multiplicación, acumulación, control de flujo, activación y almacenamiento. Para coordinar el flujo de los datos entre los bloques se realiza mediante una máquina de estados finitos (FSM, por sus siglas en inglés). El diagrama general del sistema de red neuronal se muestra en la Figura 3.11.

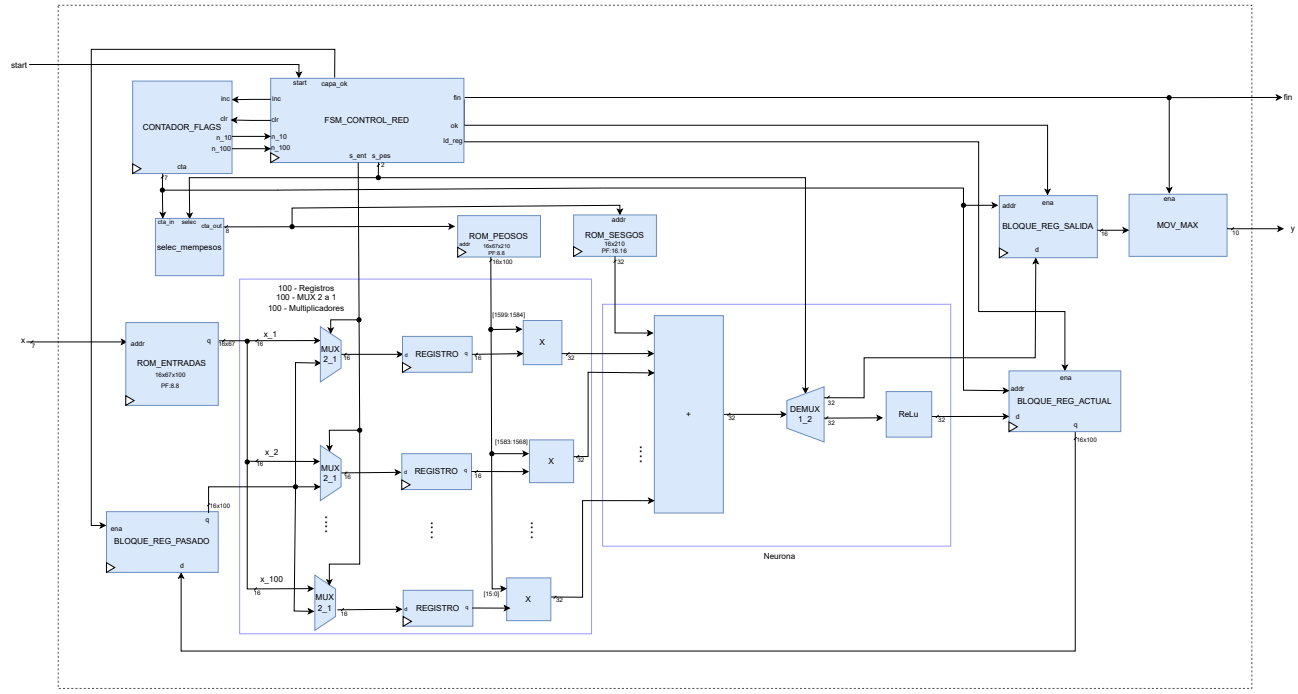


Figura 3.11: Diagrama de bloques de red neuronal en hardware.

A continuación, se describen las principales unidades de hardware utilizadas.

3.5.1. Módulo principal de la red MLP

El módulo principal integra todos los bloques de hardware, formando la unidad superior diseñada para la inferencia del modelo de red neuronal ejecutada en el FPGA. Este módulo funciona como

un único circuito que coordina el direccionamiento y lectura de datos, ejecución de operaciones aritméticas y de codificación, y finalmente la entrega del resultado de clasificación.

Con el fin de facilitar la integración con otros sistemas digitales, el módulo está diseñado con una interconexión simple de control, ya que está compuesta de únicamente dos señales de entrada y dos señales de salida, además de la señal de reloj y la señal de reinicio. La Figura 3.12 muestra el diagrama de caja negra de la arquitectura de la red neuronal en hardware, y la Tabla 3.5 detalla las señales de entrada/salida del módulo.

Tabla 3.5: Entradas y salidas del módulo principal de la red neuronal MLP.

Nombre	**Tamaño**	**Tipo**	**Descripción**
`clk`	1 bit	Entrada	Señal de reloj del sistema, con frecuencia de 25 MHz, utilizada para sincronizar todas las operaciones internas del módulo.
`rst`	1 bit	Entrada	Señal de reinicio. Al activarse, inicializa el sistema, reiniciando registros, contador y la máquina de estados finitos.
`start`	1 bit	Entrada	Señal de control que inicia el proceso de inferencia. Se activa por un ciclo de reloj cuando se desea clasificar una nueva muestra.
`x`	7 bits	Entrada	Dirección de acceso a la memoria ROM de entradas EMG.
`fin`	1 bit	Salida	Señal que indica la finalización del proceso de inferencia por parte del modelo MLP. Se activa cuando la clasificación está completa.
`y`	10 bits	Salida	Vector donde un único bit en alto ('1') representa la clase predicha por la red neuronal. Cada posición del vector está asociada a una clase distinta.

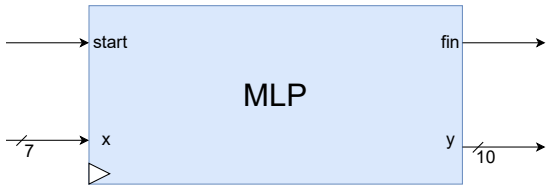


Figura 3.12: Diagrama de caja negra del módulo principal de la red neuronal MLP.

El diseño modular de la red permite que todas las operaciones funcionen internamente, de forma secuencial y ordenada, sin requerir intervención externa durante la clasificación. El flujo de datos es controlado por una máquina de estados finitos que se describe a continuación.

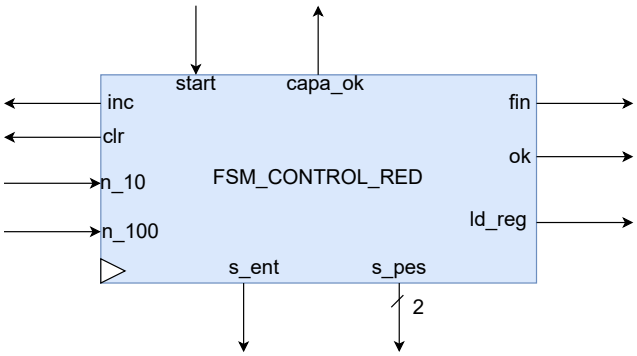


Figura 3.13: Diagrama de caja negra del módulo de máquina de estados finitos.

3.5.2. Módulo máquina de estados finitos para controlar el flujo de datos en la red MLP

Este módulo FSM es responsable de la sincronización y coordinación del funcionamiento de todos los bloques de hardware de la arquitectura para la inferencia del modelo de red neuronal MLP dentro del FPGA. Esta máquina de estados gestiona el flujo de datos de las memorias ROM, unidades aritméticas, registros, contador y bloques de activación, lo que asegura que cada operación se realice en el orden y ubicación adecuados.

La FSM diseñada es de tipo *Moore*, ya que sus salidas dependen únicamente del estado actual, no de las entradas. La elección de este tipo de máquina de estados permite un control estable de las señales internas, ya que se tiene un ciclo de reloj adicional en la transición de estados para verse reflejadas las salidas, lo que facilita la implementación y verificación en el diseño de hardware digital.

Durante la ejecución de la inferencia del modelo, la FSM realiza:

- Inicia la lectura de las muestras EMG desde la memoria de entrada.
- Activa la lectura secuencial de pesos y sesgos desde sus respectivas ROMs, según la capa actual.
- Controla la carga de datos en los registros.
- Coordina la multiplicación y acumulación de productos en cada neurona.
- Determina el tipo de activación en las neuronas y de almacenar sus salidas.
- Señaliza el cambio de capa y el fin del proceso de clasificación.

La Figura 3.13 presenta el diagrama de caja negra del módulo FSM, mientras que la Figura 3.14 muestra el grafo de estados definidos en el proceso.

La Tabla 3.6 detalla las señales de entrada y salida utilizadas por el módulo FSM:

Entradas	
clk	1 bit
rst	1 bit
start	1 bit
n_10	1 bit
n_100	1 bit

Salidas	
inc	1 bit
clr	1 bit
ld_reg	1 bit
s_ent	1 bit
s_pes	2 bit
capa_ok	1 bit
ok	1 bit
s_fin	1 bit

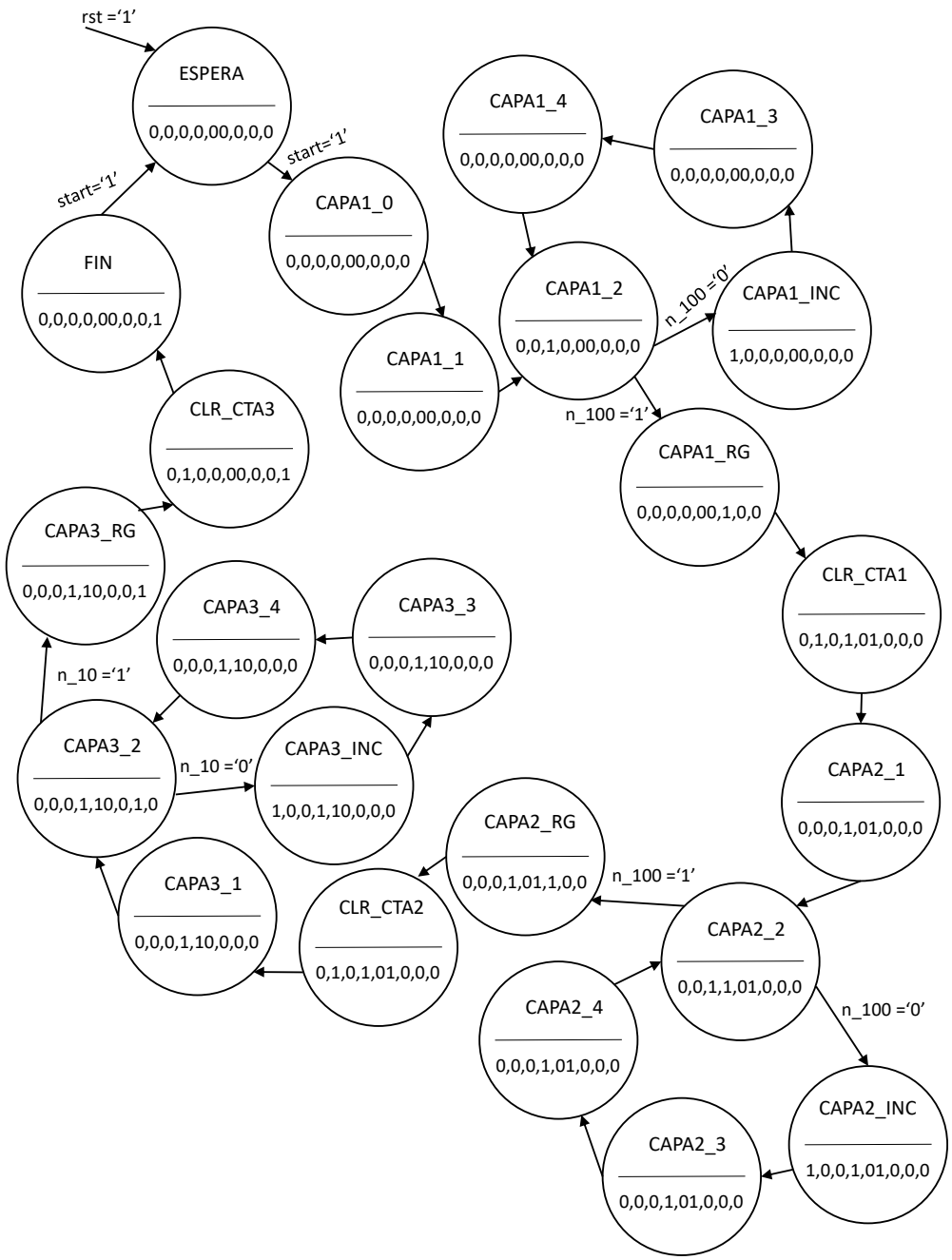


Figura 3.14: Grafo de estados de la FSM para el control del flujo de datos del modelo MLP.

Tabla 3.6: Entradas y salidas del módulo de la máquina de estados finitos.

Nombre	Tamaño	Tipo	Descripción
clk	1 bit	Entrada	Señal de reloj del sistema (25 MHz), utilizada para sincronizar las transiciones entre estados y controlar las operaciones internas.
rst	1 bit	Entrada	Señal de reinicio asíncrono. Al activarse, reinicia la FSM al estado inicial y pone en cero las señales de control.
start	1 bit	Entrada	Señal que indica el inicio del proceso de inferencia. Se activa por un solo ciclo de reloj.
s_ent	1 bit	Salida	Selecciona si las entradas provienen de la memoria ROM de muestras EMG o del registro de salidas de la capa anterior, dependiendo de la capa en curso.
s_pes	1 bit	Salida	Señal que informa al codificador de direcciones qué capa está activa, para acceder a los pesos y sesgos correspondientes.
inc	1 bit	Salida	Activa el incremento del contador de neuronas, permitiendo pasar a la siguiente neurona en la capa actual.
clr	1 bit	Salida	Limpia el valor del contador de neuronas al finalizar una capa.
n_100	1 bit	Entrada	Bandera proveniente del contador que indica que se ha procesado la neurona número 100 (capas ocultas).
n_10	1 bit	Entrada	Bandera proveniente del contador que indica que se ha procesado la neurona número 10 (capa de salida).
ld_reg	1 bit	Salida	Señal que indica que el resultado de una neurona ya fue calculado y puede almacenarse en los registros de salida.
capa_ok	1 bit	Salida	Señal que indica que todas las neuronas de la capa actual han sido procesadas. Activa la transferencia de resultados al siguiente bloque.
ok	1 bit	Salida	Indica que todas las salidas de la capa final han sido almacenadas y están listas para el bloque de decisión.
fin	1 bit	Salida	Señal de finalización global. Se activa cuando el proceso completo de inferencia ha concluido y la clase ha sido determinada.

3.5.3. Módulos de memoria ROM

Para el almacenamiento de los datos necesarios durante la inferencia del modelo MLP en el FPGA, se utilizaron tres bloques de memoria ROM, cada uno configurado y generado mediante la herramienta *Generate Memory IP Core* incluidada en el entorno de desarrollo *Vivado 2021.1®*. A cada módulo se le cargaron archivos .coe generados a partir del modelo entrenado en software, así como de las muestras EMG utilizadas para pruebas.

La Figura 3.15 muestra el diagrama de caja negra de las tres memorias ROM utilizadas en el sistema.

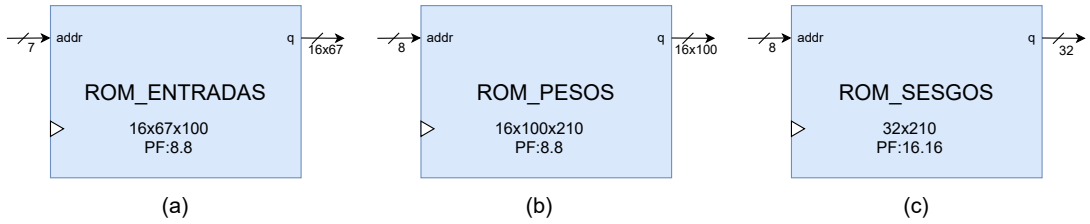


Figura 3.15: Diagrama de cajas negras de las memorias ROM. (a) ROM de entradas. (b) ROM de pesos. (c) ROM de sesgos.

A continuación, se describen las características de cada una:

Memoria ROM de entradas

Esta memoria contiene las muestras EMG utilizadas como entradas para el modelo MLP. Cada muestra está compuesta por 67 características, cada una codificada en 16 bits con formato punto fijo 8 bits parte entera y 8 bits parte fraccionaria. La memoria permite almacenar hasta 128 muestras pero se estableció en 100 muestras para pruebas.

- Entrada `addr`: 7 bits. Direcciona una de las 100 muestras almacenadas.
- Salida `q`: Vector de 16×67 bits. Contiene todas las características de una muestra.

Memoria ROM de pesos

Esta memoria contiene los pesos sinápticos del modelo MLP. La red neuronal cuenta con 210 neuronas en total (100 en la primera capa oculta, 100 en la segunda y 10 en la salida), y cada neurona utiliza 100 pesos. Cada peso se representa con 16 bits en formato punto fijo 8 bits parte entera y 8 bits parte fraccionaria.

- Entrada `addr`: 8 bits. Permite direccionar una neurona específica.
- Salida `q`: Vector de 16×100 bits. Contiene los 100 pesos asociados a la neurona seleccionada.

Memoria ROM de sesgos

Esta memoria almacena los 210 sesgos asociados a cada neurona de la red. Cada sesgo es codificado con 32 bits utilizando formato punto fijo 16 bits parte entera y 16 bits parte fraccionaria, para mantener una alta precisión en las operaciones de suma acumulativa.

- Entrada `addr`: 8 bits. Direcciona uno de los 210 sesgos.
- Salida `q`: 32 bits. Valor del sesgo correspondiente a la neurona direccionada.

3.5.4. Módulo contador con banderas

Este módulo fue diseñado como un contador síncrono ascendente, su función es llevar el control de la cantidad de neuronas procesadas en cada capa de la red durante la inferencia en hardware, como el proceso se divide por neurona procesada y por capas, fue necesario tener un conteo de la neurona presente. El contador se activa por medio de señales generadas por la FSM que controlan el incremento de la cuenta y su restablecimiento en cero automáticamente al completar el procesamiento de una capa.

Cuenta con una salida principal de 7 bits que representa el valor de la cuenta actual. Esta cantidad de bits es suficiente para cubrir las 100 neuronas de las capas ocultas. Además, este módulo fue diseñado para incorporar dos banderas como señales de salida que se activan al alcanzar umbrales específicos en la cuenta, lo que permite sincronizar los eventos del flujo en la FSM. La Figura 3.16 presenta el diagrama de caja negra del módulo contador con banderas.

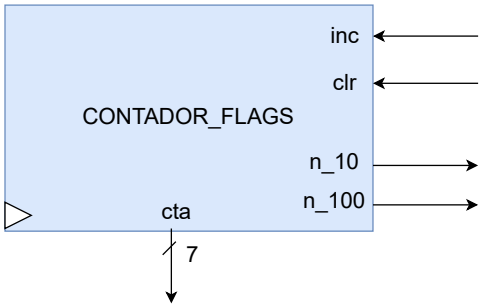


Figura 3.16: Diagrama de caja negra del módulo contador con banderas.

La Tabla 3.7 detalla las señales de entrada y salida utilizadas por el módulo contador.

Tabla 3.7: Entradas y salidas del módulo contador con banderas.

Nombre	Tamaño	Tipo	Descripción
`clk`	1 bit	Entrada	Señal de reloj del sistema, con una frecuencia de 25 MHz. Sincroniza el incremento y el restablecimiento del contador.
`rst`	1 bit	Entrada	Señal de reinicio asíncrono que reinicia el contador a cero, sin depender del ciclo de reloj.
`inc`	1 bit	Entrada	Señal de habilitación para incrementar el valor del contador en uno, en el flanco positivo del reloj.

Continúa en la siguiente página

Tabla 3.7 – continuación de la página anterior

Nombre	Tamaño	Tipo	Descripción
`clr`	1 bit	Entrada	Señal de limpieza síncrona. Al activarse, restablece el contador a cero en el siguiente ciclo de reloj.
`cta`	7 bits	Salida	Valor actual del contador, representado como un número binario de 7 bits.
`n_10`	1 bit	Salida	Bandera que se activa cuando el valor del contador es igual a 10. Se utiliza para identificar el final de la capa de salida.
`n_100`	1 bit	Salida	Bandera que se activa cuando el valor del contador es igual a 100. Se utiliza para identificar el final de una capa oculta.

3.5.5. Módulo Codificador de dirección de memoria para pesos

El módulo codificador de dirección de memoria de pesos, nombrado como *selec_mempesos* en el diseño, tiene como objetivo generar la dirección correcta para acceder a la memoria ROM que contiene los pesos de todas las neuronas del modelo MLP. Dado que los pesos de las 210 neuronas están almacenados de forma continua en una única memoria ROM, es necesario ajustar la cuenta local de cada capa para que apunte al bloque correspondiente dentro de la memoria global.

Este módulo recibe como entrada la cuenta local de neuronas, generada por el contador, y una señal de selección de capa de 2 bits proveniente de la FSM, que indica en qué capa del modelo se encuentra actualmente el proceso de inferencia.

Con base en estos valores, el codificador realiza un desplazamiento del valor de cuenta del contador según la capa activa, sumando un desplazamiento predefinido para cada capa:

- Capa oculta 1: desplazamiento 0.
- Capa oculta 2: desplazamiento 100.
- Capa de salida: desplazamiento 200.

El resultado se entrega en la salida, una dirección de 8 bits que se utiliza para acceder a la memoria ROM de pesos. Se optó por utilizar este codificador del direccionamiento de los pesos para evitar el uso de un segundo contador, ya que la cuenta local también direcciona los registros de almacenamiento de los resultados de las neuronas de la capa activa.

La Figura 3.18 muestra el diagrama de caja negra del módulo, mientras que la Tabla 3.8 describe sus señales de entrada y salida.

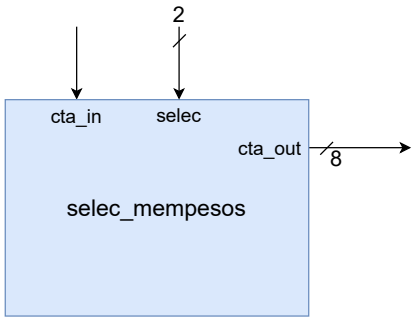


Figura 3.17: Diagrama de caja negra del codificador de dirección de memoria de pesos.

Tabla 3.8: Entradas y salidas del módulo codificador de dirección de pesos.

Nombre	**Tamaño**	**Tipo**	**Descripción**
`cta_in`	7 bits	Entrada	Cuenta local que indica la neurona actual en la capa activa. Proviene del módulo contador.
`selec`	2 bits	Entrada	Señal de selección de capa. Codifica la etapa del modelo en la que se encuentra el proceso: `00` para la primera capa oculta, `01` para la segunda capa oculta, y `10` para la capa de salida.
`cta_out`	8 bits	Salida	Dirección ajustada que se utiliza para acceder a la memoria ROM de pesos. Esta dirección corresponde al índice absoluto dentro del bloque completo de pesos y permite el direccionamiento secuencial de neuronas a lo largo de todas las capas.

3.5.6. Módulo multiplexor para selección de entradas a las neuronas

Este módulo cumple una función clave en la etapa de propagación hacia adelante de la red neuronal MLP implementada en hardware. Su propósito es seleccionar, en cada etapa del proceso, cuál conjunto de datos se utilizará como entrada para las neuronas: ya sea el vector de características EMG provenientes de la memoria ROM de entrada, o los resultados intermedios de cada neurona después de su función de activación (en este caso la función ReLu) que fueron almacenados en los registros correspondientes a la capa anterior.

La función de selección se realiza mediante una señal de control proveniente de la FSM, la cual indica si el sistema se encuentra procesando la capa de entrada o una capa intermedia. Este módulo, por su simplicidad, se implementa como un multiplexor dos a uno, donde las dos entradas son de 16 bits.

La Figura 3.18 muestra el diagrama de caja negra del módulo, y la Tabla 3.9 detalla sus señales de entrada y salida.

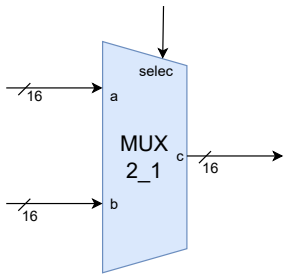


Figura 3.18: Diagrama de caja negra del codificador de dirección de memoria de pesos.

Tabla 3.9: Entradas y salidas del módulo multiplexor.

Nombre	Tamaño	Tipo	Descripción
`a`	16 bits	Entrada	Vector proveniente de la memoria ROM que contiene las características de entrada EMG. Es utilizado únicamente en la primera capa de la red neuronal.
`b`	16 bits	Entrada	Valor proveniente del bloque de registros que contiene las salidas de activación de la capa anterior. Es utilizado en capas ocultas y de salida.
`selec`	1 bit	Entrada	Señal de control proveniente de la FSM. Si su valor es '0', la salida toma el valor de `a`; si es '1', toma el valor de `b`.
`c`	16 bits	Salida	Valor de salida del multiplexor. Corresponde al valor seleccionado entre las dos entradas en función de la señal `selec`.

3.5.7. Módulos aritméticos

En el diseño en hardware implementado para la red neuronal, se emplean tres módulos encargados de realizar operaciones matemáticas fundamentales: el sumador, multiplicador y el módulo de activación ReLu. Estos módulos fueron diseñados en VHDL empleando las bibliotecas estándar de IEEE, específicamente la biblioteca *IEEE.NUMERIC_STD* que permite realizar operaciones aritméticas con señales que representan números binarios con signo. El uso de las bibliotecas facilitó el diseño de estos módulos, ya que no hay que realizar las arquitecturas de estas operaciones a nivel de bit a bit.

A continuación, se describe cada módulo.

Multiplicador

Este módulo realiza la multiplicación de dos números de 16 bits representados en complemento a dos. Una de las entradas corresponde al peso de la neurona actual, y la otra representa una carac-

terística de entrada EMG o un resultado intermedio proveniente de la capa anterior. El resultado es un valor de 32 bits, que es el producto de las dos entradas. En el diseño completo del modelo de la red, se instancian 100 de estos módulos en paralelo para calcular simultáneamente todos los productos requeridos por cada neurona. La Figura 3.19 muestra el diagrama de caja negra del módulo multiplicador.

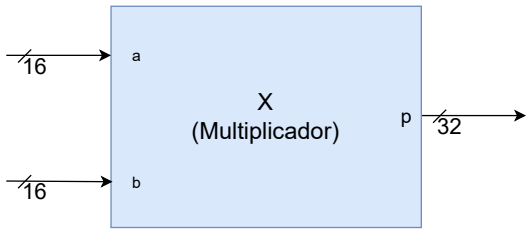


Figura 3.19: Diagrama de caja negra del módulo multiplicador.

Sumador

El sumador es el bloque que suma los productos generados por los multiplicadores, junto con el sesgo correspondiente a la neurona en proceso. El módulo tiene 101 entradas de 32 bits (100 productos + 1 sesgo) y una única salida también de 32 bits, que representa la salida de la neurona antes de la función de activación. La Figura 3.20 muestra su diagrama de caja negra.

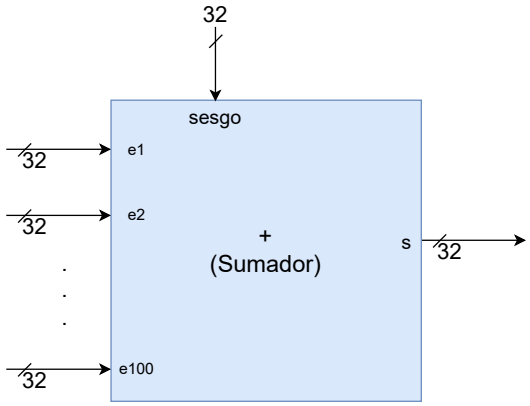


Figura 3.20: Diagrama de caja negra del módulo sumador.

Bloque ReLU

Este módulo realiza la función de activación ReLU, que se aplica al resultado del sumador. Este evalúa si el valor de entrada en complemento a dos es menor o igual a cero. En ese caso, la salida es cero. Si el valor es positivo, la salida reproduce el valor de la entrada. De esta manera, se aplica la transformación no lineal característica de la función ReLU. La Figura 3.21 muestra su caja negra.

Figura 3.21: Diagrama de caja negra del módulo ReLU.

3.5.8. Módulos de bloques de registros

Para almacenar los resultados intermedios y de salida del modelo de la red implementada en hardware, se diseñaron tres módulos que funcionan como bloques de registros especializados. Cada uno de los bloques gestiona los datos generados durante el proceso de inferencia, facilitando la transferencia de los datos entre las capas del modelo o hacia el bloque de decisión final. Para sincronizar los módulos en la arquitectura diseñada, se optó por utilizar estos bloques con múltiples registros internos organizados por dirección, al igual para realizar las tareas de lectura y transferencia de los datos en paralelo.

A continuación, se describen el funcionamiento de cada uno de estos bloques.

Bloque de registros actuales

Este módulo contiene 100 registros de 16 bits y se encarga de guardar los resultados de las neuronas en las capas intermedias, específicamente almacena los resultados en la capa que se está evaluando en la etapa presente del proceso. Cada resultado es escrito en un registro determinado por la señal proveniente del contador. La escritura es controlada por una señal de habilitación generada por la FSM. Los registros pueden ser leídos simultáneamente por la señal de salida que concatena los datos; esto permite su transferencia simultánea hacia el bloque de registros pasados.

En la Figura 3.22 se muestra el diagrama de caja negra de este módulo, y la Tabla 3.10 presenta sus señales de entrada y salida.

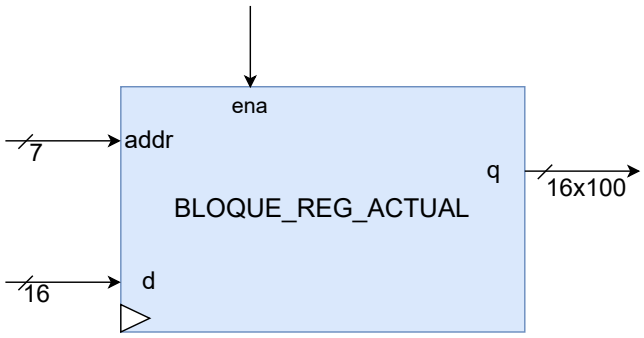


Figura 3.22: Diagrama de caja negra del bloque de registros actuales.

Tabla 3.10: Entradas y salidas del bloque de registros actuales.

Nombre	Tamaño	Tipo	Descripción
`clk`	1 bit	Entrada	Señal de reloj del sistema (25 MHz).
`rst`	1 bit	Entrada	Señal de reinicio asíncrono.
`addr`	7 bits	Entrada	Dirección de escritura del registro, proveniente del contador.
`ena`	1 bit	Entrada	Habilita la escritura del dato en el registro seleccionado.
`d`	16 bits	Entrada	Dato de activación a almacenar.
`q`	16x100 bits	Salida	Todos los registros disponibles para lectura simultánea.

Bloque de registros pasados

Este módulo también contiene 100 registros internos de 16 bits, al igual que el módulo de registros actuales. Su función es leer y almacenar todos los valores del bloque de registros actuales una vez que el proceso completo de activar las neuronas de una capa. La escritura se realiza en paralelo de todos los registros y se produce al activarse la señal de habilitación producida por la FSM. Todos los valores de este bloque se leen simultáneamente desde la salida que concatena los valores, para alimentar las siguientes neuronas en el proceso de la red.

La Figura 3.23 muestra su diagrama de caja negra y la Tabla 3.11 describe sus señales.

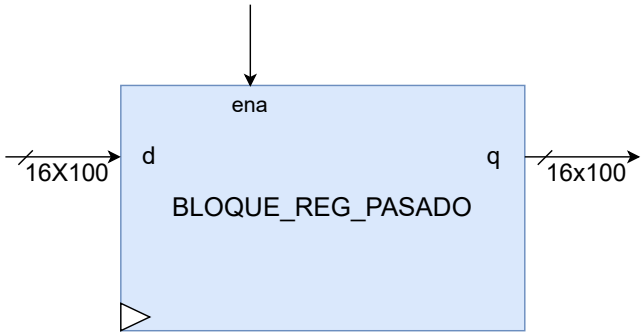


Figura 3.23: Diagrama de caja negra del bloque de registros pasados.

Tabla 3.11: Entradas y salidas del bloque de registros pasados.

Nombre	Tamaño	Tipo	Descripción
`clk`	1 bit	Entrada	Señal de reloj del sistema.
`rst`	1 bit	Entrada	Señal de reinicio asíncrono.

Continúa en la siguiente página

Tabla 3.11 – continuación de la página anterior

Nombre	Tamaño	Tipo	Descripción
`ena`	1 bit	Entrada	Habilita la escritura de todos los registros en paralelo.
`d`	16x100 bits	Entrada	Conjunto de datos provenientes del bloque de registros actuales.
`q`	16x100 bits	Salida	Registros disponibles para lectura paralela.

Bloque de registros de salida

Este bloque contiene 10 registros internos de 16 bits. Cada registro corresponde a cada clase del modelo a clasificar. Su funcionalidad es replicada del bloque de registros actuales, con la diferencia de que este bloque cuenta con menos registros, los cuales están dedicados a almacenar los resultados de las neuronas de la capa de salida. Los registros son leídos en paralelo para que el módulo de número mayor determine la clase con mayor activación.

La Figura 3.24 presenta el diagrama de caja negra de este módulo y la Tabla 3.12 su especificación funcional.

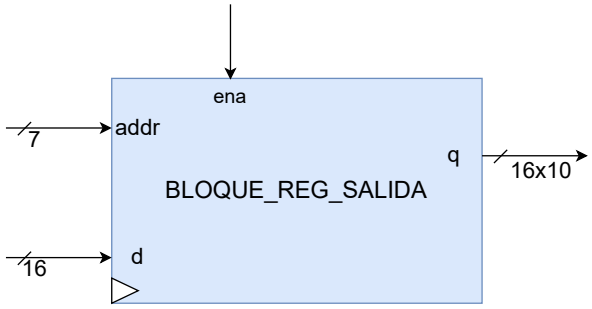


Figura 3.24: Diagrama de caja negra del bloque de registros de salida.

Tabla 3.12: Entradas y salidas del bloque de registros de salida.

Nombre	Tamaño	Tipo	Descripción
`clk`	1 bit	Entrada	Señal de reloj del sistema.
`rst`	1 bit	Entrada	Señal de reinicio.
`addr`	7 bits	Entrada	Dirección de escritura del resultado de activación de cada neurona de salida.
`ena`	1 bit	Entrada	Habilita la escritura del resultado.
`d`	16 bits	Entrada	Dato proveniente de las neuronas en la capa de salida.

Continúa en la siguiente página

Tabla 3.12 – continuación de la página anterior

Nombre	Tamaño	Tipo	Descripción
`q`	16x10 bits	Salida	Resultados de salida disponibles para comparación.

3.5.9. Módulo número mayor

El módulo número mayor, o nombrado *MOV_MAX* en el diseño, es el bloque final de la arquitectura digital de la red neuronal MLP implementada en FPGA. Funciona para determinar cuál de las salidas generadas por las 10 neuronas de la capa final tiene el mayor valor numérico, considerando la representación binaria en punto fijo, 8 bits para parte entera y 8 bits para la parte fraccionaria. Determinar la neurona de salida con mayor peso, equivale a identificar la clase predicha por el modelo. Este módulo funciona como la operación Argmax que sustituye a la función de activación Softmax empleada en el modelo en software, esto porque no es necesario obtener la distribución completa de probabilidades sobre las clases. Basta con identificar la salida con el valor máximo para determinar el gesto que clasificó la red.

El módulo genera un vector de 10 bits en su salida, donde únicamente uno de los bits es activado ('1'), donde, la posición de este bit indica la clase con mayor peso, es decir, la clase que clasificó la red. Mientras el proceso no termina, determinado por la FSM, el vector de salida permanece en ceros, esto para evitar una salida transitoria errónea.

En la Figura 3.25 se presenta el diagrama de caja negra del módulo, y en la Tabla 3.13 se describen sus señales de entrada y salida.

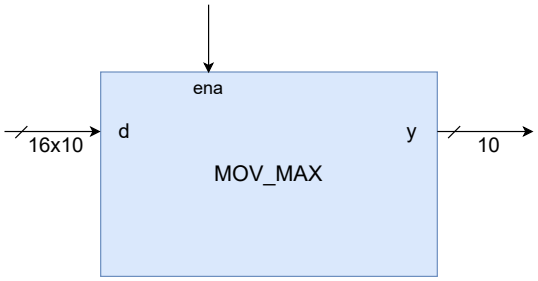


Figura 3.25: Diagrama de caja negra del módulo número mayor.

Tabla 3.13: Entradas y salidas del módulo número mayor.

Nombre	Tamaño	Tipo	Descripción
`ena`	1 bit	Entrada	Señal de habilitación. Cuando está activa ('1'), permite que el módulo realice la comparación y active la salida correspondiente. Si está en '0', la salida 'y' se mantiene en ceros.

Continúa en la siguiente página

Tabla 3.13 – continuación de la página anterior

Nombre	Tamaño	Tipo	Descripción
`d`	16x10 bits	Entrada	Vector que contiene las salidas de las 10 neuronas finales de la red. Cada valor representa el resultado asociado a una clase distinta.
`y`	10 bits	Salida	Vector codificado en un solo bit alto (‘1’) que representa la clase con la mayor activación. La posición activa indica la predicción del modelo.

3.6. Diseño para evaluar la precisión de la red MLP en hardware

Una vez completada la implementación de la red neuronal MLP en hardware, se diseñó una arquitectura adicional con el objetivo de evaluar la precisión en la clasificación de señales EMG por la red. En este nuevo diseño, la red MLP se emplea como un módulo funcional encapsulado dentro de un sistema mayor, ya que el diseño modular en el FPGA permite reutilizar los diseños. A Figura 3.26 presenta el diagrama de bloques del sistema completo diseñado.

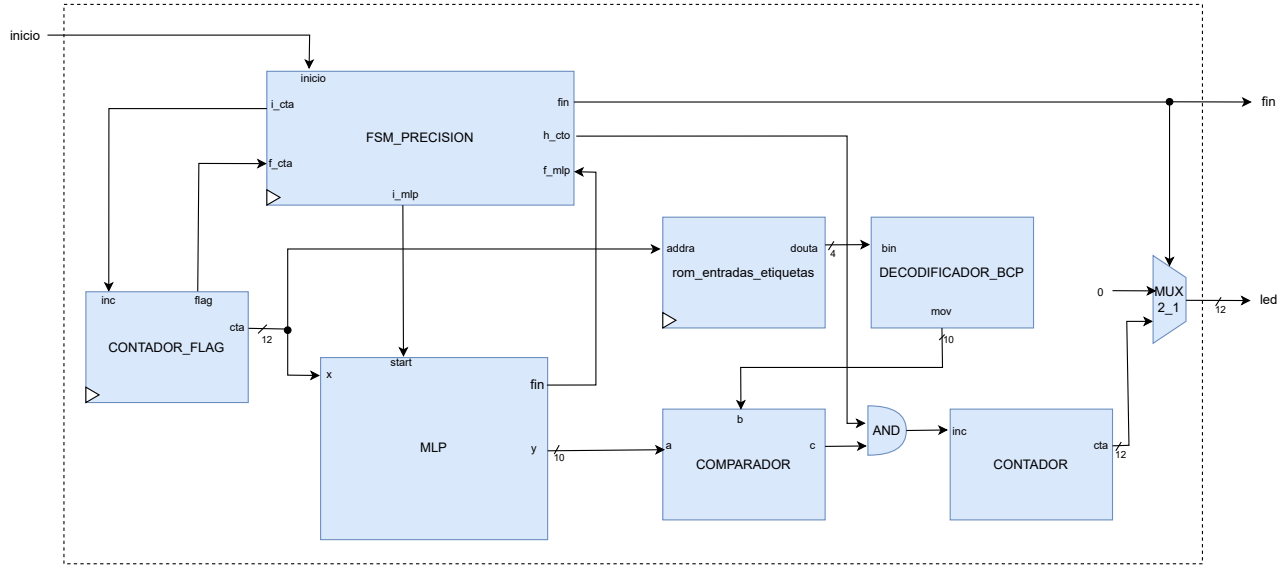


Figura 3.26: Diagrama de bloques del diseño de prueba de precisión para la red neuronal MLP en hardware.

Este sistema incluye una nueva FSM, memoria ROM, un decodificador, un multiplexor 2 a 1 y dos contadores, que interactúan con el diseño de red neuronal. A continuación se describen los módulos más relevantes que componen la arquitectura de pruebas de precisión. El módulo *MLP* ya no se describe, ya que su arquitectura fue desglosada en la sección anterior. Se hace especial énfasis en el módulo *FSM_PRECISION*, ya que es el más relevante al coordinar el flujo de datos.

- **FSM_PRECISION**: La máquina de estados finitos es de tipo *Moore* y se encarga de la sincronización secuencial de todos los módulos. Inicia el proceso de inferencia de la red, espera

la finalización del resultado, válida la predicción y actualiza los contadores correspondientes. En la Figura 3.27 se muestra el grafo de estados de esta FSM. Y en la Tabla 3.14 se muestra la descripción de entradas y salidas de este módulo FSM.

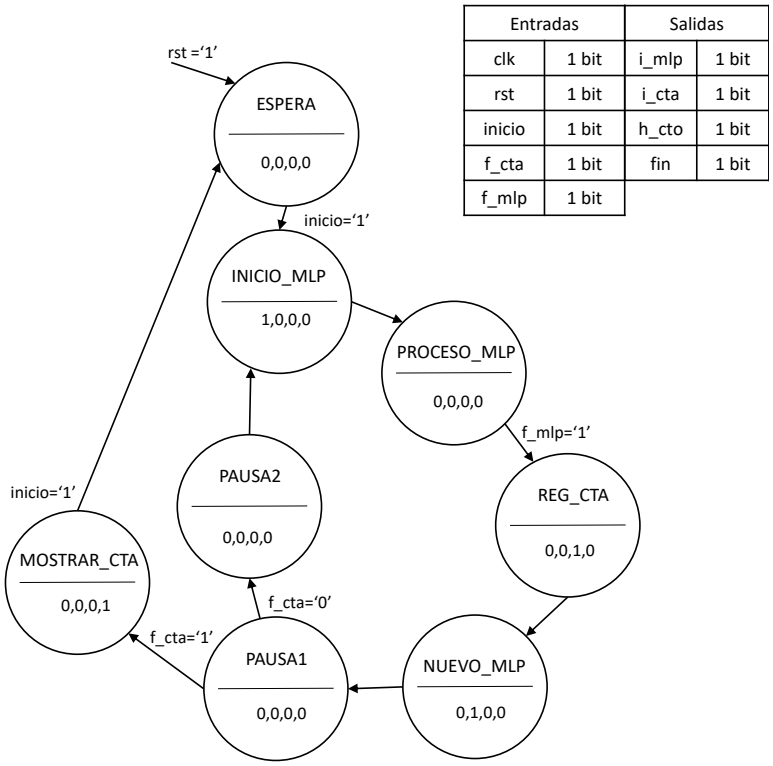


Figura 3.27: Grafo de los estados de la FSM para prueba de precisión de red neuronal.

Tabla 3.14: Entradas y salidas del módulo `FSM_PRECISION`.

Nombre	**Tamaño**	**Tipo**	**Descripción**
`clk`	1 bit	Entrada	Señal de reloj del sistema.
`rst`	1 bit	Entrada	Señal de reinicio asíncrono.
`inicio`	1 bit	Entrada	Señal para iniciar el proceso de evaluación de precisión.
`i_mlp`	1 bit	Salida	Señal de inicio para el módulo *MLP*, activa la propagación hacia adelante.
`fmlp`	1 bit	Entrada	Indica que la red MLP ha terminado la inferencia y su salida está disponible.
`h_cto`	1 bit	Salida	Habilita el incremento del contador de aciertos cuando la predicción es correcta.

Continúa en la siguiente página

Tabla 3.14 – continuación

Nombre	Tamaño	Tipo	Descripción
`i_cta`	1 bit	Salida	Señal para incrementar el contador de muestras una vez procesada cada muestra.
`f_cta`	1 bit	Entrada	Señal que indica que se han evaluado todas las muestras almacenadas.
`fin`	1 bit	Salida	Señal de finalización del proceso completo de evaluación. Permite mostrar el total de aciertos.

- **Contador con bandera**: Este bloque, nombrado como *CONBTADOR_FLAG* en el diseño, es un contador síncrono que lleva el control del número de muestras evaluadas. Su valor se utiliza como dirección para acceder a la ROM de etiquetas. La bandera de salida indica cuándo se ha alcanzado el total de muestras almacenadas.

- **Memoria ROM para etiquetas**: Este módulo, nombrado *rom_entradas_etiquetas* en el diseño, contiene las etiquetas reales correspondientes a cada muestra de entrada almacenada en el módulo *MLP*. Estas etiquetas fueron previamente extraídas de la base de datos EMG. El contenido de esta ROM permite validar la predicción realizada por la red neuronal, comparando la salida del módulo *MLP* con la clase esperada.

- **Decodificador de etiquetas a vector de 10 bits**

  Este módulo nombrado como *DECODIFICADOR_BCP* convierte las etiquetas almacenadas en la memoria ROM como números binarios de 0 a 9 a una representación por posición en un vector de 10 bits, donde únicamente el bit correspondiente a la clase está en alto. Esta representación es compatible con la salida del módulo *MLP*, que también entrega un vector de 10 bits donde el bit activo es la clase predicha según la ubicación de este. Se hace esta conversión para poder realizar la comparación.

- **Comparador**: El comparador verifica si la clase predicha por el módulo *MLP* es igual con la clase real proveniente de la ROM de etiquetas. Si ambos vectores coinciden, genera una señal en alto que indica un acierto en la predicción.

- **Contador de aciertos**

  Este contador incrementa su valor cada vez que el comparador detecta una predicción correcta, a la par que el proceso de la FSM habilita el incremento de aciertos, permitiendo así obtener el total de aciertos al final del proceso. El valor final del proceso se usa para calcular la precisión del modelo en hardware con respecto al total de las muestras probadas.

3.7. Diseño para control de servomotores con red neuronal en hardware

Una vez validada la precisión de la inferencia del modelo red neuronal MLP en hardware, se desarrolló el diseño de la arquitectura final para integrar el modelo a un sistema embebido de control físico. Este diseño utiliza las predicciones de la red neuronal para controlar directamente

cinco servomotores SG90 que simulan los movimientos de los dedos de la mano en el actuador para pruebas físicas. La arquitectura general de este sistema se presenta en la Figura 3.28.

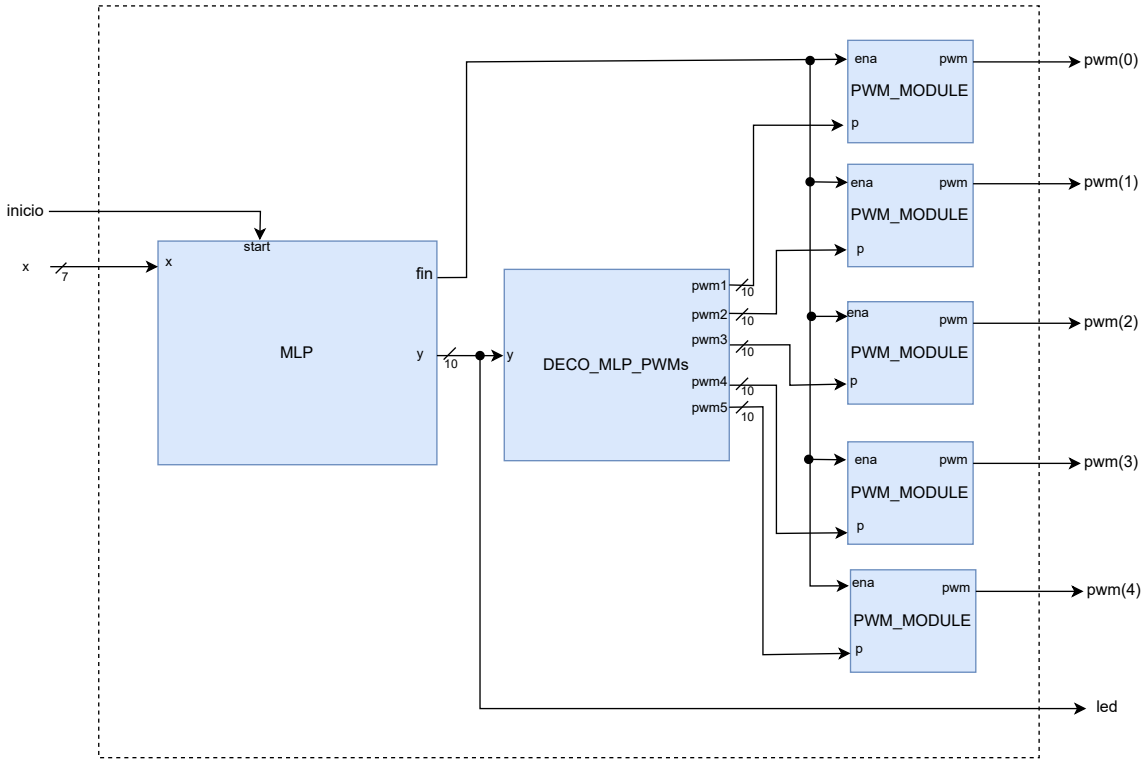


Figura 3.28: Diagrama de bloques del diseño de control de servomotores con red neuronal en hardware.

Esta arquitectura representa un sistema de control en lazo abierto, ya que no se emplea retroalimentación de posición. La salida generada por la red neuronal al procesar las características de las señales EMG produce las decisiones de control. El sistema está formado por tres bloques principales: el módulo *MLP*, el decodificador *DECO_MLP_PWMs* y los módulos generadores de PWM.

El módulo *MLP* es el núcleo de inferencia que recibe un vector de características de 67 componentes de 100 muestras para pruebas que se integraron en la memoria ROM interna. Y entrega un vector de 10 bits, donde un único bit en alto representa la clase predicha, correspondiente a una de las diez posibles posiciones de la mano.

El decodificador, *DECO_MLP_PWMSY*, interpreta la salida de 10 bits del *MLP* y genera cinco señales de control, una por servomotor, asignando a cada uno un ciclo de trabajo específico. Este módulo determina si cada dedo debe estar extendido o contraído dependiendo la clase que asignó la red en el proceso de clasificación de las muestras EMG. Cada clase se interpreta como una codificación que configura la posición de los cinco dedos; por lo tanto, la salida del decodificador se traduce en señales PWM ajustadas.

Cada servomotor SG90 está asociado a un módulo *PWM_MODULE*, el cual genera una señal de modulación por ancho de pulso de 20 ms de período y con una resolución de 10 bits. En este caso,

se definieron dos posiciones por dedo: extensión y contracción, determinadas mediante el ancho de los ciclos en alto. La Tabla 3.15 muestra la configuración correspondiente para cada servomotor.

Tabla 3.15: Configuraciones del ciclo en alto del PWM por dedo y posición.

Dedo (servomotor)	**Contracción**	**Extensión**
Pulgar	1.63 ms en alto de 20 ms	1.3 ms en alto de 20 ms
Índice	1.3 ms en alto de 20 ms	1.9 ms en alto de 20 ms
Medio	1.1 ms en alto de 20 ms	1.8 ms en alto de 20 ms
Anular	1.1 ms en alto de 20 ms	1.9 ms en alto de 20 ms
Meñique	1.1 ms en alto de 20 ms	1.9 ms en alto de 20 ms

Estos ciclos de trabajo en cada servomotor se eligieron de forma empírica con los servomotores acoplados a cada mecanismo que simula el movimiento de dedo hasta ajustar las posiciones deseadas.

El sistema resultante representa físicamente las clases predichas por el modelo de red neuronal MLP mediante los movimientos en paralelo de los dedos en el actuador de mano artificial.

3.8. Desarrollo de actuador que simule movimientos de mano

A la par que se desarrollaban las anteriores secciones de la metodología y como parte del sistema de validación físico para el modelo de red neuronal, se desarrolló un actuador impreso en 3D, el cual simula los movimientos de los dedos en una mano humana. Con este dispositivo se permite representar visualmente las clases predichas por la red MLP mediante el posicionamiento de los dedos de forma paralela, por el accionamiento de los servomotores.

Para el diseño del mecanismo que conforma los dedos se tuvo la referencia de un modelo de código abierto publicado en [72]. Este modelo base incluye un mecanismo con dos articulaciones para el dedo pulgar y otro mecanismo de tres articulaciones para el resto de los dedos. Se realizó un ajuste en el diseño para el ensamble de los servomotores SG90 empleados en este trabajo, ya que el modelo original está diseñado para servomotores de distinto modelo.

Se realizaron modificaciones, en los engranajes, actuadores y en los soportes de los servomotores, para el correcto ensamblaje con estos. Además, se rediseñó completamente la base y el soporte que alinean los cinco servomotores. Para el control de los dedos se optó un servomotor por dedo, ya que los mecanismos del diseño base, el dedo índice y medio comparten servo, al igual que el dedo anular y meñique.

La edición y creación de las piezas se realizó empleando el software *SolidWorks 2020®*. Las piezas se imprimieron con una impresora 3D modelo *Ender*, como se muestra en las Figuras 3.29 y 3.30.

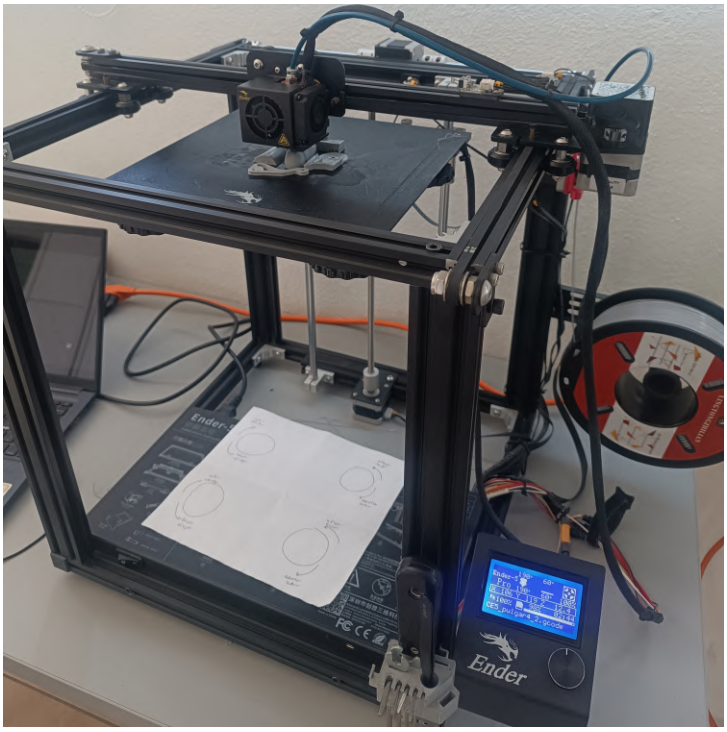


Figura 3.29: Proceso de impresión 3D del dispositivo.

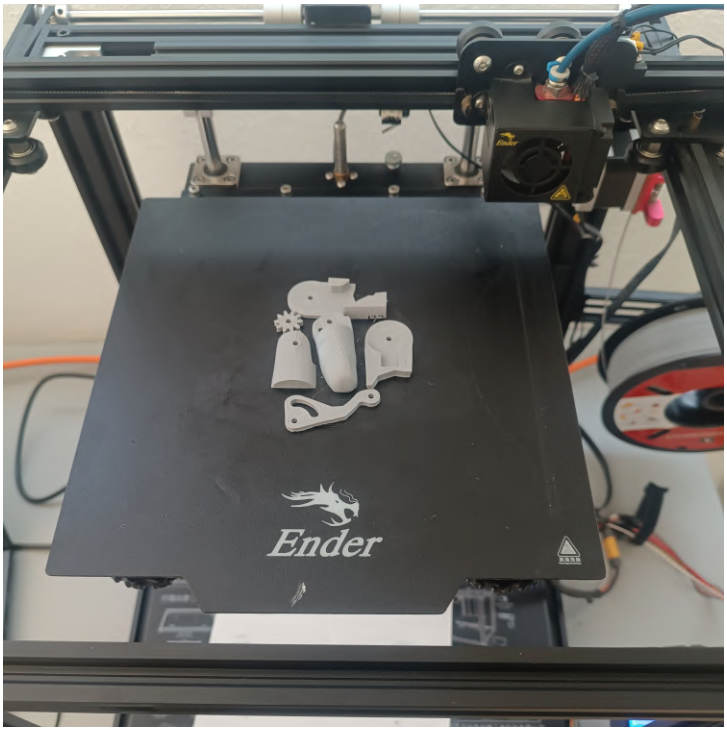


Figura 3.30: Impresión 3D de partes del dispositivo.

Las Figuras 3.31, 3.34, 3.32 y 3.33 muestran algunos componentes principales del diseño impreso, incluyendo la base y tapa para montaje de servos, así como los mecanismos individuales de los dedos.

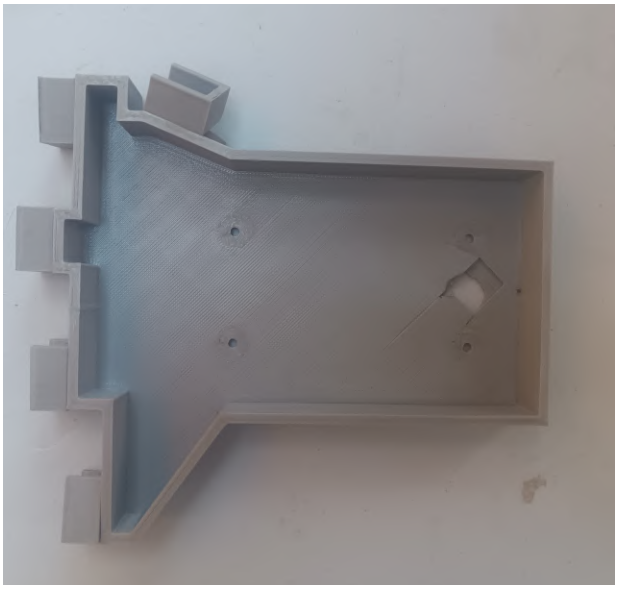


Figura 3.31: Soporte para servomotores.

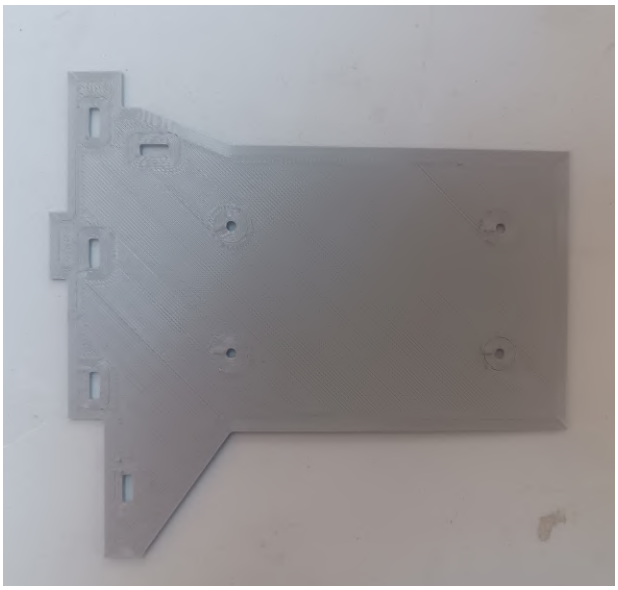


Figura 3.32: Tapa de soporte para servomotores.

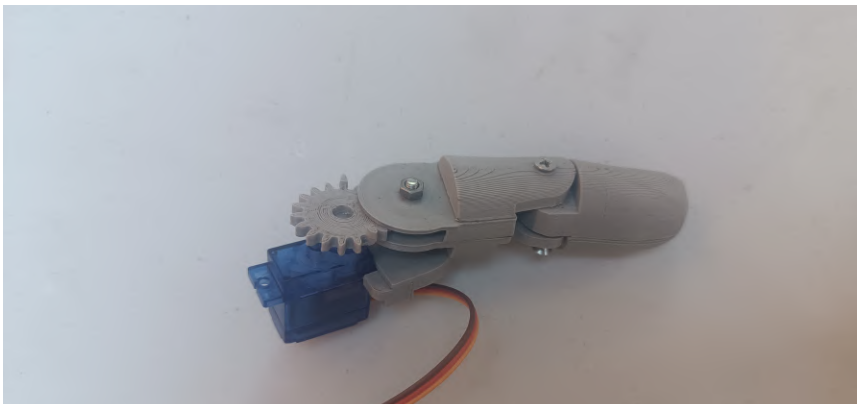


Figura 3.33: Dedo pulgar acoplado a servomotor.

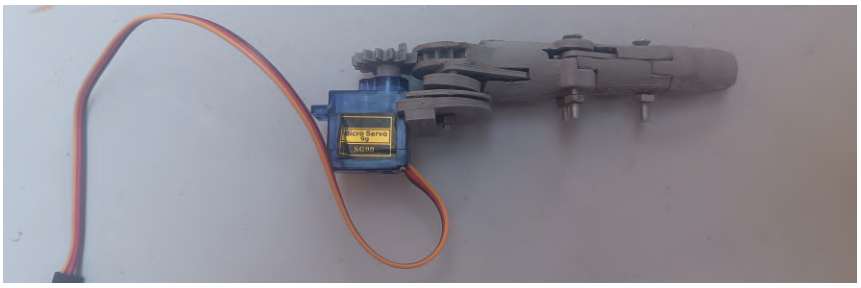


Figura 3.34: Dedo con tres articulaciones acoplado a servomotor.

Finalmente, en las Figuras 3.35 y 3.36 se muestra el dispositivo ensamblado en dos posiciones:

Figura 3.35: Dispositivo ensamblado con posición “mano extendida”.

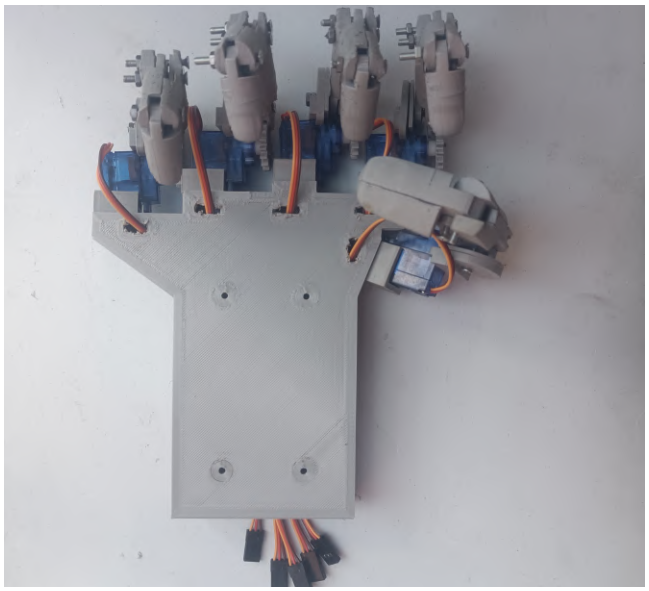


Figura 3.36: Dispositivo ensamblado con posición “mano cerrada”.

CAPÍTULO 4

Resultados

En esta sección se presenta el análisis de los resultados obtenidos. Se hacen varios análisis del modelo MLP implementado en hardware, tales como: análisis de la utilización de recursos en el FPGA, análisis de tiempo de procesamiento en la inferencia del modelo, análisis de potencia en el chip, y se analiza la eficiencia obtenida del modelo. Estos resultados se describen con más detalle en las secciones siguientes.

4.1. Recursos utilizados en FPGA

En el FPGA los recursos son limitados, como la lógica programable, los bloques de memoria, líneas de entrada/salida (IO), y bloques DSP. Evaluar la cantidad de recursos que requiere el diseño es de suma importancia, para considerar la capacidad del dispositivo para agregar nuevas etapas en el sistema o para su posible migración a otros dispositivos con diferentes capacidades. Las herramientas de síntesis del diseño nos proporcionan estos resultados de recursos en el diseño en este caso el software *Vivado 2021.1®*, en la tabla 4.1 se muestran el reporte de la utilización del diseño de red neuronal MLP en el FPGA *Artix 7®* que se empleó para la implementación.

Tabla 4.1: Utilización de recursos por la red neuronal MLP en FPGA Artix 7®.

Recurso	**Utilización**	**Disponibles**	**Utilizado (%)**
LUT	5735	20800	27.57
LUTRAM	3	9600	0.03
FF	3764	41600	9.05
BRAM	38	50	76.00
DSP	90	90	100.00
IO	21	106	19.81

Se muestra que los recursos críticos en este diseño son los bloques DSP, con una utilización del 100 %, lo que es esperado por la cantidad de multiplicaciones que se realizan en paralelo durante la inferencia de la red. Otro recurso de alta demanda son los bloques de RAM, ya que se utiliza el 76 % de los disponibles, los cuales se emplean para almacenar los pesos y sesgos del modelo. La

cantidad de lógica programable en las LUTs y flip-flops presenta un uso moderado, lo que indica que existe un margen para agregar lógica adicional para otras funciones. En cambio, las líneas IO emplean una baja utilización, lo cual es considerable para la posible adaptación a módulos externos o sensores.

De acuerdo al análisis de los recursos, si se requiere la implementación de una red de mayor tamaño con esta arquitectura de diseño, es necesario considerar un dispositivo FPGA con mayores recursos en DSPs y BRAM para no afectar el rendimiento de la inferencia. O también la posibilidad de considerar el uso de estrategias de optimización para reducir el uso de recursos

Además, con el fin de contextualizar la eficiencia de este diseño se hizo una comparación con trabajos reportados en el estado del arte en la Tabla 4.2, donde se puede apreciar resultados similares e incluso por debajo en la cantidad de recursos reportados con otras implementaciones de modelos de clasificación de señales EMG en FPGA.

Tabla 4.2: Comparación de uso de recursos en FPGA entre diferentes trabajos.

Trabajo	**Dispositivo**	**Modelo**	**FF**	**LUT**	**BRAM**	**DSP**
[73]	Pynq-Z1®	SVM	43624	38836	91	147
[74]	Zynq XC7Z020	KNN	9770	12783	33	16
[75]	Kintex 7® (XC7K325T)	SVM	186635	38087	94	108
[11]	Intel MAX 10®	Red neuronal binarizada (BNN)	–	3577	–	46
[31]	Zedboard®	Red neuronal de picos (R-SNN)	4256	7980	3	–
Este trabajo	Basys 3® (Artix 7®)	MLP	3764	5735	38	90

4.2. Tiempo de procesamiento

En esta sección se presenta el análisis del tiempo de inferencia del modelo MLP implementado en el FPGA, considerando la ejecución completamente en hardware. El tiempo de procesamiento en sistemas embebidos es fundamental para la clasificación de señales EMG en tiempo real.

El diseño de la red neuronal fue probado en la tarjeta *Basys 3®*, la cual cuenta con una frecuencia de reloj de 100 MHz, pero en esta frecuencia resulto incapaz de funcionar el modelo, debido al retraso necesario en los componentes físicos, tales como el tiempo necesario para acceder a los bloques de memoria RAM y para la propagación de las señales a lo largo de todos los módulos. Por lo que se optó por configurar la frecuencia del reloj a 25 MHz, lo cual equivale a un periodo de 40 ns por ciclo de reloj. Esta configuración del reloj del sistema se estableció al hacer pruebas físicas con el modelo implementado en la tarjeta de desarrollo.

Se evaluó la simulación del funcionamiento del diseño midiendo el número de ciclos de reloj necesarios desde que se inicia la inferencia de la red al activar la señal (`start`), hasta que se tiene una predicción de clase, indicada por la señal de finalización del proceso (`fin`). Con esto se obtuvo la latencia total de la red neuronal implementada, la cual fue de:

- **Latencia total:** 843 ciclos de reloj
- **Frecuencia del sistema:** 25 MHz ($T = 40$ ns)
- **Tiempo de inferencia:** 843×40 ns = **33.72** μs

La Figura 4.1 muestra la simulación obtenida en el entorno de desarrollo *Vivado 2021.1®*, donde se observa medición del tiempo de procesamiento.

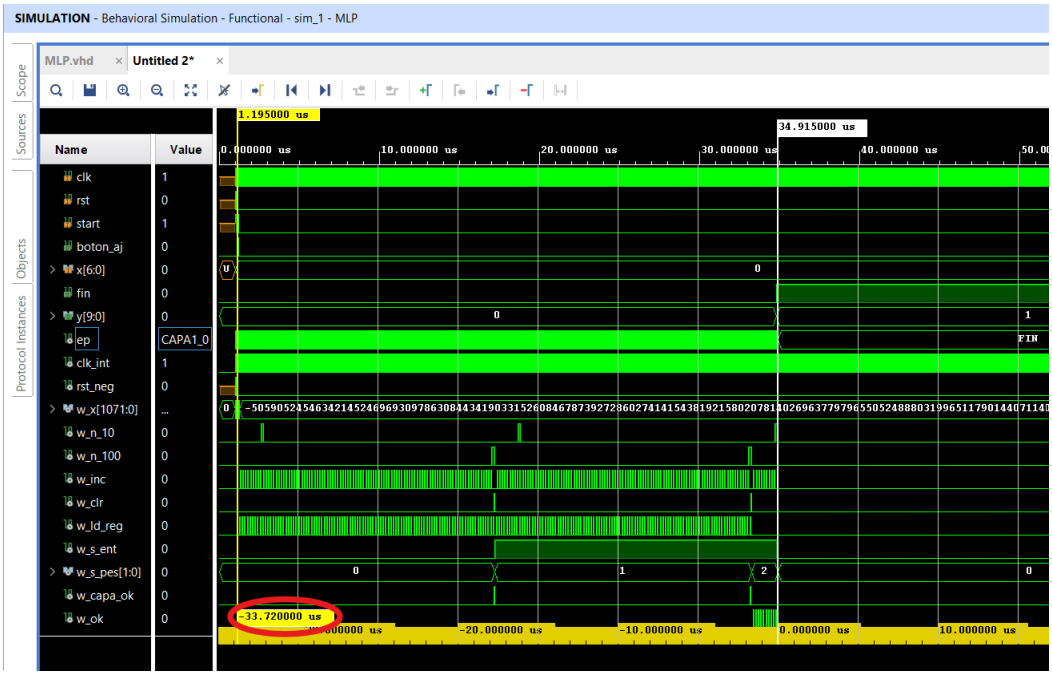


Figura 4.1: Simulación de la red para clasificar una muestra, con medición de tiempo de procesamiento.

Este resultado de latencia a la frecuencia de reloj configurado, indica que el modelo, en su caso ideal simulado, puede clasificar una muestra de entrada en menos de 34 microsegundos, lo que equivale a aproximadamente 29600 muestras por segundo. Logrando la inferencia del modelo de clasificación en tiempo real, adecuado para aplicaciones que requieran una respuesta lo más inmediata posible.

Esta medición de tiempo solo corresponde al proceso de inferencia de la red neuronal. Por lo que no se están considerando etapas como adquisición de las señales, procesamiento de estas, conversión A/D, entre otras etapas que pueden ser agregadas para tener una solución de sistema completo.

Para finalizar el análisis del tiempo de procesamiento del modelo, se realizó una comparación entre la implementación en hardware (FPGA) y la ejecución en software utilizando el lenguaje Python. El modelo en software fue ejecutado en una computadora portátil equipada con un procesador Intel® Core™ i7-1255U de 12ª generación a 1.70 GHz. La medición del tiempo en software se realizó utilizando las herramientas disponibles en Python *time* y *perf_counter*, con las que se hizo el registro del tiempo necesario para que el modelo tenga una predicción desde que se inicia la clasificación. Los resultados se muestran en la Tabla 4.3.

Tabla 4.3: Comparación del tiempo de procesamiento entre la implementación en software y hardware.

Implementación	Dispositivo	Tiempo de procesamiento
Software (Python)	Intel® Core™ i7-1255U @ 1.70 GHz	52.03730 ms
Hardware (VHDL – FPGA)	Artix 7® (Basys 3®)	0.03372 ms

Con esta comparación, se obtuvo que la inferencia del modelo en hardware logra una aceleración considerable con respecto a la ejecución en software.

4.3. Consumo de potencia en FPGA

Es importante conocer el consumo de potencia del diseño, para determinar su capacidad de usarse en aplicaciones embebidas donde la eficiencia energética es un factor clave. La estimación de potencia se obtuvo mediante la herramienta de análisis de consumo incluida en *Vivado 2021.1®*, tras la implementación del modelo MLP en el FPGA.

La Figura 4.2 muestra el reporte de potencia generado, donde se presenta tanto la potencia total estimada como la distribución entre componentes estáticos y dinámicos.

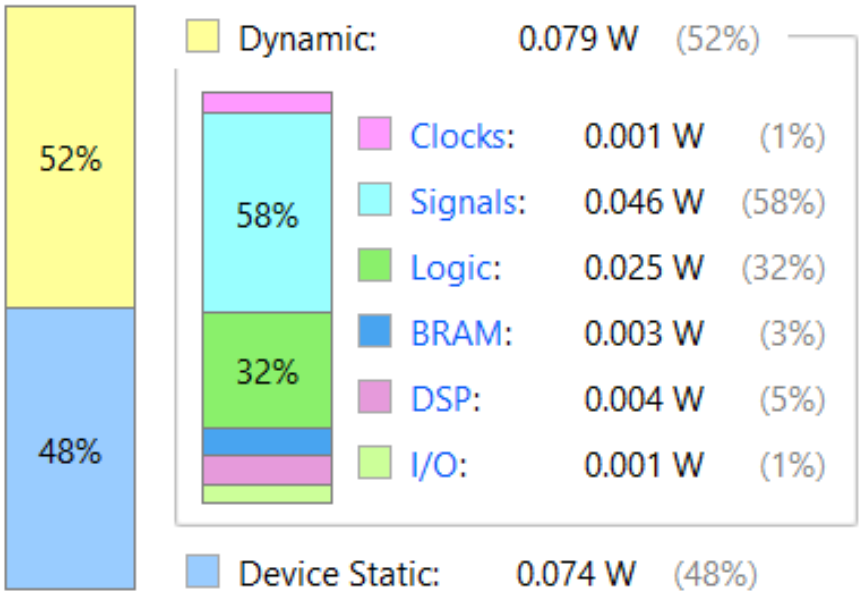


Figura 4.2: Reporte de potencia del modelo en el FPGA.

La potencia total estimada del diseño es de **0.153 W**, distribuida en dos componentes principales:

- **Potencia dinámica:** 0.079 W (52 % del total)

- **Potencia estática:** 0.074 W (48 % del total)

La potencia dinámica representa el consumo de la actividad de los relojes, señales internas, lógica programable, memoria BRAM, DSPS y periféricos durante el funcionamiento del modelo. En el modelo, el mayor consumo de la potencia dinámica está dada por:

- Señales de interconexión: 0.046 W (58 % de la potencia dinámica)
- Lógica: 0.025 W (32 %)
- BRAM, DSPs, I/O y relojes: 0.008 W en conjunto (10 %)

De acuerdo con este análisis, se concluye que la mayoría del consumo dinámico se encuentra en el ruteo de señales y en la lógica combinacional de la red neuronal.

La potencia estática corresponde al consumo sin actividad lógica, mientras el modelo se encuentra en espera. Representa el 48 % del consumo total, el cual está relacionado con el consumo medio en las hojas de datos del chip FPGA.

Con un consumo total en el chip de 153 mW, la implementación del modelo MLP en FPGA, muestra ser eficaz de forma energética y adecuada para aplicaciones portátiles. Aunque se debe tener en cuenta que esta medición solo está enfocada en el consumo en el chip, no se consideran los demás componentes en la tarjeta de desarrollo, la cual debe ser alimentada con 5 V para su funcionamiento.

4.4. Evaluación del modelo en FPGA

Para evaluar el rendimiento del modelo implementado en hardware, se realizaron pruebas de la red neuronal con el diseño descrito en la sección de la metodología 3.6. Donde este sistema compara las predicciones generadas por el modelo con las etiquetas reales provenientes de una memoria ROM, contabilizando los aciertos y errores de clasificación.

Para estas pruebas se evaluaron, 21490 muestras, correspondientes a una parte de los datos para pruebas que se asignaron de la base de datos. Se descartó una porción de los datos para no saturar la capacidad de la memoria ROM y, ya que el tiempo en simulación se incrementa considerablemente, entre más muestras.

Se documentaron los resultados de clasificación para cada muestra, con lo cual sé obtuvo la precisión de la red neuronal del **94.09 %**. Donde el modelo presentó una disminución en la precisión con respecto a la implementación en software, ya que se obtuvo una precisión de 96.19 % al ejecutar el algoritmo del modelo en *Python* con los mismos datos para pruebas. Esta diferencia se atribuye a la pérdida de valores debido a la representación de los datos, ya que en la implementación en hardware se utilizó una representación de 16 y 32 bits, en cambio, en el software se manejan datos de 64 bits.

Adicionalmente, con el registro de los resultados de la clasificación, se generó la matriz de confusión del modelo. En la figura 4.3 se presenta la matriz.

Figura 4.3: Matriz de confusión del modelo MLP implementado en FPGA.

En la matriz, se muestra la comparación de las predicciones de clases del modelo, representadas por las columnas, comparadas con las clases reales, representadas con las filas. De esta manera se permitió analizar en detalle los errores y aciertos de la clasificación. En la diagonal principal se muestran los casos en que la red clasificó de forma correcta. Los valores fuera de la diagonal equivalen a los casos en que la red calificó erróneamente.

De acuerdo con los resultados, la clase 10, que corresponde a la mano en posición de reposo, fue la que tiene la mayor cantidad de aciertos en la clasificación, debido a que su diferenciación es más notable, al no contar con activación significativa en los músculos. En cambio, la clase 5, que es la extensión de los dedos índice y medio es la que menos aciertos obtuvo, al confundirse con la clase 1 y clase 7, por la activación de músculos similares en los movimientos, ya que la clase 7 corresponde a la extensión del dedo índice, y la clase 1 a la extensión de los dedos a excepción del pulgar.

Con base en la matriz de confusión se obtuvieron las variables: verdaderos positivos, verdaderos negativos, falsos positivos y falsos positivos. Estas variables son necesarias para calcular las métricas

para evaluar el rendimiento del modelo. Cuyas ecuaciones son: ecuación 2.23 para exactitud, 2.24 para sensibilidad y 2.25 para obtener la especificidad. En la tabla 4.4 se muestran las métricas para cada clase.

Tabla 4.4: Métricas de evaluación del modelo MLP por clase.

Clase	**Exactitud**	**Sensibilidad**	**Especificidad**
Movimiento 1	98.288 %	96.231 %	98.516 %
Movimiento 2	98.571 %	93.951 %	99.085 %
Movimiento 3	99.172 %	96.789 %	99.436 %
Movimiento 4	98.846 %	93.532 %	99.436 %
Movimiento 5	98.227 %	86.738 %	99.504 %
Movimiento 6	98.497 %	90.368 %	99.400 %
Movimiento 7	98.944 %	94.416 %	99.447 %
Movimiento 8	99.004 %	97.255 %	99.199 %
Movimiento 9	98.967 %	92.880 %	99.643 %
Movimiento 10	99.684 %	98.837 %	99.778 %
Promedio	98.820 %	94.100 %	99.344 %

A partir de estos resultados, se calculó la eficiencia del modelo implementado, dando como resultado un porcentaje de **97.42 %**, cumpliendo con la hipótesis planteada.

4.5. Control del actuador por medio de red neuronal implementada en FPGA

Para validar el modelo de red neuronal, se realizó un control de un actuador que simula los movimientos de mano a partir de las predicciones del modelo. En las Figura 4.4 se muestra el actuador de la mano artificial en cada posición de las 10 clases que el modelo clasifica. De esta forma se apreciaron físicamente los resultados de la red neuronal. Y se muestra que el módulo de red neuronal MLP desarrollado puede aplicarse en el diseño de sistemas embebidos para aplicaciones útiles, en este caso el control de posición por servomotores en un sistema en lazo abierto.

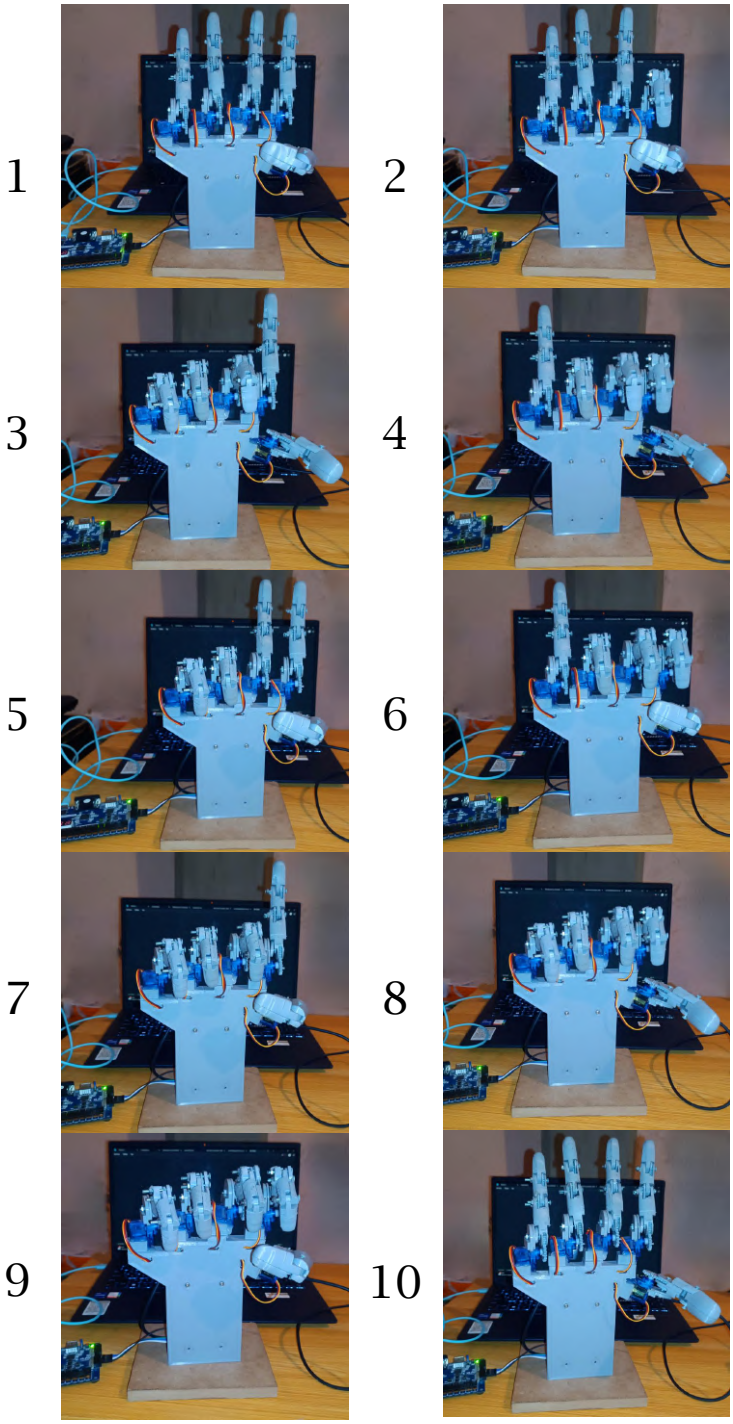


Figura 4.4: Pruebas de control de movimientos. 1) Oposición del pulgar. 2) Oposición de los dedos pulgar e índice. 3) Extensión de los dedos pulgar e índice. 4) Extensión de los dedos pulgar y meñique. 5) Extensión de los dedos índice y medio. 6) Extensión de meñique. 7) Extensión de índice. 8) Extensión de pulgar. 9) Mano cerrada. 10) Mano en reposo o mano abierta.

CAPÍTULO 5

Conclusiones

En este trabajo se describe cómo implementar una red neuronal MLP funcional para clasificar señales EMG en hardware con un dispositivo FPGA. La inferencia de la red se realizó de forma modular por bloques que conforman una arquitectura completa. La arquitectura de este trabajo permite clasificar 10 movimientos de la mano a partir de características extraídas de las señales EMG.

El FPGA se demostró como una plataforma adecuada para la inferencia de modelos de redes neuronales, empleando hasta 100 neuronas por capa, con la posibilidad de aumentarse, siguiendo la arquitectura propuesta. Además, la implementación en este tipo de dispositivos permite tiempos de procesamiento reducidos en la ejecución de la red neuronal, al ejecutar tareas en paralelo por su diseño a nivel hardware, así como por la configuración altamente personalizada de lógica y conexiones.

Según los resultados obtenidos, en cuestión de recursos del FPGA, estos incrementan a medida que incrementa la complejidad del modelo. Especialmente los requisitos en componentes como bloques de memoria BRAM, ya que estos se emplean para almacenar los pesos y sesgos de la red. También consideración en los bloques DSPs, útiles para realizar las multiplicaciones en el diseño. Al aumentar el tamaño de la red neuronal, hay que considerar estos recursos en el FPGA, por lo que resalta la necesidad de un diseño eficiente y balanceado. En este trabajo, los recursos de la tarjeta de desarrollo empleados fueron suficientes para ejecutar el diseño, pero si se espera aumentar la complejidad, es necesario optar por un dispositivo de mayores capacidades, o en su caso optar por usar técnicas de optimización de recursos en el diseño.

El modelo propuesto da como resultado un bloque funcional pre-diseñado que se puede integrar en un sistema FPGA de mayor tamaño para realizar la tarea específica de red neuronal MLP para clasificación. El diseño es modular y flexible, lo que facilita la migración a otro dispositivo y es posible aplicarlo en la calificación de diferentes tipos de datos.

De manera relevante, el calsificador EMG en el FPGa logra resultados adecuados para poder ser integrados en aplicaciones como prótesis inteligentes o aplicaciones de control por gesto. Considerando la integración de otras etapas y diseño de los componentes necesarios. Ya que en este trabajo solo se realizó un prototipo, para visualizar los resultados de forma física.

Como posibles trabajos futuros, se propone integrar las etapas de adquisición de señales EMG, así como la conversión AD, y el procesamiento de estas, especialmente su extracción de características. Para desarrollar un sistema más autónomo y completo. Así como mejorar el prototipo de mano artificial para su posible uso como prótesis inteligente.

Por otro lado, se propone utilizar el bloque prediseñado de la red MLP para otras aplicaciones útiles que puedan embeberse para posibles funciones portátiles que ayuden a más sectores de la población y de la industria.

Bibliografía

[1] M. Aviles, J. Rodríguez-Reséndiz, and D. Ibrahimi, "Optimizing emg classification through metaheuristic algorithms. technologies, 11(4), 87," 2023.

[2] M. R. Azghadi, C. Lammie, J. K. Eshraghian, M. Payvand, E. Donati, B. Linares-Barranco, and G. Indiveri, "Hardware implementation of deep network accelerators towards healthcare and biomedical applications," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 6, pp. 1138–1159, 2020.

[3] J. O. d. O. de Souza, M. D. Bloedow, F. C. Rubo, R. M. de Figueiredo, G. Pessin, and S. J. Rigo, "Investigation of different approaches to real-time control of prosthetic hands with electromyography signals," *IEEE Sensors Journal*, vol. 21, no. 18, pp. 20674–20684, 2021.

[4] A. Boschmann, G. Thombansen, L. Witschen, A. Wiens, and M. Platzner, "A zynq-based dynamically reconfigurable high density myoelectric prosthesis controller," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 1002–1007, IEEE, 2017.

[5] C. Cedeño, J. Cordova-Garcia, V. Asanza, R. Ponguillo, and L. Muñoz, "K-nn-based emg recognition for gestures communication with limited hardware resources," in *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pp. 812–817, IEEE, 2019.

[6] J. F. Castruita-López, M. Aviles, D. C. Toledo-Pérez, I. Macías-Socarrás, and J. Rodríguez-Reséndiz, "Electromyography signals in embedded systems: A review of processing and classification techniques," *Biomimetics*, vol. 10, no. 3, p. 166, 2025.

[7] E. Mastinu, J. Ahlberg, E. Lendaro, L. Hermansson, B. Håkansson, and M. Ortiz-Catalan, "An alternative myoelectric pattern recognition approach for the control of hand prostheses: A case study of use in daily life by a dysmelia subject," *IEEE journal of translational engineering in health and medicine*, vol. 6, pp. 1–12, 2018.

[8] S. Benatti, F. Casamassima, B. Milosevic, E. Farella, P. Schönle, S. Fateh, T. Burger, Q. Huang, and L. Benini, "A versatile embedded platform for emg acquisition and gesture recognition," *IEEE transactions on biomedical circuits and systems*, vol. 9, no. 5, pp. 620–630, 2015.

[9] D. Bonilla, M. Bravo, S. P. Bonilla, A. M. Iragorri, D. Mendez, I. F. Mondragon, C. Alvarado-Rojas, and J. D. Colorado, "Progressive rehabilitation based on emg gesture classification and an mpc-driven exoskeleton," *Bioengineering*, vol. 10, no. 7, p. 770, 2023.

[10] S. Bisi, L. De Luca, B. Shrestha, Z. Yang, and V. Gandhi, "Development of an emg-controlled mobile robot," *Robotics*, vol. 7, no. 3, p. 36, 2018.

[11] S. Kang, H. Kim, C. Park, Y. Sim, S. Lee, and Y. Jung, "semg-based hand gesture recognition using binarized neural network," *Sensors*, vol. 23, no. 3, p. 1436, 2023.

[12] H.-S. Choi, "Siamese neural network for user authentication in field-programmable gate arrays (fpgas) for wearable applications," *Electronics*, vol. 12, no. 19, p. 4030, 2023.

[13] A. T. Nguyen, M. W. Drealan, D. K. Luu, M. Jiang, J. Xu, J. Cheng, Q. Zhao, E. W. Keefer, and Z. Yang, "A portable, self-contained neuroprosthetic hand with deep learning-based finger control," *Journal of neural engineering*, vol. 18, no. 5, p. 056051, 2021.

[14] M. Aviles, J. Rodríguez-Reséndiz, and D. Ibrahimi, "Optimizing emg classification through metaheuristic algorithms," 2023.

[15] D. Baptista, S. Abreu, F. Freitas, R. Vasconcelos, and F. Morgado-Dias, "A survey of software and hardware use in artificial neural networks," *Neural Computing and Applications*, vol. 23, no. 3–4, p. 591–599, 2013.

[16] E. Tikhonov, K. Chebanov, and V. Burlyaeva, "Hardware and software implementation of neural network control of power systems based on the system of residual classes," in *2019 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)*, pp. 1–5, IEEE, 2019.

[17] M. R. Azghadi, C. Lammie, J. K. Eshraghian, M. Payvand, E. Donati, B. Linares-Barranco, and G. Indiveri, "Hardware implementation of deep network accelerators towards healthcare and biomedical applications," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 6, pp. 1138–1159, 2020.

[18] M. M. Saady and M. H. Essai, "Hardware implementation of neural network-based engine model using fpga," *Alexandria Engineering Journal*, vol. 61, no. 12, pp. 12039–12050, 2022.

[19] J. Hernandez, "Frecuencia y causas de amputaciÓn en pacientes atendidos en la direcciÓn de atenciÓn a la discapacidad, por el programa de apoyo de ayudas funcionales del dif estado de mÉxico, 2011-2012," 2012.

[20] "Porcentaje de la población con discapacidad según dificultad en la actividad 2020," 2020.

[21] E. V. V. Sánchez, "Los amputados y su rehabilitación," *Un reto para el Estado. Academia Nacional de Medicina. Mexico*, 2016.

[22] J. C. B. Gámez, F. Cabrera, ""diseno de una prótesis biomecánica para ninos"," 2016.

[23] S. C. A. y. E. M. R. F. Álvarez, ""desarrollo histórico y fundamentos teóricos de la electromiografía como medio diagnóstico"," 2006.

[24] I. Vasilev, D. Slater, G. Spacagna, P. Roelants, and V. Zocca, *Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow*. Packt Publishing Ltd, 2019.

[25] J. Venugopalan, L. Tong, H. R. Hassanzadeh, and M. D. Wang, "Multimodal deep learning models for early detection of alzheimer's disease stage," *Scientific Reports*, vol. 11, no. 1, 2021.

[26] M. B. I. Reaz, M. S. Hussain, and F. Mohd-Yasin, "Techniques of emg signal analysis: detection, processing, classification and applications," *Biological procedures online*, vol. 8, pp. 11–35, 2006.

[27] W. Liu, Q. Guo, S. Chen, S. Chang, H. Wang, J. He, and Q. Huang, "A fully-mapped and energy-efficient fpga accelerator for dual-function ai-based analysis of ecg," *Frontiers in Physiology*, vol. 14, p. 1079503, 2023.

[28] S. S. Selvi, B. D, A. Qadir, and P. K. R, "Fpga implementation of a face recognition system," *2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, 2021.

[29] H.-S. Choi, "Simple siamese model with long short-term memory for user authentication with field-programmable gate arrays," *Electronics*, vol. 13, no. 13, p. 2584, 2024.

[30] M. A. Scrugli, G. Leone, P. Busia, L. Raffo, and P. Meloni, "Real-time semg processing with spiking neural networks on a low-power 5k-lut fpga," *IEEE Transactions on Biomedical Circuits and Systems*, 2024.

[31] M. I. Ogbodo, K. N. Dang, and A. B. Abdallah, "Study of a multi-modal neurorobotic prosthetic arm control system based on recurrent spiking neural network," in *SHS Web of Conferences*, vol. 139, p. 03019, EDP Sciences, 2022.

[32] H.-S. Choi, "Electromyogram (emg) signal classification based on light-weight neural network with fpgas for wearable application," *Electronics*, vol. 12, no. 6, p. 1398, 2023.

[33] T. Triwiyanto, S. Luthfiyah, W. Caesarendra, and A. A. Ahmed, "Implementation of supervised machine learning on embedded raspberry pi system to recognize hand motion as preliminary study for smart prosthetic hand," *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, vol. 11, no. 3, pp. 685–699, 2023.

[34] A. Yılmaz, B. Büyükyılmaz, H. C. Sert, O. Uğuroğlu, and A. E. Algüner, "Hand movement classification with four channel emg signals for underactuated hand prosthesis test platform," in *2024 32nd Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, IEEE, 2024.

[35] R. D. Babu, S. S. Adithya, and M. Dhanalakshmi, "Design and development of an emg controlled transfemoral prosthesis," *Measurement: Sensors*, vol. 36, p. 101399, 2024.

[36] V. A. Bezrukov, R. R. Vakhitov, S. A. Chuykin, P. Y. Anuchin, A. V. Kruglov, and A. Y. Siziakova, "Evaluating the accuracy of knn classifier for gesture detection based on forearm emg signal," in *2024 6th International Youth Conference on Radio Electronics, Electrical and Power Engineering (REEPE)*, pp. 1–5, IEEE, 2024.

[37] R. Suppiah, K. Noori, K. Abidi, and A. Sharma, "Real-time edge computing design for physiological signal analysis and classification," *Biomedical Physics & Engineering Express*, 2024.

[38] P. A. Sanipatín-Díaz, P. D. Rosero-Montalvo, and W. Hernandez, "Portable facial expression system based on emg sensors and machine learning models," *Sensors*, vol. 24, no. 11, p. 3350, 2024.

[39] O. Kerdjidj, K. Amara, F. Harizi, and H. Boumridja, "Implementing hand gesture recognition using emg on the zynq circuit," *IEEE Sensors Journal*, 2023.

[40] M. R. Avilés *et al.*, "Clasificación de senales mioeléctricas por medio de algoritmos genéticos y máquinas de soporte de vectores," 2023.

[41] G. M. Hernandez *et al.*, "Diseño e implementación de una red neutral artificial en fpga para la identificación de sistemas en línea," 2013.

[42] I. J. R. Ángeles, "Detección y clasificación de señales mioeléctricas en el brazo mediante el uso de algoritmos basados en inteligencia artificial.," 2023.

[43] J. F. Girón *et al.*, "Algoritmos de clasificación de movimientos de señales mioeléctricas basados en técnicas kernel," 2021.

[44] D. C. A. Blair and F. John, "Emg control systems and methods for instructing extracorporeal prosthesis users," 2020.

[45] S. P. A. Einarsson and S. Atli, "Electromyography with prosthetic or orthotic devices," 2016.

[46] K. Momen and T. Kin, "Method, system and apparatus for real-time classification of muscle signals from self-selected intentional movements," 2013.

[47] M. Reaz, M. Hussain, and F. Mohd-Yasin, "Techniques of emg signal analysis: detection, processing, classification and applications (correction)," *Biological procedures online*, vol. 8, pp. 163–163, 2006.

[48] D. C. Toledo-Pérez, J. Rodríguez-Reséndiz, R. A. Gómez-Loenzo, and J. Jauregui-Correa, "Support vector machine-based emg signal classification techniques: A review," *Applied Sciences*, vol. 9, no. 20, p. 4402, 2019.

[49] L. Bi, C. Guan, *et al.*, "A review on emg-based motor intention prediction of continuous human upper limb motion for human-robot collaboration," *Biomedical Signal Processing and Control*, vol. 51, pp. 113–127, 2019.

[50] M. Hakonen, H. Piitulainen, and A. Visala, "Current state of digital signal processing in myoelectric interfaces and related applications," *Biomedical Signal Processing and Control*, vol. 18, pp. 334–359, 2015.

[51] S. Zhou, K. Yin, F. Fei, and K. Zhang, "Surface electromyography–based hand movement recognition using the gaussian mixture model, multilayer perceptron, and adaboost method," *International Journal of Distributed Sensor Networks*, vol. 15, no. 4, p. 1550147719846060, 2019.

[52] T. Song, Z. Yan, S. Guo, Y. Li, X. Li, and F. Xi, "Review of semg for robot control: techniques and applications," *Applied Sciences*, vol. 13, no. 17, p. 9546, 2023.

[53] T. M. Vieira, R. Merletti, and L. Mesin, "Automatic segmentation of surface emg images: Improving the estimation of neuromuscular activity," *Journal of biomechanics*, vol. 43, no. 11, pp. 2149–2158, 2010.

[54] M. Rojas-Martínez, M. A. Mañanas, and J. F. Alonso, "High-density surface emg maps from upper-arm and forearm muscles," *Journal of neuroengineering and rehabilitation*, vol. 9, pp. 1–17, 2012.

[55] E. Tikhonov, K. Chebanov, and V. Burlyaeva, "Hardware and software implementation of neural network control of power systems based on the system of residual classes," *2019 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)*, 2019.

[56] J. Zou, Y. Han, and S.-S. So, "Overview of artificial neural networks," *Artificial neural networks: methods and applications*, pp. 14–22, 2009.

[57] A. Krenker, J. Bešter, and A. Kos, "Introduction to the artificial neural networks," *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech*, pp. 1–18, 2011.

[58] A. Abraham, "Artificial neural networks," *Handbook of measuring system design*, 2005.

[59] A. N. Perez-Garcia, G. M. Tornez-Xavier, L. M. Flores-Nava, F. Gómez-Castañeda, and J. A. Moreno-Cadenas, "Multilayer perceptron network with integrated training algorithm in fpga," in *2014 11th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, pp. 1–6, IEEE, 2014.

[60] A. Krogh, "What are artificial neural networks?," *Nature biotechnology*, vol. 26, no. 2, pp. 195–197, 2008.

[61] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[62] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia tools and applications*, vol. 80, pp. 8091–8126, 2021.

[63] K. Sastry, D. Goldberg, and G. Kendall, "Genetic algorithms," *Search methodologies: Introductory tutorials in optimization and decision support techniques*, pp. 97–125, 2005.

[64] C. Maxfield, *The design warrior's guide to FPGAs: devices, tools and flows*. Elsevier, 2004.

[65] W. Wolf, *FPGA-based system design*. Pearson education, 2004.

[66] M. M. Mano and J. F. Gonzalez, *Diseño digital*. Pearson Educación, 2003.

[67] N. Jiang, A. Pradhan, and J. He, "Gesture recognition and biometrics electromyogram (grabmyo)." PhysioNet, 2022. Version 1.0.2. RRID:SCR_007345.

[68] K. Englehart, B. Hudgin, and P. A. Parker, "A wavelet-based continuous classification scheme for multifunction myoelectric control," *IEEE Transactions on Biomedical Engineering*, vol. 48, no. 3, pp. 302–311, 2001.

[69] D. C. Toledo-Pérez, J. Rodríguez-Reséndiz, R. A. Gómez-Loenzo, and J. Jauregui-Correa, "Support vector machine-based emg signal classification techniques: A review," *Applied Sciences*, vol. 9, no. 20, p. 4402, 2019.

[70] K. Englehart and B. Hudgins, "A robust, real-time control scheme for multifunction myoelectric control," *IEEE transactions on biomedical engineering*, vol. 50, no. 7, pp. 848–854, 2003.

[71] M.-F. Lucas, A. Gaufriau, S. Pascual, C. Doncarli, and D. Farina, "Multi-channel surface emg classification using support vector machines and signal-based wavelet optimization," *Biomedical Signal Processing and Control*, vol. 3, no. 2, pp. 169–174, 2008.

[72] N. Witham, "Atlas hand." `https://grabcad.com/library/atlas-hand-1`, 2024. 2018-09-22.

[73] G. Franco, P. Cancian, L. Cerina, E. Besana, N. Beretta, and M. D. Santambrogio, "Fpga-based muscle synergy extraction for surface emg gesture classification," in *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–4, IEEE, 2017.

[74] O. Kerdjidj, K. Amara, F. Harizi, and H. Boumridja, "Implementing hand gesture recognition using emg on the zynq circuit," *IEEE Sensors Journal*, vol. 23, no. 9, pp. 10054–10061, 2023.

[75] M. Majolo and A. Balbinot, "Proposal of a hardware svm implementation for fast semg classification," in *XXVI Brazilian Congress on Biomedical Engineering: CBEB 2018, Armação de Buzios, RJ, Brazil, 21-25 October 2018 (Vol. 2)*, pp. 381–386, Springer, 2019.