UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Electrónica

IMPLEMENTACIÓN EN FPGA DE REDES NEURONALES ARTIFICIALES APLICADAS A LAZOS DE CONTROL PREDICTIVO POR MODELO

JUAN JOSÉ VÁSQUEZ CÁDIZ

MEMORIA DE TITULACIÓN Y TESIS DE GRADO PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO Y AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA ELECTRÓNICA

PROFESOR SUPERVISOR : GONZALO CARVAJAL BARRERA
PROFESOR CO-SUPERVISOR : CÉSAR SILVA JIMÉNEZ

29 DE AGOSTO DE 2025

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☐ Memoria o trabajo de título; ☒ Tesis de Postgrado;

Título del trabajo: Implementación en FPGA de redes neuronales artificiales aplicadas a lazos de control predictivo por modelo

Nombre del candidato(a): Juan José Vásquez Cádiz

Carrera / Grado: Magíster en Ciencias de la Ingeniería Electrónica

Campus: Casa Central Valparaiso **;** **Departamento**: Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Gonzalo Carvajal Barrera, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 26/08/2025 ; **Firma:** _____

Estudiante o Candidato(a):

Fecha: 26/08/2025 ; **Firma:** _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

A mi familia...

Agradecimientos

Quisiera comenzar expresando mi gratitud a mi familia por el cariño y el apoyo incondicional durante todo este proceso. Nada de esto habría sido posible sin el esfuerzo de mis padres, quienes han sido el pilar central que me ha permitido avanzar hasta este punto. También quiero agradecer especialmente a mi hermana por su compañía y comprensión a lo largo de estos años.

Asimismo, quiero agradecer a los miembros del Laboratorio Turing, cuya disposición y enseñanzas fueron de gran ayuda para salir de la caja y entender mis propios errores. Todas esas misiones secundarias que compartimos, sin duda, son una muestra de que disfruté el proceso.

Quisiera también agradecer a mis amigos por recordarme la importancia de mantener el equilibrio y permitirme distracciones. Han sido una fuente inagotable de chistes y sinsentidos durante toda la etapa universitaria, una verdadera maravilla. No podría pedir más de ustedes.

Finalmente, deseo expresar mi gratitud al profesor César Silva por su apoyo durante el desarrollo de esta tesis, y al profesor Gonzalo Carvajal, cuya guía, consejos y paciencia han sido invaluables desde mucho antes de iniciar este trabajo. A ambos, muchas gracias por el tiempo dedicado, los desafíos planteados y el espacio brindado para la creatividad.

Esta tesis se enmarca en el proyecto Fondecyt Regular 1211676, cuyo financiamiento hizo posible su desarrollo. Además, también se recibió apoyo financiero por parte del proyecto interno multidisciplinario PI_M_23_05 de la Universidad Técnica Federico Santa María.

Resumen

Tradicionalmente, la implementación en tiempo real de esquemas de *Model Predictive Control* (MPC) ha estado limitada por las exigencias computacionales y de almacenamiento. El uso de redes neuronales artificiales (ANNs) para aproximar la ley de control de MPC ha surgido como un enfoque prometedor, especialmente cuando se implementa en *Field Programmable Gate Arrays* (FPGAs). Este trabajo presenta un flujo de diseño completo *end-to-end* para la implementación de aproximadores de MPC basados en ANNs, integrando Keras y QKeras para el diseño y cuantización de ANNs, junto con HLS4ML y el entorno de diseño de AMD/Xilinx para su implementación en FPGA. La metodología se valida mediante dos casos de estudio: un sistema compuesto por un motor DC y un sistema de recursos energéticos distribuidos (DER).

Los resultados experimentales demuestran que la selección de hiperparámetros y niveles de cuantización de la ANN impacta significativamente el error de aproximación y el uso de recursos en la implementación. Las implementaciones en FPGA alcanzan latencias inferiores a un microsegundo, con un uso de recursos que varía según la estructura de la red neuronal y el nivel de paralelismo del hardware. El análisis de los compromisos de diseño destaca el impacto de la cuantización y la reutilización de hardware en la latencia y el uso de recursos, proporcionando directrices para optimizar controladores MPC basados en ANNs para aplicaciones de tiempo real. Estos hallazgos respaldan la viabilidad del uso de ANNs para aproximar MPC implementadas en FPGA, ofreciendo una alternativa práctica a la implementación de MPC tradicional, permitiendo una ejecución con uso de recursos y latencia balanceados adecuadamente para sistemas embebidos con restricciones de tiempo.

Abstract

Traditionally, the real-time implementation of Model Predictive Control (MPC) schemes has been limited by computational and storage demands. The use of artificial neural networks (ANNs) to approximate the MPC control law has emerged as a promising approach, particularly when implemented on Field Programmable Gate Arrays (FPGAs). This work presents a complete end-to-end design flow for implementing ANN-based MPC approximators, integrating Keras and QKeras for ANN design and quantization, along with HLS4ML and the AMD/Xilinx design environment for FPGA deployment. The methodology is validated through two case studies: a system composed of a DC motor and a distributed energy resources (DER) system.

Experimental results demonstrate that the selection of hyperparameters and quantization levels of the ANN significantly impacts both the approximation error and resource utilization in the implementation. The FPGA implementations achieve latencies below one microsecond, with resource usage varying depending on the neural network structure and the hardware parallelism level. The design trade-off analysis highlights the impact of quantization and hardware reuse on latency and resource consumption, providing guidelines for optimizing ANN-based MPC controllers for real-time applications. These findings support the feasibility of using ANNs to approximate MPC laws implemented on FPGAs, offering a practical alternative to traditional MPC implementations, enabling execution with a balanced trade-off between resource usage and latency for embedded systems with real-time constraints.

Índice general

Índice de tablas

Índice de figuras

Acrónimos

Acrónimo	Descripción
ADMM	Alternating Direction Method of Multipliers
ANN	Artificial Neural Network
BRAM	Block Random Access Memory
CLB	Configurable Logic Block
CPU	Central Processing Unit
DER	Distributed Energy Resources
DNN	Deep Neural Network
DSP	Digital Signal Processor
FF	Flip-Flop
FPGA	Field Programmable Gate Array
FGM	Fast Gradient Method
HDL	Hardware Description Language
HIL	Hardware-in-the-Loop
HLS	High-Level Synthesis
IOB	Input/Output Block
IP	Intellectual Property
LP	Linear Programming
LUT	Look-Up Table
MERS	Multiplications-Error Ranking Score
MPC	Model Predictive Control
mpQP	Multi-Parametric Quadratic Programming
MSE	Mean Squared Error
PWA	Piecewise Affine
QP	Quadratic Programming
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RTL	Register Transfer Level
SoC	System-on-Chip

Capítulo 1

Introducción

En este capítulo, se presenta el contexto y la motivación del trabajo, seguido de la formulación del problema a resolver. Posteriormente, se detallan los alcances y contribuciones de esta tesis para, finalmente, concluir con la organización de los capítulos siguientes de este informe.

1.1 Contexto y Motivación

Model Predictive Control (MPC) es una técnica avanzada de control automático que emplea un modelo matemático de la planta a controlar para anticipar el comportamiento futuro de los estados y salidas frente a una secuencia de entradas de control aplicadas a lo largo de un número predefinido de muestras futuras, conocido como horizonte de predicción. Utilizando la información del estado actual y las predicciones para cada muestra dentro del horizonte de predicción, se resuelve un problema de optimización que determina la secuencia de entradas que conducen la salida a un valor deseado, al tiempo que se cumplen las restricciones de las variables internas y entradas del sistema.

En el caso de un sistema lineal e invariante en el tiempo, el problema de optimización resultante se puede formular en la forma de *Quadratic Programming* (QP) [1]. En este contexto, la función objetivo es cuadrática y las restricciones son lineales. Los problemas QP se resuelven a través de métodos numéricos que ajustan las variables de control iterativamente para aproximar la actuación óptima que minimiza un objetivo a medida que respeta las restricciones del sistema. En general, el tiempo de ejecución de los algoritmos de optimización depende del número de muestras en el horizonte de predicción, el número de variables de control, la cantidad de restricciones y el número de iteraciones necesarias para aproximar la solución óptima con cierto grado de error.

En la práctica, la implementación de MPC se aborda mediante formulaciones implícitas o explícitas. En la formulación implícita, en cada intervalo de muestreo se plantea un problema de optimización en base al estado actual del sistema, el cual se resuelve de manera *online* utilizando métodos numéricos iterativos que convergen a una aproximación de la solución. El proceso de plantear y resolver un problema de optimización se repite en cada instante de muestreo, lo que permite al controlador responder a cambios en el entorno y en las dinámicas del sistema, a cambio de un elevado costo computacional que suele resultar prohibitivo para sistemas con dinámicas rápidas o con recursos computacionales limitados [2]. Por otro lado, la formulación explícita de MPC se basa en plantear un único problema de optimización en tiempo de diseño, el cual se resuelve de manera *offline* para todos los estados posibles del sistema. El espacio de soluciones del problema de optimización, o ley de control, se representa por un conjunto finito de regiones politópicas, donde cada región tiene asociada una función lineal que relaciona el estado del sistema con la actuación óptima correspondiente. La ley de control precalculada se almacena en memoria, y en tiempo de ejecución, la acción de control para el estado actual se obtiene mediante la búsqueda de la región correspondiente al estado y la evaluación de una función asociada, lo que simplifica significativamente las operaciones requeridas en comparación con un algoritmo numérico iterativo [1].

Una forma exacta de obtener la ley de control para la formulación explícita es a través de *Multi-Parametric Quadratic Programming* (mpQP) [3]. En tiempo de ejecución, la evaluación de la ley de control exacta obtenida con mpQP conlleva un menor tiempo de ejecución comparado con

MPC implícito para sistemas de baja dimensionalidad; sin embargo, la memoria requerida para almacenar las regiones que componen la ley de control exacta aumenta rápidamente con el tamaño de las matrices del problema QP, las cuales a su vez crecen a medida que se extiende el horizonte de predicción, el número de estados y las restricciones del problema [1]. Además del aumento en los requisitos de memoria, el aumento en el número de regiones de la ley de control también implica un mayor tiempo de búsqueda de la región activa durante la ejecución del lazo de control [3, 4]. Estas limitaciones de escalabilidad se vuelven particularmente relevantes cuando el controlador debe implementarse en plataformas embebidas con limitada capacidad de cómputo y almacenamiento. Aunque el MPC explícito ha permitido implementar lazos de control con latencias en el orden de algunos microsegundos [4, 5], esto solo se ha logrado en sistemas de baja dimensionalidad.

Durante los últimos años, investigadores han reportado diversos enfoques para reducir el costo computacional de las formulaciones explícitas de MPC, apuntando a reducir los requerimientos de memoria y tiempo de ejecución para extender su aplicabilidad a sistemas de mayor complejidad. Entre los enfoques propuestos se incluyen el uso de árboles de búsqueda para optimizar la evaluación de la ley de control [6] y la implementación de versiones aproximadas de la misma [7]. Posteriormente, se han estudiado métodos como la reducción de la cantidad de regiones para aproximar la ley de control [8], la exploración de diferentes representaciones numéricas para codificar dicha ley [9], e incluso el uso de versiones comprimidas de las regiones de la ley de control [5]. Más recientemente, se ha planteado el uso de redes neuronales artificiales (ANNs, del inglés *Artificial Neural Networks*) entrenadas para imitar al algoritmo MPC [10–13].

El uso de ANNs para implementar aproximaciones de la ley de control presenta múltiples potenciales ventajas en términos de eficiencia computacional, sobre todo para controladores que requieran operación en tiempo real con baja latencia y que deban ser implementados en plataformas embebidas. El uso de ANNs en MPC se basa en la capacidad de estas redes para aproximar funciones de complejidad arbitraria mediante una estructura de cómputo regular y pesos ajustables entrenados con datos de ejemplo. En específico, se ha mostrado que las redes del tipo *fully connected* pueden funcionar como aproximadores universales de funciones [14], lo cual permite aproximar la ley de control con un error arbitrario y requiriendo significativamente menos memoria que la ley de control exacta [15]. Una vez entrenada, la ANN realiza inferencias sobre nuevas entradas por medio de un número fijo de operaciones que incluyen álgebra lineal y evaluación de funciones de activación, lo cual facilita la derivación de cotas para el tiempo de cálculo de la actuación óptima. Por otro lado, las operaciones requeridas para realizar inferencias presentan un grado de independencia que las hace adecuadas para ejecución paralela. Además, las ANNs pueden implementarse utilizando representación numérica reducida o cuantizada para los pesos y activaciones, lo cual permite reducir aún más los requerimientos de memoria y costo computacional de las operaciones sin un aumento significativo del error de aproximación [16].

Dentro de las tecnologías computacionales para la implementación de algoritmos de MPC, las FPGAs han ganado terreno en los sistemas de control que requieren operación en tiempo real y baja latencia. Las FPGAs son dispositivos compuestos por arreglos de bloques lógicos e interconexiones programables, que pueden configurarse granularmente para requerimientos específicos. Esto permite a los desarrolladores implementar arquitecturas especializadas que pueden explotar paralelismo espacial y temporal de las operaciones, así como la posibilidad de utilizar representaciones de datos arbitrarias, con el fin de cumplir con los requisitos de la aplicación de forma costo-efectiva, balanceando la latencia de ejecución, la precisión numérica y el uso de recursos.

Las FPGAs se han usado para implementar lazos MPC en formulaciones implícitas [4, 17] y explícitas [5, 18], y más recientemente, en la implementación de algoritmos MPC aproximados basados en ANNs [19–22]. En este último caso, los desarrollos aún son incipientes, y los trabajos existentes se centran principalmente en demostrar la factibilidad de utilizar redes neuronales para aproximar la ley de control de casos de estudio específicos, con un enfoque en validar la funcionalidad, sin profundizar en aspectos de implementación y eficiencia computacional. En general, las evaluaciones reportadas se realizan utilizando arquitecturas de hardware generadas en forma automática o asistida por medio de herramientas de Síntesis de Alto Nivel (*High Level Synthesis*, o HLS), las cuales ofrecen la ventaja de acelerar el desarrollo al permitir diseñar hardware utilizando lenguajes de alto nivel como C/C++, pero presentan la desventaja de generar circuitos que, en la mayoría de los casos, no alcanzan el mismo nivel de optimización y eficiencia que los diseñados manualmente en lenguajes de descripción de hardware (HDL) como VHDL o Verilog.

Al momento de comenzar esta tesis, ninguno de los trabajos reportados en esta línea entregaba detalles de las herramientas y configuraciones utilizadas para generar las arquitecturas de hardware, lo cual permite inferir que, a pesar de reducir considerablemente el tiempo de ejecución con respecto a formulaciones implícitas y explícitas tradicionales para las aplicaciones consideradas, las implementaciones resultantes son computacionalmente ineficientes y díficiles de extender a otras aplicaciones. Con el fin de evaluar la escalabilidad de estas técnicas a problemas de mayor complejidad, es necesario establecer metodologías que integren distintas técnicas y herramientas para el diseño e implementación de controladores especializados, incorporando en el proceso la exploración del espacio de diseño de arquitecturas con distinto nivel de paralelismo y cuantización de señales, con el fin de lograr implementaciones costo-efectivas que balanceen el tiempo de ejecución, uso de recursos, y desempeño del controlador.

Por otro lado, la generación de arquitecturas especializadas para la implementación de redes neuronales en FPGA a partir de descripciones de alto nivel es un problema que ha sido ampliamente estudiado en otros campos de aplicación distintos a la implementación de controladores [23]. Un caso destacable es el campo de instrumentación especializada para experimentos de física de partículas [24], donde se han reportado implementaciones con aceleración por hardware que alcanzan latencias de inferencia en el orden de decenas de nanosegundos con redes estructuralmente similares a las que se han usado para aproximar la ley de control. Actualmente, existen distintos *frameworks* como FINN [25], HLS4ML [24] o el toolbox DLHDL de Matlab [26], los cuales permiten generar descripciones en HDL sintetizables en FPGA a partir de modelos de redes neuronales descritos en alto nivel usando frameworks populares como Tensorflow, Keras o PyTorch, reduciendo significativamente la complejidad asociada a describir sistemas usando abstracciones de *Register Transfer Level* (RTL). La disponibilidad de estas herramientas facilita considerablemente la exploración de arquitecturas de hardware optimizadas para la realización de inferencias con ANNs, y abre nuevas oportunidades para la implementación de controladores MPC aproximados con ANNs que habiliten su uso en sistemas con requerimientos de tiempo real y baja latencia; sin embargo, para aprovechar estas tecnologías en el contexto de MPC, es necesario realizar estudios sistemáticos y evaluaciones empíricas que proporcionen lineamientos sobre los *tradeoffs* involucrados en el proceso de diseño e implementación de controladores especializados.

Este trabajo de tesis tiene como objetivo el desarrollo y validación una metodología integral para la descripción e implementación de aceleradores especializados en FPGA para algoritmos MPC basados en ANNs. Con este fin, se propone la integración de múltiples técnicas y herramientas en un flujo de desarrollo *end-to-end*, abarcando desde el diseño y validación de la red neuronal que aproxime a un controlador de referencia previamente diseñado, hasta la implementación y validación de la red en hardware para la realización de inferencias. Para realizar la interfaz entre software y hardware, el flujo propuesto se basa en el uso de herramientas HLS que actualmente se usan en otros campos de aplicación para facilitar la exploración del espacio de diseño de las arquitecturas de hardware especializadas para los controladores. Por medio del análisis y documentación de casos de estudio representativos de lazos MPC, se espera facilitar la reproducción de los resultados y proveer directrices generales para promover nuevos estudios que permitan extender la aplicabilidad de MPC a sistemas complejos con requisitos de tiempo real con baja latencia.

1.2 Formulación del problema

El proceso típico para implementar aceleradores en FPGA para aproximadores de MPC basados en redes neuronales sigue una serie de pasos estructurados, cada uno empleando técnicas y herramientas especializadas:

- **Generación de conjunto de datos:** Es fundamental que estos datos representen adecuadamente el comportamiento entrada-salida del controlador en todo el rango de operación. En la práctica, es común contar con una implementación funcional del lazo de control descrita en Matlab/Simulink, de la cual se pueden extraer datos de referencia. El desafío es generar una cantidad y diversidad suficiente de datos que permitan al modelo generalizar sin sesgos hacia patrones específicos.
- **Selección y optimización del modelo de red neuronal:** Esta fase abarca desde la definición de la arquitectura del modelo hasta el ajuste de parámetros y validación de hiperparámetros. Frameworks como Tensorflow, Pytorch o Keras proporcionan herramientas de alto nivel que facilitan la creación, manipulación y evaluación de arquitecturas de red arbitrariamente complejas.
- **Cuantización del modelo:** Aunque es una etapa opcional, en esta etapa se convierten pesos y activaciones de precisión flotante a representaciones de menor precisión, como por ejemplo punto fijo con número reducido de bits. El desafío radica en minimizar la pérdida de precisión y rendimiento que puede resultar de esta conversión. Frameworks como Ristretto, QKeras o Brevitas proveen herramientas que permiten ajustar los parámetros de la red considerando representaciones cuantizadas.
- **Diseño del pipeline de procesamiento:** En esta etapa se estructura el flujo de datos entre capas para maximizar el paralelismo de la red en la FPGA. Este paso requiere una exploración cuidadosa del espacio de diseño para balancear el uso de recursos en base a métricas objetivos de latencia y desempeño del controlador. Hoy en día, es común utilizar herramientas HLS para este propósito, las cuales facilitan la generación de aceleradores especializados reduciendo o eliminando la necesidad de describir código RTL; sin embargo, en la práctica se observa que el uso efectivo de las herramientas HLS aún requiere comprender los fundamentos del diseño digital sincrónico [27].
- **Integración del acelerador de hardware:** En la práctica, es habitual que el acelerador de hardware deba integrarse con una CPU en una arquitectura heterogénea. Configurar adecuadamente la comunicación entre CPU y FPGA es crucial para evitar que el tiempo de transferencia de datos limite o anule las mejoras de rendimiento logradas por el hardware especializado.
- **Validación de hardware:** Esta fase evalúa el comportamiento del hardware, generalmente mediante el uso de entornos simulados, que permiten simular la interacción del acelerador con una planta virtual. Herramientas como Matlab/Simulink ofrecen entornos integrados para realizar esta clase de validaciones, facilitando la verificación de la funcionalidad del hardware en condiciones controladas que emulan el sistema final.

Cada uno de los pasos descritos anteriormente opera en distintos niveles de abstracción y requiere enfoques técnicos especializados, normalmente asociadas a distintas subdisciplinas de la Ingeniería Electrónica. La complejidad inherente a cada etapa ha llevado a que la mayoría de los estudios sobre MPC basados en redes neuronales se centren en aspectos de funcionalidad general, demostrando la capacidad de aproximación de las redes en casos específicos y los beneficios potenciales de su implementación en hardware. Sin embargo, estos estudios suelen abordar solo de manera superficial, o incluso omitir, los detalles de implementación en cada etapa, limitando la posibilidad de reproducir y validar experimentalmente sus resultados, lo que dificulta la extensión de estos a otras aplicaciones de diversa complejidad.

Para avanzar en el desarrollo práctico de algoritmos MPC basados en redes neuronales y explotar todas las ventajas de este enfoque, es esencial definir y evaluar directrices que faciliten la integración

de las diversas etapas del flujo de trabajo en el diseño e implementación de estos controladores. Esto abarca desde la descripción funcional del lazo de control hasta su implementación eficiente en hardware especializado, incluyendo la exploración del espacio de diseño y los ajustes necesarios para equilibrar los compromisos entre latencia, uso de recursos y desempeño del lazo de control en la aplicación objetivo.

A partir del contexto planteado previamente, el problema a tratar en esta tesis se formula de la siguiente manera:

"Dado un controlador MPC de referencia, previamente diseñado y validado del cual se puedan extraer un conjunto de datos de actuaciones óptimas para ciertos valores de estado, se requiere diseñar una red neuronal que aproxime la ley de control con un nivel de precisión preestablecido. La red entrenada será implementada en una FPGA, explorándose distintas configuraciones de arquitecturas, considerando nivel de paralelismo y precisión numérica, para así caracterizar el desempeño y eficiencia de la red en base al uso de recursos de cómputo y memoria, error de aproximación en las acciones de control, y latencia del controlador implementado en una plataforma de arquitectura heterogénea que integre CPU y FPGA."

Este enfoque permitirá avanzar hacia la implementación efectiva de controladores MPC basados en ANNs, ofreciendo un marco de trabajo detallado que simplifique y guíe el proceso de integración de todas las etapas de diseño, facilitando así el desarrollo de controladores eficientes para aplicaciones de distinta complejidad.

1.3 Alcances y Contribución

Esta tesis considera los siguientes alcances y restricciones:

A1 En el contexto, se asume la disponibilidad de una descripción en Matlab del lazo de control, previamente validada a nivel funcional, que permite generar datos de referencia para diseñar, entrenar e implementar una ANN que reproduzca el comportamiento de entrada/salida con un margen de error predefinido respecto al controlador de referencia. El análisis sobre la preservación de las garantías teóricas del controlador de referencia [28] queda fuera del alcance de este estudio.

A2 Las estructuras de redes neuronales a utilizar en este trabajo se restringen a redes del tipo *fully-connected*, con función de activación ReLU en las capas ocultas y activación lineal en la salida. Además, el número de capas ocultas y neuronas por capa se limita a una vecindad representativa de las redes reportadas en la literatura reciente [12, 13, 15, 20, 22].

A3 En base a evaluaciones previas a este trabajo, se decidió utilizar la herramienta de código abierto HLS4ML [24] para la generación de descripciones de hardware de modelos de redes neuronales descritos en frameworks de alto nivel. El resto de herramientas y procedimientos en el flujo de diseño propuesto se adaptan a los requerimientos impuestos por HLS4ML.

A4 Las redes implementadas en hardware estarán optimizadas para tareas de inferencia. Como tal, se asume que la estructura y los parámetros de la red se encuentran definidos al momento de realizar la síntesis y no necesitarán actualizarse en tiempo de ejecución. Además, se considera que todos los parámetros de la red implementada en la FPGA deben ser almacenados en memoria on-chip, sin requerir acceso a memoria externa.

A5 Las evaluaciones experimentales de los aceleradores consideran la implementación en plataformas de desarrollo de la familia Zynq de AMD/Xilinx, las cuales integran procesadores ARM con lógica programable. Estas plataformas son escogidas debido a su disponibilidad física y de documentación. No obstante, los resultados obtenidos son aplicables a cualquier las plataformas que incluya una FPGA con los recursos mencionados.

Desde los resultados obtenidos, se identifican las siguientes contribuciones:

- Se establecen directrices sistemáticas para la implementación de redes neuronales en FPGA que guían la exploración del espacio de diseño hacia alternativas de implementación más

favorables para la aplicación, balanceando error de aproximación, latencia y uso de recursos. Estas directrices, aunque se desarrollaron en el contexto de MPC, son generalizables a otras aplicaciones que requieran implementar redes neuronales *fully-connected* en FPGA.

- Se presenta y valida un flujo de diseño *end-to-end* para la implementación de controladores MPC aproximados mediante redes neuronales en FPGA. Este flujo integra herramientas estandarizadas, permitiendo la implementación eficiente de controladores sin necesidad de descripciones manuales en RTL. La metodología propuesta facilita la transición desde el diseño en software hasta la implementación en hardware, reduciendo la complejidad del desarrollo y siendo aplicable a problemas MPC de distinta complejidad, lo que la hace flexible para diversos escenarios y requisitos de implementación.
- Se demuestra que la técnica de aproximación basada en redes neuronales permite alcanzar latencias inferiores a un microsegundo, superando implementaciones tradicionales en la literatura, mientras mantiene un equilibrio entre uso de recursos y desempeño de control.

La documentación asociada y los códigos de referencia que complementan este informe se encuentran disponibles en un repositorio de GitHub [1]. Al momento de completar esta tesis, el repositorio tiene restricciones de acceso (disponible previa solicitud), pero se espera que sea de acceso libre una vez finalizados los proyectos de investigación asociados.

1.4 Organización del Informe

A continuación, se describe la organización del resto de esta tesis:

- El Capítulo 2 presenta una revisión del estado del arte, seguida de los fundamentos conceptuales subyacentes a MPC y redes neuronales que posibilitan la aproximación de la ley de control.
- El Capítulo 3 propone un flujo de diseño de controladores MPC en FPGA basados en redes neuronales.
- El Capítulo 4 muestra los resultados obtenidos de una exploración de optimizaciones de hardware usando HLS4ML, proporcionando directrices de optimización que complementan el flujo de diseño e implementación.
- El Capítulo 5 presenta la aplicación del flujo de diseño a casos de estudio, considerando su implementación en un SoC compuesto de FPGA y CPU.
- El Capítulo 6 resume las principales conclusiones de este trabajo, y propone direcciones posibles para futuros trabajos.

[1] Disponible en: `https://github.com/JuanjoV/fpga-dnn-based-explicit-mpc`

Capítulo 2

Antecedentes

Este capítulo resume conceptos relevantes para contextualizar el trabajo de tesis. Primero, se presentará conceptualmente el control predictivo por modelo (MPC), seguido de una descripción general de las redes neuronales utilizadas para aproximar MPC. Luego, se detallará el funcionamiento de la herramienta HLS4ML para la generación de código en *Register Transfer Level* (RTL) que describe la estructura de redes neuronales, para seguir con algunos conceptos importantes a considerar durante una implementación en FPGA. Finalmente, se presentará una revisión del estado del arte relevante para la implementación en FPGA de MPC aproximado con redes neuronales.

2.1 Control Predictivo por Modelo

El MPC es una estrategia avanzada de control automático que utiliza un modelo matemático de la planta a controlar para predecir su comportamiento futuro, y con eso determinar la secuencia de actuaciones óptimas para una cantidad determinada de instantes futuros, denominada horizonte de predicción. La Figura 2.1 ilustra la idea de MPC para un ejemplo de un sistema de un estado $\mathbf{x}$ y una acción de control $\mathbf{u}$. En el instante de muestreo t, se calcula el valor del estado actual $\mathbf{x}_0$, a partir del cual se determina la acción de control óptima $\mathbf{u}_k$ para llevar al estado a un valor deseado, representado con la línea segmentada amarilla, según una función de costo predeterminada. Utilizando los valores del estado y actuación actual, se determinan los valores de estados y actuaciones para los siguientes N intervalos de muestreo. La línea segmentada verde representa un valor máximo para la acción de control $\mathbf{u}$, el cual queda especificado en tiempo de diseño. Al calcular las acciones de control, el algoritmo MPC debe garantizar que se cumplan estas restricciones. Una vez calculada la secuencia de acciones de control para el intervalo t a lo largo del horizonte de predicción, sólo $\mathbf{u}_0$, la primera de ellas, es aplicada a la planta. En el intervalo de muestreo siguiente, el proceso se repite, resolviéndose un nuevo problema de optimización formulado a partir del nuevo estado actual y con un horizonte desplazado. Esta estrategia se denomina *receding horizon* y es la forma en que los lazos MPC implementan retroalimentación.

A continuación, se presenta cómo se construye el problema de optimización de MPC para un instante t, que permite determinar la acción de control óptima de acuerdo al estado del sistema.

2.1.1. Formulación del problema de optimización

El cálculo de la actuación en MPC depende del modelo dinámico de la planta, el horizonte de predicción de N muestras, una función de costos y las restricciones en las entradas y estados. Para modelos dinámicos lineales, la función de costos es usualmente expresada en términos lineales o cuadráticos, lo que conduce a que el problema resultante se resuelva a través de *Linear Programming* (LP) o *Quadratic Programming* (QP), respectivamente.

En general, la planta a controlar en MPC es modelada como un sistema de estado de tiempo discreto, de la forma

$$\mathbf{x}(t+1) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \tag{2.1}$$

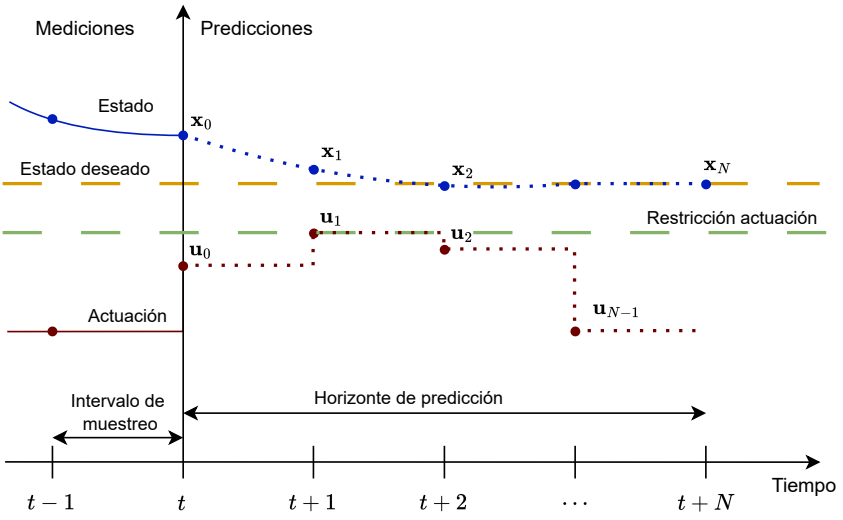


Figura 2.1: Esquema de MPC de un estado y una actuación.

donde en una muestra t, $\mathbf{x}(t) \in \mathbb{R}^{n_x}$ es el vector de n_x estados y $\mathbf{u}(t) \in \mathbb{R}^{n_u}$ es el vector de n_u actuaciones. Luego, las restricciones del sistema pueden expresarse en términos de:

$$\mathbf{x}(t) \in \mathcal{X} \ , \ \mathbf{u}(t) \in \mathcal{U}, \tag{2.2}$$

donde $\mathcal{X} \subseteq \mathbb{R}^{n_x}$ y $\mathcal{U} \subseteq \mathbb{R}^{n_u}$ son conjuntos cerrados que contienen al origen. Asumiendo que se dispone de una medición o estimación completa del vector de estados $\mathbf{x}(t)$ en el instante de tiempo t y que el objetivo del control es llevar este vector de estados al origen, el problema de optimización se plantea de la siguiente forma [1]:

$$\mathcal{P}_N(\mathbf{x}(t)) \ : \min_{\tilde{\mathbf{u}}} \sum_{k=0}^{N-1} l(\mathbf{x}_k, \mathbf{u}_k) + F(\mathbf{x}_N) \tag{2.3a}$$

$$\text{sujeto a } \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k), \quad k = 0, ..., N-1 \tag{2.3b}$$

$$\mathbf{x}_0 = \mathbf{x}(t), \tag{2.3c}$$

$$\mathbf{u}_k \in \mathcal{U}, \quad k = 0, ..., N_u - 1 \tag{2.3d}$$

$$\mathbf{x}_k \in \mathcal{X}, \quad k = 1, ..., N-1 \tag{2.3e}$$

$$\mathbf{x}_N \in \mathcal{X}_N, \tag{2.3f}$$

$$\mathbf{u}_k = \boldsymbol{\kappa}(\mathbf{x}_k), \quad k = N_u, ..., N-1, \tag{2.3g}$$

donde $\tilde{\mathbf{u}} \in \mathbb{R}^{\ell}$, es el vector de las variables de optimización, $\tilde{\mathbf{u}} = [\mathbf{u}_0^\intercal \ ... \ \mathbf{u}_{N_u-1}^\intercal]^\intercal$, $\ell = n_u N_u$, y $l(\mathbf{x}_k, \mathbf{u}_k)$ es la función de costos de los estados futuros y actuaciones, mientras que $F(\mathbf{x}_N)$ representa los costos en el estado final del sistema. En cada instante de tiempo t, $\mathbf{x}_k$ denota la predicción para el tiempo $t+k$ del vector de estados, obtenido aplicando la secuencia de actuaciones $\mathbf{u}_0, ..., \mathbf{u}_{k-1}$ al modelo (2.1), comenzando desde el estado actual $\mathbf{x}_0 = \mathbf{x}(t)$. De la misma manera, $N > 0$ denota el horizonte de predicción; y N_u es el horizonte de actuación, que cumple con $1 \leq N_u \leq N$. Luego, considerando una función de costos cuadrática y un horizonte de predicción N finito, se fijan:

$$l(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_k^\intercal \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^\intercal \mathbf{R} \mathbf{u}_k \tag{2.4}$$

$$F(\mathbf{x}_N) = \mathbf{x}_N^\intercal \mathbf{P} \mathbf{x}_N, \tag{2.5}$$

donde las matrices $\mathbf{Q} \in \mathbb{R}^{n_x \times n_x}$ y $\mathbf{R} \in \mathbb{R}^{n_u \times n_u}$ son los pesos asociados a estados y actuaciones, respectivamente. Además, se incluye la función de costos en estado final $F(\mathbf{x}_N)$ que es definida por la matriz de costos $\mathbf{P} \in \mathbb{R}^{n_x \times n_x}$. Para garantizar que el problema de optimización $\mathcal{P}_N$ planteado en

(2.3) es convexo, las matrices de pesos son elegidas tales que $\mathbf{R}$ es definida positiva, y $\mathbf{Q}$ y $\mathbf{P}$ son semi-definidas positivas [1].

Sea el modelo en tiempo discreto de la planta determinístico y lineal, entonces:

$$\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k. \tag{2.6}$$

Si se asume que $N_u = N$, es decir, que el horizonte de control es igual al horizonte de predicción, entonces se puede descartar la ecuación (2.3g). Además, sean las regiones $\mathcal{U}$, $\mathcal{X}$ y $\mathcal{X}_N$ polítopas definidas por

$$\mathcal{U} = \{\mathbf{u} \in \mathbb{R}^{n_u} : \mathbf{C_u}\mathbf{u} \leq \mathbf{c_u}\} \tag{2.7}$$
$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^{n_x} : \mathbf{C_x}\mathbf{x} \leq \mathbf{c_x}\} \tag{2.8}$$
$$\mathcal{X}_N = \{\mathbf{x}_N \in \mathbb{R}^{n_x} : \mathbf{C_N}\mathbf{x}_N \leq \mathbf{c_N}\}, \tag{2.9}$$

donde $\mathbf{C_x} \in \mathbb{R}^{n_{cx} \times n_x}$ y $\mathbf{c_x} \in \mathbb{R}^{n_{cx}}$ representan las n_{cx} restricciones aplicadas sobre los estados. A su vez, $\mathbf{C_u} \in \mathbb{R}^{n_{cu} \times n_u}$ y $\mathbf{c_u} \in \mathbb{R}^{n_{cu}}$ representan las n_{cu} restricciones aplicadas sobre las actuaciones. Finalmente, $\mathbf{C_N} \in \mathbb{R}^{n_{cN} \times n_x}$ y $\mathbf{c_N} \in \mathbb{R}^{n_{cN}}$ definen las n_{cN} restricciones aplicadas sobre los estados finales.

Entonces, al reemplazar las expresiones desde (2.4) hasta (2.9) en la formulación inicial (2.3), el problema de optimización se convierte en un problema QP con la siguiente forma:

$$\min_{\tilde{\mathbf{u}}} \quad \mathbf{x}_N^\intercal \mathbf{P}\mathbf{x}_N + \sum_{k=0}^{N-1} (\mathbf{x}_k^\intercal \mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^\intercal \mathbf{R}\mathbf{u}_k) \tag{2.10a}$$
$$\text{sujeto a} \quad \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \tag{2.10b}$$
$$\mathbf{C_u}\mathbf{u}_k \leq \mathbf{c_u}, \tag{2.10c}$$
$$\mathbf{C_x}\mathbf{x}_k \leq \mathbf{c_x}, \tag{2.10d}$$
$$\mathbf{C_N}\mathbf{x}_N \leq \mathbf{c_N}, \tag{2.10e}$$
$$\mathbf{x}_0 = \mathbf{x}(t), \tag{2.10f}$$
$$\forall k = 0, \ldots, N-1. \tag{2.10g}$$

2.1.2. Seguimiento de referencia

La formulación de (2.10) puede extenderse para añadir el seguimiento de referencias en el controlador. Dado un vector de salidas de la planta $\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) \in \mathbb{R}^{n_y}$ y una señal de referencia $\mathbf{r}(t) \in \mathbb{R}^{n_y}$, se busca que $\mathbf{y}_\infty \to \mathbf{r}$, donde $\mathbf{y}_\infty$ es la salida en estado estacionario. Para determinar el valor de los estados y actuaciones que permiten llevar la salida al valor de referencia, se reescribe la expresión en estado estacionario para el sistema en (2.6) como:

$$\mathbf{x}_\infty = \mathbf{A}\mathbf{x}_\infty + \mathbf{B}\mathbf{u}_\infty \tag{2.11}$$
$$\mathbf{y}_\infty = \mathbf{C}\mathbf{x}_\infty. \tag{2.12}$$

Entonces, se despejan $\mathbf{x}_\infty$ y $\mathbf{u}_\infty$:

$$\underbrace{\begin{bmatrix} \mathbf{A} - \mathbf{I} & \mathbf{B} \\ \mathbf{C} & \mathbf{0} \end{bmatrix}}_{\mathbf{T}} \begin{bmatrix} \mathbf{x}_\infty \\ \mathbf{u}_\infty \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{r} \end{bmatrix} \tag{2.13}$$

$$\begin{bmatrix} \mathbf{x}_\infty \\ \mathbf{u}_\infty \end{bmatrix} = \mathbf{T}^{-1} \begin{bmatrix} \mathbf{0} \\ \mathbf{r} \end{bmatrix} \tag{2.14}$$

donde $\mathbf{T} \in \mathbb{R}^{(n_x+n_y)\times(n_x+n_u)}$.

Luego, se realiza el cambio de variables:

$$\bar{\mathbf{x}} = \mathbf{x} - \mathbf{x}_\infty \tag{2.15}$$
$$\bar{\mathbf{u}} = \mathbf{u} - \mathbf{u}_\infty \tag{2.16}$$

donde $\bar{\mathbf{x}}$ y $\bar{\mathbf{u}}$ representan la desviación de los estados y actuaciones respecto de los valores necesarios para llevar la salida a su valor de referencia en estado estacionario. Considerando que las variables de desviación satisfacen el mismo modelo dinámico que las variables originales, es necesario actualizar la función de costos (2.10a) por:

$$\bar{\mathbf{x}}_N^\intercal \mathbf{P} \bar{\mathbf{x}}_N + \sum_{k=0}^{N-1} \left(\bar{\mathbf{x}}_k^\intercal \mathbf{Q} \bar{\mathbf{x}}_k + \bar{\mathbf{u}}_k^\intercal \mathbf{R} \bar{\mathbf{u}}_k \right). \tag{2.17}$$

Al igual que la función de costos, es necesario actualizar las restricciones del problema QP en (2.10) aplicando el cambio de variables de la siguiente manera:

- Restricciones de actuación:

$$\mathbf{C_u}(\bar{\mathbf{u}}_k + \mathbf{u}_\infty) \leq \mathbf{c_u} \tag{2.18}$$
$$\mathbf{C_u} \bar{\mathbf{u}}_k \leq \mathbf{c_u} - \mathbf{C_u} \mathbf{u}_\infty \tag{2.19}$$

- Restricciones de estados:

$$\mathbf{C_x}(\bar{\mathbf{x}}_k + \mathbf{x}_\infty) \leq \mathbf{c_x} \tag{2.20}$$
$$\mathbf{C_x} \bar{\mathbf{x}}_k \leq \mathbf{c_x} - \mathbf{C_x} \mathbf{x}_\infty \tag{2.21}$$

- Restricciones de estado final:

$$\mathbf{C_N}(\bar{\mathbf{x}}_N + \mathbf{x}_\infty) \leq \mathbf{c_N} \tag{2.22}$$
$$\mathbf{C_N} \bar{\mathbf{x}}_N \leq \mathbf{c_N} - \mathbf{C_N} \mathbf{x}_\infty \tag{2.23}$$

De esta forma, al resolver el nuevo problema de optimización resultante del cambio de variables, se obtendrá un vector de actuaciones $\bar{\mathbf{u}}$, con el cual se calcula la actuación mediante:

$$\mathbf{u}(t) = \bar{\mathbf{u}} + \mathbf{u}_\infty. \tag{2.24}$$

2.1.3. Formulaciones implícita y explícita

Una de las estrategias para determinar la actuación óptima dado el problema MPC formulado en (2.10) consiste en resolver la optimización en línea, durante la ejecución del controlador. Esta estrategia se denomina MPC implícito. En las formulaciones implícitas de MPC, la acción de control se obtiene en cada intervalo de muestreo mediante algoritmos de optimización numérica empleados *online* para resolver el problema QP (2.10). En la práctica, la complejidad de la formulación implícita radica en el uso de *solvers* capaces de encontrar una solución al problema QP con un cierto margen de error y dentro del límite de tiempo impuesto por la tasa de muestreo del sistema. Existen múltiples *solvers* iterativos eficientes que operan usando operaciones simples de álgebra lineal [29–31]; sin embargo, el número de operaciones por iteración incrementa significativamente con la cantidad de estados, restricciones y horizonte de predicción del problema.

Alternativamente, existe la formulación explícita de MPC, que busca resolver el problema de optimización QP de (2.10) de manera *offline* para cada estado $\mathbf{x}(t)$ posible dentro de un conjunto de estados factibles $\mathcal{X}$. Esta formulación hace explícita la dependencia de $\mathbf{u}(t)$ respecto de $\mathbf{x}(t)$, mediante el uso de técnicas que aproximan la ley de control, como la técnica *multiparametric programming* (mpQP). Estas técnicas dan como resultado una función afín definida por tramos (PWA, por sus siglas en inglés) para $\mathbf{u}(\mathbf{x})$, donde cada tramo se define en una región particular del espacio de estados:

$$\mathbf{u}(\mathbf{x}) = \begin{cases} \mathbf{F}_1 \mathbf{x} + \mathbf{g}_1 & \text{si } \mathbf{x} \in \mathcal{R}_1 \\ \vdots & \vdots \\ \mathbf{F}_\eta \mathbf{x} + \mathbf{g}_\eta & \text{si } \mathbf{x} \in \mathcal{R}_\eta \end{cases}, \tag{2.25}$$

En esta formulación, el espacio definido por $\mathcal{X}$ se divide en η regiones, donde $\mathbf{F}_i \in \mathbb{R}^{Nn_u \times n_x}$ y $\mathbf{g}_i \in \mathbb{R}^{Nn_u}$ definen una función lineal para cada región $\mathcal{R}_i$, las que se definen como:

$$\mathcal{R}_i = \{\mathbf{x} \in \mathbb{R}^{n_x} \ : \ \mathbf{H}_i\mathbf{x} \leq \mathbf{k}_i\} \quad \forall i = 1, ..., \eta, \tag{2.26}$$

donde $\mathbf{H}_i \in \mathbb{R}^{c_i \times n_x}$, $\mathbf{k}_i \in \mathbb{R}^{c_i}$, describen los límites de la i-ésima región. Además, el espacio $\mathcal{X} = \cup_{i=1}^{\eta} \mathcal{R}_i$ cumple con $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$ para $i \neq j$, lo que asegura que cada estado pertenece a una única región y, por lo tanto, tiene asociada una única acción de control.

En el contexto de este trabajo, se asume que se cuenta con un conjunto de datos representativo del espacio de estados del sistema y sus correspondientes actuaciones óptimas. Estas muestras pueden obtenerse a partir de formulaciones implícitas, explícitas, o una combinación de ambas, ya que esencialmente representan la misma ley de control. Además, se asume que estos datos se obtienen antes de diseñar la red que aproximará la ley de control y, dado que la red es agnóstica respecto al origen de los datos, el análisis de las herramientas específicas para las distintas formulaciones de MPC queda fuera del alcance de este trabajo.

2.2 Uso de redes neuronales artificiales para implementación de MPC

Una red neuronal artificial (ANN, del inglés *Artificial Neural Network*) es un tipo de modelo de aprendizaje profundo que consta de múltiples capas de neuronas artificiales interconectadas, diseñadas para representar relaciones de datos mediante el ajuste de sus parámetros internos durante un proceso iterativo conocido como entrenamiento. Una vez concluida la etapa de entrenamiento de la red, esta se utiliza para obtener resultados en su salida, en un proceso llamado inferencia.

En general, se puede considerar una red neuronal como un estimador $\boldsymbol{\mathcal{N}}$ de la forma

$$\mathbf{u} \approx \hat{\mathbf{u}} = \boldsymbol{\mathcal{N}}(\mathbf{x}; \boldsymbol{\theta}) \tag{2.27}$$

donde $\mathbf{u}$ es el valor real de los datos, mientras que $\hat{\mathbf{u}}$ es la aproximación lograda por la red neuronal, $\mathbf{x}$ es la entrada de la red y $\boldsymbol{\theta}$ denota los parámetros ajustables durante el entrenamiento de la red neuronal.

2.2.1. *Fully-Connected Neural Networks*

Las *Fully-Connected Neural Networks*, son un tipo de ANN que se caracteriza por poseer un flujo de información secuencial sin realimentaciones, donde las capas de la red son llamadas capas densas, y están completamente conectadas en cascada. De este modo, una red neuronal de L capas ocultas y M unidades por capa describe una función $\boldsymbol{\mathcal{N}} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ con la siguiente forma [12]:

$$\boldsymbol{\mathcal{N}}(\mathbf{x}; \boldsymbol{\theta}, M, L) = \begin{cases} \mathbf{g}_{L+1} \circ \mathbf{f}_{L+1} \circ \mathbf{g}_L \circ \mathbf{f}_L \circ \cdots \circ \mathbf{g}_1 \circ \mathbf{f}_1(\mathbf{x}) & \text{si } L \geq 2, \\ \mathbf{g}_{L+1} \circ \mathbf{f}_{L+1} \circ \mathbf{g}_1 \circ \mathbf{f}_1(\mathbf{x}) & \text{si } L = 1, \end{cases} \tag{2.28}$$

donde $\circ$ es el operador de composición de funciones, $\mathbf{g} \circ \mathbf{f}(\mathbf{x}) = \mathbf{g}(\mathbf{f}(\mathbf{x}))$, $\mathbf{x} \in \mathbb{R}^{n_x}$ es la entrada a la red neuronal, y $\hat{\mathbf{u}} \in \mathbb{R}^{n_u}$ es la salida. Si $L = 1$, la red neuronal $\boldsymbol{\mathcal{N}}$ se llama *Shallow Neural Network*, mientras que si $L \geq 2$, se denomina *Deep Neural Network* (DNN) [14]. La cantidad de salidas de la red neuronal se define de acuerdo a la cantidad de neuronas de la última capa, por ende la capa $L + 1$ tiene n_u neuronas.

Luego, la función $\mathbf{f}_i$ definida por la i-ésima capa densa de la red es:

$$\mathbf{f}_i(\boldsymbol{\xi}_{i-1}) = \boldsymbol{\theta}_i^W \boldsymbol{\xi}_{i-1} + \boldsymbol{\theta}_i^b \tag{2.29}$$

donde $\boldsymbol{\theta}_i = [\boldsymbol{\theta}_i^W, \ \boldsymbol{\theta}_i^b]$ denota los pesos y bias de la i-ésima capa densa, respectivamente. Además, $\boldsymbol{\xi}_{i-1}$ es la salida de la función de activación de la capa anterior. De esta manera, $\boldsymbol{\xi}_i$ queda definida por:

$$\boldsymbol{\xi}_i = \mathbf{g}_i(\mathbf{f}_i(\boldsymbol{\xi}_{i-1})) \tag{2.30}$$

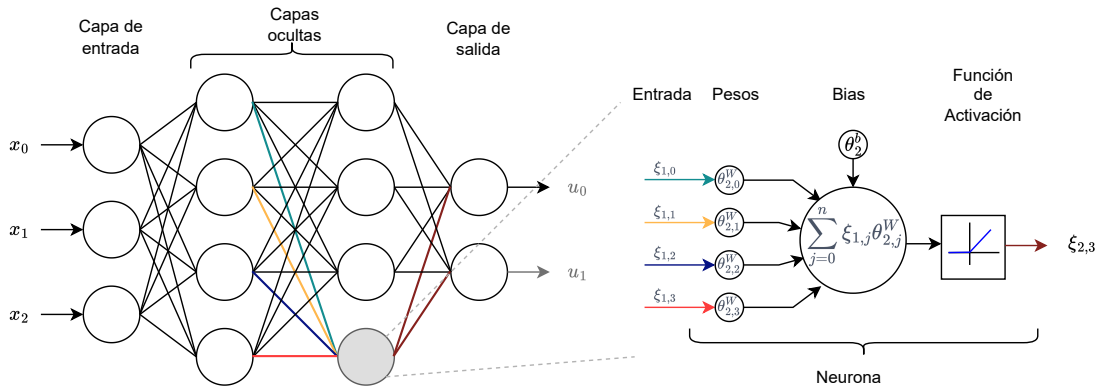


Figura 2.2: DNN de tres entradas, dos salidas, dos capas ocultas y cuatro neuronas por capa.

note que $\boldsymbol{\xi}_0 = \mathbf{x}$ y por consiguiente, $\hat{\mathbf{u}} = \boldsymbol{\xi}_{L+1}$.

Por su parte, la función de activación $\mathbf{g}$ es una función no-lineal que es elegida por el diseñador de la red. En general, para redes modernas se recomienda utilizar la función *Rectified Linear Unit* (ReLU) [14]. Si se considera una ReLU, la función de activación se define como:

$$\mathbf{g}_i(\boldsymbol{f}_i) = \begin{bmatrix} \max(0, \mathbf{f}_{i,0}) \\ \max(0, \mathbf{f}_{i,1}) \\ \vdots \\ \max(0, \mathbf{f}_{i,M}) \end{bmatrix} \tag{2.31}$$

Por lo general, la función de activación $\mathbf{g}$ se aplica de manera uniforme en todas las capas ocultas de la red. No obstante, la elección de la función de activación en la capa de salida varía según el tipo de problema que la red neuronal deba resolver.

La Figura 2.2 muestra una DNN con dos capas ocultas, cada una con cuatro unidades, en la que las salidas de cada neurona están conectadas a todas las neuronas de la capa siguiente. La capa de entrada distribuye las entradas x_0, x_1 y x_2, mientras que las capas ocultas y la capa de salida realizan operaciones de multiplicación y acumulación: cada entrada se multiplica por su respectivo peso, los resultados se suman, se añade un bias, y finalmente se aplica la función de activación ReLU. Además, la Figura 2.2 ilustra el caso específico de la ecuación (2.29) para $i = 2$, donde su salida es solo una de las cuatro entradas que alimentarán las neuronas de la capa de salida.

En general, las ANNs se utilizan para resolver una amplia variedad de tareas. En el contexto de MPC, son especialmente relevantes dos tipos de problemas de *machine learning*: regresión y clasificación. Los problemas de clasificación tienen como objetivo inferir una etiqueta discreta a partir de las entradas, mientras que en los problemas de regresión, el objetivo es inferir un valor continuo a partir de las entradas [32]. En este trabajo, los casos de interés requieren una acción de control continua, por lo que la aproximación de la ley de control se abordará como un problema de regresión.

El tipo de problema que la ANN debe resolver influye en la relación entre la salida de la red, la cantidad de neuronas en la capa de salida y su función de activación. En problemas de regresión, la capa de salida de la ANN contiene tantas neuronas como acciones de control se deban inferir. Para este estudio, se utiliza una función de activación lineal en la capa de salida, $\mathbf{g}_{L+1}(\mathbf{x}) = \mathbf{x}$, ya que simplifica la representación continua de la ley de control y no introduce cálculos adicionales durante la inferencia.

Otro aspecto importante a considerar al diseñar una red neuronal es la elección de la función de pérdida, la cual se utiliza para evaluar el desempeño de la red y ajustar correctamente sus parámetros durante el entrenamiento. Según el tipo de problema, suelen emplearse diferentes funciones de pérdida. En el caso de problemas de regresión, es habitual el uso de funciones que cuantifican

directamente el error de aproximación de la red, como el *Mean Squared Error* (MSE), definido como

$$\mathcal{L}(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\mathcal{N}}(\mathbf{x};\boldsymbol{\theta}), \mathbf{u}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{u}_i - \hat{\mathbf{u}}_i)^2 \tag{2.32}$$

donde N es el tamaño de los vectores $\mathbf{u}$ y $\hat{\mathbf{u}}$, que representan el valor real y la estimación de la red, respectivamente.

Uno de los desafíos comunes al diseñar redes neuronales es lograr que generalicen correctamente las relaciones aprendidas durante el entrenamiento. En esta fase, la red tiene acceso a un conjunto de datos específico y su desempeño se evalúa mediante una función de costo, que cuantifica el error de las estimaciones realizadas. A medida que el entrenamiento avanza, los pesos de la red se ajustan para minimizar el error de aproximación. No obstante, el verdadero objetivo es que el desempeño de la red se mantenga consistente al enfrentarse a datos nuevos y no vistos previamente, lo que se conoce como error de generalización o error de *test*.

Dado que no es posible disponer de datos que cubran todo el espacio de entradas posibles, es inviable usar todos los casos en el entrenamiento. Por ello, se emplean conjuntos de datos que representen adecuadamente el conjunto total de entradas, asumiendo que los datos de entrenamiento y los de prueba están distribuidos de manera idéntica. Así, se busca (i) disminuir el error de entrenamiento y (ii) minimizar la diferencia entre el error de entrenamiento y el de test.

De manera informal, se denomina capacidad de una red neuronal a su habilidad para ajustarse a una amplia variedad de funciones. Una red con baja capacidad tendrá dificultades para ajustarse al conjunto de entrenamiento, mientras que una red con una capacidad demasiado alta tenderá a ajustarse en exceso a los datos de entrenamiento, lo que aumenta el error en los datos de test, un fenómeno conocido como *overfitting*. Una técnica ampliamente utilizada para mitigar el *overfitting* es la regularización [14, 32], que consiste en añadir términos a la función de costo que penalizan directamente los pesos $\boldsymbol{\theta}$ ajustados durante el entrenamiento.

Las configuraciones definidas antes del entrenamiento de una red neuronal, como la cantidad de neuronas por capa, el número de capas, la función de activación y los parámetros de regularización, se denominan hiperparámetros de la red. Para evitar ambigüedad entre los conceptos de "arquitectura de hardware" y "arquitectura de red", en este trabajo se utilizará el término "hiperparámetros" exclusivamente para referirse al número de capas ocultas (L) y al número de neuronas por capa (M).

2.2.2. Aproximación de la ley de control con redes neuronales

Gracias a su estructura jerárquica, las redes de tipo DNN pueden aproximar funciones complejas mediante la composición alternada de funciones afines y no lineales. El uso de capas ReLU permite que la red capture adecuadamente las no linealidades presentes en una función definida por tramos afines, como lo es en el caso de la ley de control MPC con la estructura mostrada anteriormente en (2.25).

En relación con este tema, [15] analiza la capacidad de aproximación de redes con activaciones ReLU, determinando una cota inferior para el número máximo de regiones lineales de la forma presentada en (2.25) que puede aproximar una red neuronal con n_x entradas, L capas ocultas con activación ReLU y $M \geq n_x$ unidades por capa. Esta cota inferior está dada por

$$\left(\prod_{l=1}^{L-1}\left\lfloor\frac{M}{n_x}\right\rfloor^{n_x}\right)\sum_{j=0}^{n_x}\binom{L}{j} \tag{2.33}$$

La Figura 2.3 ilustra, en escala logarítmica, el crecimiento de la cantidad de regiones representables a medida que aumenta el número de unidades por capa de la red neuronal, según la expresión (2.33). También, se observa que la cantidad de regiones representables incrementa significativamente al añadir capas ocultas, especialmente en redes con un mayor número de neuronas por capa. El análisis de la expresión (2.33) permite estimar el orden de magnitud de la red neuronal necesaria para aproximar una ley de control dada, considerando su cantidad de regiones. Aunque esta expresión proporciona solo una cota inferior del número máximo de regiones representables, es

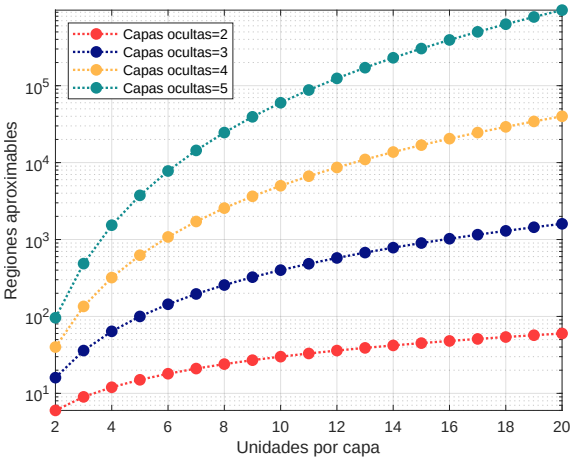


Figura 2.3: Crecimiento de las regiones representables respecto de la cantidad de unidades por capa para una red neuronal de $n_x = 1$ y $n_u = 1$, según la expresión (2.33).

útil para establecer directrices sobre los hiperparámetros a tener en cuenta al aproximar la ley de control.

Más recientemente, en [12], los autores extienden el análisis respecto de la capacidad de representación de las redes neuronales y proponen un método para obtener una representación exacta de la ley de control a través del uso de $2n_u$ redes neuronales. Sin embargo, en la aplicación de este trabajo no es estrictamente necesario contar con una representación exacta de la ley de control en la red neuronal, puesto que en los procesos posteriores, que serán descritos en el Capítulo 3, se introducirá inevitablemente cierto error. El objetivo es, más bien, mantener dicho error dentro de un rango acotado.

2.2.3. Cuantización de redes neuronales

La cuantización de redes neuronales consiste en utilizar representaciones numéricas de menor precisión, como enteros o punto fijo, en lugar de representaciones en punto flotante de alta precisión. Este proceso normalmente permite reducir el número de bits necesarios para almacenar y procesar los parámetros y activaciones de la red, en comparación a las representaciones nominales en punto flotante de 32 o 64 bits. La cuantización de redes ha ganado popularidad en los últimos años debido a su potencial para reducir el espacio requerido en memoria para almacenar los modelos y mejorar la eficiencia computacional de las operaciones, permitiendo acelerar la ejecución durante la inferencia y habilitar la ejecución de redes neuronales complejas en dispositivos con recursos limitados.

El uso de redes neuronales cuantizadas ha mostrado ser altamente efectivo en tareas de clasificación de imágenes. Debido a la naturaleza discreta de las salidas esperadas en las tareas de clasificación, el error de aproximación (la diferencia entre los valores originales en alta precisión y los valores cuantizados) en las operaciones intermedias no se traduce directamente a una degradación en la precisión de la clasificación [16]. Sin embargo, en tareas de regresión, como se requiere para la aproximación de leyes de control, el efecto de la cuantización sobre la precisión de salida no ha sido exhaustivamente estudiado. En este contexto, resulta necesario evaluar si las representaciones cuantizadas permiten mantener el error de inferencia dentro de límites aceptables para no comprometer el desempeño del lazo de control.

Una estrategia eficaz para reducir el impacto del error de cuantización en el error de inferencia es el entrenamiento cuantizado. Este enfoque implica entrenar la red directamente con pesos, bias y activaciones en precisión reducida, de modo que los parámetros se ajusten considerando las limitaciones impuestas por la cuantización. En la práctica, el proceso de selección de la precisión en bits para la cuantización suele ser iterativo y se basa en pruebas empíricas. Herramientas como QKeras [33] permiten diseñar y entrenar redes con especificaciones de precisión reducida, entregando control sobre el número de bits utilizados en cada capa de la red.

En aspectos de implementación de los modelos de redes cuantizados, las FPGAs ofrecen control

fino sobre la configuración del hardware de inferencia, permitiendo ajustar el número de bits necesarios para parámetros y operaciones. Si bien estas características de configuración de hardware ofrecen gran potencial para el desarrollo de lazos MPC computacionalmente eficientes, el uso de representaciones cuantizadas para aproximar la ley de control no ha sido explorado en la literatura reciente.

2.3 Implementación de redes neuronales en FPGA

En esta sección se presentan aspectos relevantes para la implementación de redes neuronales en FPGA, desde las herramientas de generación de código de alto nivel hasta los detalles de las plataformas físicas a considerar para la implementación.

2.3.1. Herramientas de alto nivel para implementar redes neuronales

La implementación de redes neuronales en FPGA es un desafío que ha sido ampliamente estudiado en aplicaciones con restricciones de tiempo fuera del contexto de MPC, como sistemas de visión por computador [34,35], reconocimiento de voz [36] y mediciones en física de partículas [24,33].

De acuerdo con la arquitectura de hardware del acelerador de redes, el trabajo en [25] identifica tres enfoques principales para implementar una red neuronal en FPGA:

- **Un motor de procesamiento único** [35, 37], típicamente implementado como un *systolic array* diseñado para acelerar las operaciones de multiplicación y acumulación, lo que permite procesar cada capa de la red de manera secuencial.
- **Arquitecturas de *streaming*** [24, 33, 38], en las que cada capa tiene su propio motor de procesamiento, y las salidas de una capa fluyen directamente como entradas a la siguiente, permitiendo el procesamiento en paralelo a cambio de un mayor uso de recursos.
- **Un procesador vectorial** [34,36], que incorpora instrucciones específicas para ejecutar operaciones propias de varias capas de red, acelerando no solo las multiplicaciones y acumulaciones, sino también las funciones de activación y capas de agrupamiento (*pooling*).

El creciente interés en utilizar ANNs para diversas aplicaciones ha impulsado el desarrollo de herramientas dedicadas para su implementación en FPGA, facilitando así el abordaje de distintos enfoques de implementación. En esta línea, el trabajo reportado en [39] presenta una revisión de herramientas de alto nivel para la implementación de redes neuronales en FPGA, clasificándolas dos categorías: *Overlay* y *Dedicated*.

Los *frameworks* de tipo *Overlay*, como MATLAB DLHDL [26], Vitis AI [40], y Vectorblox [41], generan uno o más coprocesadores programables a partir del modelo de red neuronal en software. Este enfoque se centra en acelerar operaciones específicas de la red neuronal, permitiendo que el coprocesador ejecute los cálculos básicos de cada capa, mientras el procesador principal o *host* coordina el flujo general de la inferencia. La interacción entre el *host* y el coprocesador es posible gracias a librerías propietarias provistas por la herramienta utilizada, las cuales distribuyen la carga de procesamiento entre ambos y facilitan el soporte de una mayor variedad de tipos de capa en comparación con el enfoque *Dedicated* [39].

Por otro lado, los *frameworks* de tipo *Dedicated*, como HLS4ML [24] y FINN [25], generan un módulo de *Intellectual Property* (IP) que contiene una arquitectura específica diseñada para ejecutar el algoritmo de la red neuronal, sin necesidad de un procesador *host* para realizar la inferencia. Este enfoque tampoco depende de librerías propietarias, lo que significa que la comunicación del IP con el resto del sistema debe ser resuelta manualmente por el diseñador.

La elección del *framework* para implementar la red neuronal influye directamente en la arquitectura resultante, así como en la latencia y el uso de recursos del diseño final [39]. En base a evaluaciones preliminares de las herramientas disponibles al momento de iniciar este trabajo de tesis, se determinó utilizar el *framework* HLS4ML para la generación de descripciones de hardware a partir de modelos en alto nivel de redes neuronales. Según las clasificaciones presentadas, HLS4ML corresponde a un *framework* de tipo *Dedicated*, siendo el acelerador generado con HLS4ML una

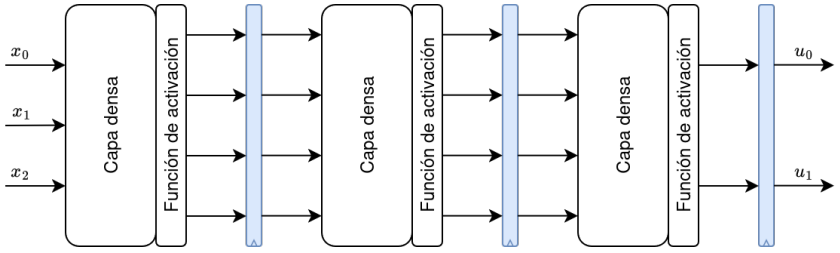


Figura 2.4: Vista de alto nivel de la implementación con HLS4ML de la ANN de la Fig. 2.2.

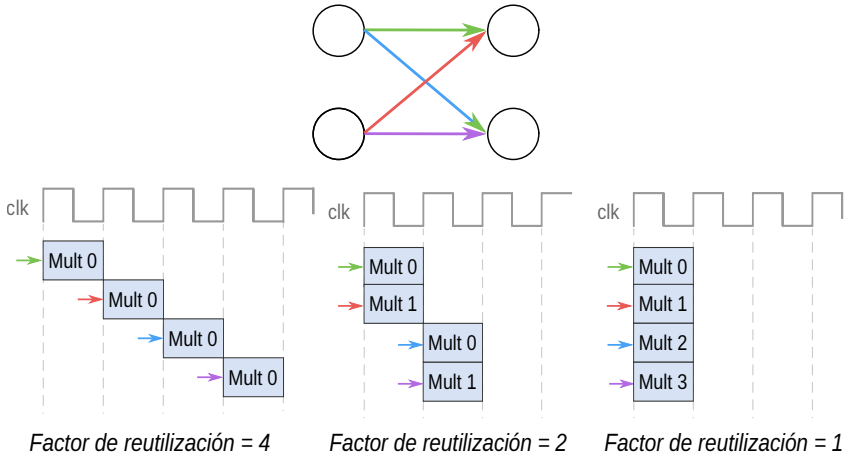


Figura 2.5: Factor de reutilización para una capa densa en HLS4ML.

arquitectura de *streaming*. Dentro de las alternativas evaluadas, HLS4ML presenta una mayor disponibilidad de documentación, ejemplos de uso reproducibles, desarrollo activo con actualizaciones recientes, y soporte para las plataformas de hardware disponibles para realizar este estudio.

2.3.2. HLS4ML

HLS4ML es una biblioteca de Python diseñada para facilitar la generación de descripciones de hardware optimizadas para FPGA a partir de modelos de ANNs desarrollados en *frameworks* populares como Keras o PyTorch. Específicamente, HLS4ML toma como entrada un modelo de red ya entrenado junto con un archivo de configuración que define parámetros de implementación para la arquitectura de hardware, como el nivel de paralelismo en las operaciones y la precisión numérica (cuantización) en cada capa de la red. A partir de esta información, la herramienta produce código en C/C++ con directivas de optimización debidamente adaptado para ser procesado por una herramienta de *High Level Synthesis* (HLS). Una de las principales características de HLS4ML, es que genera una arquitectura de tipo *streaming*, donde las señales internas del modelo se propagan a través de etapas de procesamiento o *pipeline*, como muestra la Figura 2.4, donde se puede ver que cada etapa de procesamiento corresponde a una capa con su respectiva función de activación. Además, la salida de cada etapa está registrada, lo que se denota en la figura con un rectángulo azul.

En términos de configuración, las dos opciones más relevantes para la generación de arquitecturas de hardware con HLS4ML incluyen el factor de reutilización y el nivel de cuantización de los parámetros en cada capa de la red, estas opciones permiten configurar el nivel de paralelismo y balancear el uso de recursos para el mapeo al hardware.

El parámetro llamado factor de reutilización indica cuántas veces se utiliza un mismo multiplicador dentro de la capa de la red. La Figura 2.5 presenta un ejemplo de una capa densa que requiere realizar cuatro multiplicaciones, ilustradas con flechas de colores. Asumiendo que cada multiplicación puede ejecutarse en un ciclo de reloj, se consideran los siguientes casos:

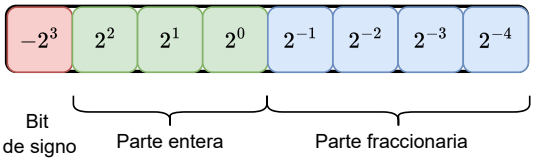


Figura 2.6: Representación en punto fijo de 8 bits, con 3 bits para parte entera.

- Cuando el factor de reutilización es igual a cuatro, un solo multiplicador se emplea de manera secuencial para completar las cuatro operaciones en cuatro ciclos de reloj, maximizando el ahorro de recursos a expensas de un mayor tiempo de cómputo.

- Con un factor de reutilización de dos, se utilizan dos multiplicadores en paralelo, permitiendo realizar dos multiplicaciones en cada ciclo y completando la operación de la capa en dos ciclos de reloj.

- Finalmente, cuando el factor de reutilización es igual a uno, se emplean cuatro multiplicadores independientes, permitiendo que todas las multiplicaciones se realicen simultáneamente en un solo ciclo de reloj. Esto maximiza la velocidad de cómputo, aunque requiere más recursos.

El factor de reutilización en HLS4ML puede definirse de manera independiente para cada capa, lo que facilita el control preciso del nivel de paralelismo en puntos específicos de la red neuronal. Sin embargo, cada capa tiene sus propios multiplicadores asignados, y estos no se comparten con las capas siguientes. Así, un factor de reutilización alto reduce los recursos en una capa específica, pero no implica que esos recursos se puedan reutilizar en las capas posteriores.

Por otro lado, la directiva de cuantización permite equilibrar el uso de recursos y la precisión de la red mediante representaciones numéricas de punto fijo. A diferencia de la representación en punto flotante, el punto fijo representa números con una cantidad fija de bits para la parte entera y para la parte fraccionaria. Por ejemplo, la Figura 2.6 muestra una representación en punto fijo de 8 bits, con 1 bit de signo (rojo), 3 bits para la parte entera (verde) y 4 bits para la parte fraccionaria (azul). Con esta configuración, la representación final resulta de sumar las potencias de 2 correspondientes a las posiciones con un "1". Así, el rango total de valores posibles para el caso de la Figura va desde -8 (en binario, 1000 0000) hasta 7.9375 (en binario, 0111 1111). En HLS4ML, las representaciones de punto fijo se definen mediante los parámetros W y Q, donde W indica la cantidad total de bits utilizados en la representación, y Q representa los bits destinados a la parte entera junto con el bit de signo, siendo para el ejemplo de la Figura 2.6 $W = 8$ y $Q = 4$. Además, HLS4ML es compatible con QKeras, lo que permite conservar la cuantización del entrenamiento del modelo sin alterar los parámetros y el error de aproximación.

Además del factor de reutilización y el nivel de cuantización, HLS4ML también permite seleccionar una estrategia de optimización a nivel de descripción en C/C++ para las capas de la red neuronal. Las opciones disponibles son *Latency* y *Resources*, siendo *Latency* la estrategia predeterminada.

El código C/C++ generado por HLS4ML es procesado posteriormente en una herramienta de HLS para simular y verificar su equivalencia funcional utilizando una *golden reference*, que contiene una secuencia de datos de entrada y su correspondiente salida esperada, obtenidas del modelo original. Luego, el código simulado pasa por el proceso de síntesis de alto nivel para generar una representación en HDL, cuya funcionalidad se verifica mediante una cosimulación, un proceso que permite comparar el comportamiento de la descripción en C/C++ con el de la implementación en HDL para asegurar su equivalencia. Tras la validación del código HDL, se realizan los procesos de síntesis lógica e implementación para producir un módulo en formato IP-Package, integrable en un sistema con módulos adicionales, o un bitstream para la configuración directa de la FPGA.

En resumen, HLS4ML proporciona una interfaz que facilita la síntesis del diseño y permite ajustar su configuración de forma iterativa, reduciendo significativamente el tiempo de prototipado de hardware en comparación con el diseño manual de arquitecturas en RTL. Estas características hacen de HLS4ML una herramienta útil para la exploración de aceleradores de redes neuronales en hardware.

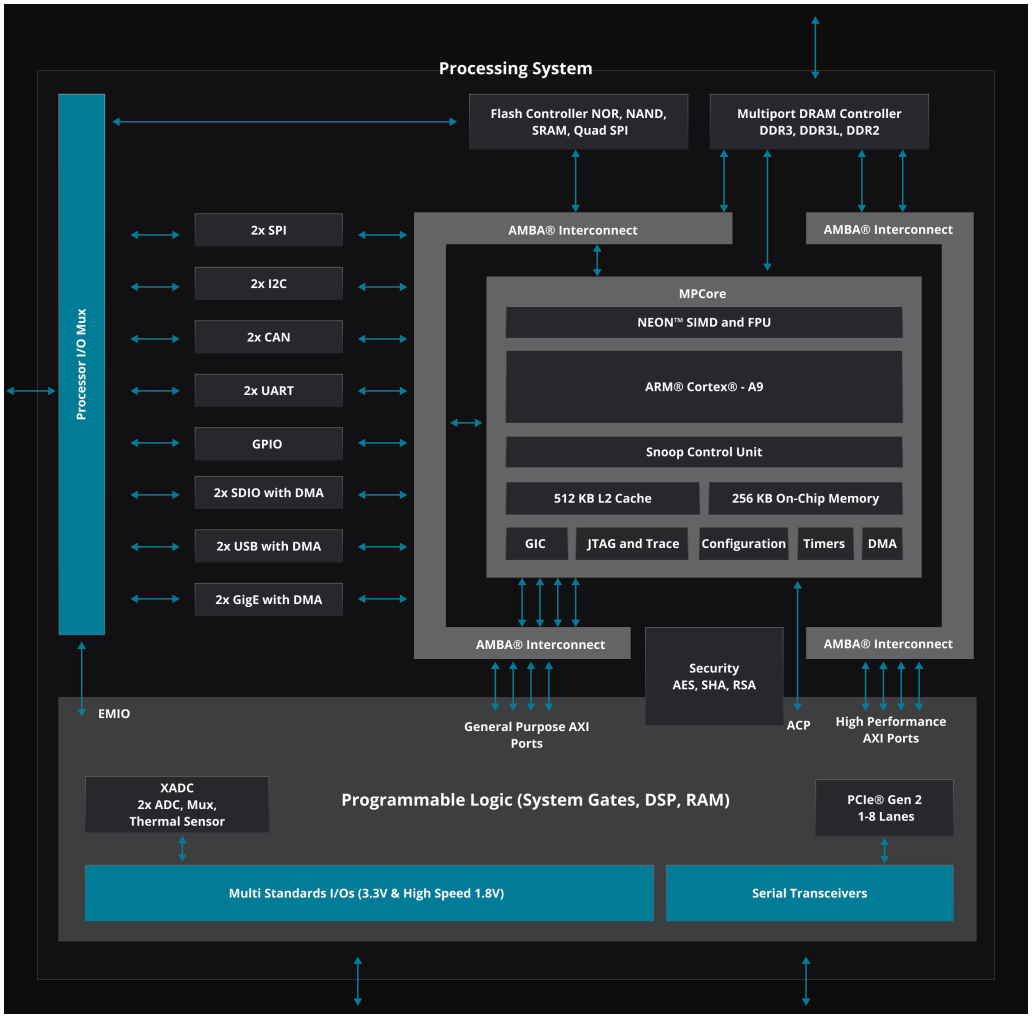


Figura 2.7: Diagrama de un sistema heterogéneo ZYNQ 7000 de AMD/Xilinx [43].

2.3.3. Sistemas heterogéneos basados en FPGA

Un sistema heterogéneo es aquel que está compuesto de dos o más unidades de procesamiento de distinto tipo. En particular, para el contexto de este trabajo se considera el sistema heterogéneo compuesto principalmente de una unidad de procesamiento general (CPU) y una unidad de lógica programable (FPGA). El uso de sistemas heterogéneos responde a un escenario general, donde se requiere liberar al CPU de carga de cómputo, o bien, acelerar determinadas operaciones, en ambos casos a través del uso de la FPGA. Dentro de los dispositivos modernos, son populares los *System-on-Chip* (SoC) que juntan CPU y FPGA en un mismo circuito integrado, como los de la familia Zynq de AMD [4, 20, 42, 43].

Para la aplicación de MPC, el CPU actúa como *host*, derivando carga computacional al acelerador en la FPGA, que actúa como *device*. En este modelo, *host* y *device* tienen espacios de memoria separados, por lo que los datos deben copiarse a la memoria del *device* antes de procesarse y luego devolverse al *host* una vez finalizado el cálculo. Este esquema heterogéneo permite distribuir la carga computacional, combinando la capacidad de procesamiento general del CPU con la aceleración de tareas específicas en la FPGA mediante paralelización.

Un aspecto crítico al implementar sistemas heterogéneos, como los empleados en este trabajo, radica en la eficiencia de la comunicación entre la CPU y la FPGA. El tiempo necesario para la transferencia de datos entre unidades de procesamiento independientes puede convertirse en un cuello de botella que limite la aceleración efectiva del sistema. En aplicaciones como MPC, por ejemplo, cuando la estimación de estados se realiza en la CPU, el cálculo de control en la FPGA, y

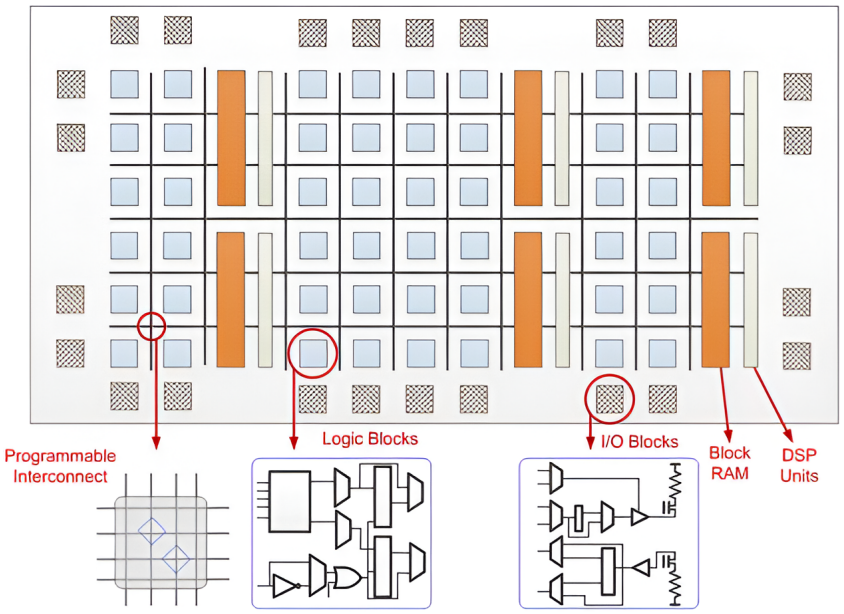


Figura 2.8: Estructura interna de una FPGA [43].

la ejecución de la acción de control en la CPU, el proceso de escribir los estados en la FPGA y leer los resultados de vuelta en la CPU introduce un retardo fijo en el tiempo de ejecución del lazo de control. En caso de no ser debidamente caracterizado y gestionado, este retardo puede neutralizar los beneficios de la aceleración en tareas de alta demanda computacional.

2.3.3.1. Arquitectura y componentes de una FPGA

La arquitectura de una FPGA se basa en bloques y conexiones configurables que permiten implementar descripciones de hardware definidas por el usuario, habitualmente en RTL. Estas descripciones son traducidas a una configuración específica que define cómo se conectan y operan los recursos internos del dispositivo. La Figura 2.8 muestra, de forma simplificada, el interior de una FPGA, donde una gran cantidad de componentes e interconexiones se disponen en una malla dentro del circuito integrado. En general, las FPGA contienen diferentes cantidades de componentes básicos, pudiendo tener desde unos pocos miles hasta cientos de miles, según su gama.

La descripción detallada de la arquitectura y las componentes disponibles en una FPGA moderna escapa del foco de este trabajo. Sin embargo, a continuación se provee una breve descripción de las componentes principales y terminología que se usará para caracterizar el uso de recursos y entender los reportes de eficiencia que se usarán en el resto de este documento:

- **Bloques de lógica configurable (Logic Blocks, o CLBs):** constituyen la unidad lógica básica de una FPGA. Los CLBs se organizan en forma de grilla dentro del dispositivo y permiten implementar funciones lógicas. Cada CLB contiene una cantidad específica de componentes lógicos, incluidos *look-up tables* (LUTs) y flip-flops (FFs).
- **Interconexión (Programmable Interconnect):** es una red de conexiones configurables que permite interconectar los distintos componentes de la FPGA.
- **Bloques Input/Output (I/O Blocks o IOBs):** Proveen la electrónica necesaria para conectar el interior de la FPGA con el exterior.
- **Digital Signal Processors (DSPs):** Son unidades especializadas que permiten realizar operaciones matemáticas de manera más eficiente que los CLBs.
- **Block Random Access Memory (BRAM):** es la memoria integrada de la FPGA. Estos bloques permiten almacenar grandes cantidades de datos (del orden de kilobits) y pueden combinarse o dividirse según los requisitos de diseño.

Para los efectos de este trabajo, los bloques DSP son de particular importancia, ya que están directamente relacionados con las operaciones de multiplicación, que son predominantes en las redes utilizadas. Adicionalmente, se consideran como recursos de interés los bloques de BRAM y los CLBs, que considerarán divididos en FFs y LUTs, dado que la herramienta de generación de hardware los contabiliza por separado. Respecto de las interconexiones, no se tiene control directo sobre ellas, ya que su uso forma parte de las decisiones de la misma herramienta. De este modo, el uso de recursos de la FPGA se reportará en este trabajo en términos de LUTs, FFs, BRAM y DSPs.

2.4 Trabajos relacionados

La revisión bibliográfica realizada para este trabajo de tesis se organiza en torno a tres factores asociados a mejorar la eficiencia computacional de lazos MPC, con el objetivo de extender su aplicación a sistemas que requieren operación en tiempo real y baja latencia:

- El uso de FPGAs como plataforma de aceleración para operaciones demandantes en algoritmos MPC con formulaciones implícitas y explícitas clásicas.
- El uso de redes neuronales como abstracción para aproximar la ley de control.
- El uso de FPGAs para implementar controladores aproximados basados en redes neuronales.

2.4.1. Implementaciones de MPC en FPGA

En primer lugar, se evalúan reportes de implementaciones de MPC en FPGA tanto para formulaciones implícitas como explícitas, con el objetivo de identificar las métricas de interés dentro de la comunidad de control y reconocer problemáticas comunes en la implementación de MPC en FPGA que también puedan surgir en este trabajo.

En [17], los autores presentan una revisión detallada de trabajos que reportan implementaciones de aceleradores FPGA, enfocándose en tareas computacionalmente demandantes en *solvers* QP empleados para MPC implícito hasta el año 2018. En esta revisión, se enfatiza que el cálculo asociado al *solver* QP para determinar la actuación óptima representa el principal cuello de botella de los algoritmos MPC. En el reporte, se muestra que los *solvers* basados en métodos de primer orden, como *Fast Gradient Method* (FGM) o *Alternating Direction Method of Multipliers* (ADMM), cuya demanda computacional se concentra en operaciones de álgebra lineal, resultan particularmente adecuados para implementarse en hardware. Consecuentemente, los trabajos más recientes reportados en ese entonces concentraban sus esfuerzos en arquitecturas de hardware para acelerar la ejecución de operaciones como el producto punto, reportándose latencias de resolución en el orden de decenas de microsegundos para el algoritmo ADMM en casos de estudio específicos de dimensión moderada.

Cabe destacar, que todas las implementaciones revisadas en [17] se basaban en descripciones RTL escritas manualmente. Estas descripciones de bajo nivel entregaban un control fino sobre la configuración de hardware, permitiendo explotar simplificaciones específicas de cada caso de estudio para reducir el tiempo de cómputo de las operaciones más demandantes del *solver*. Sin embargo, las evaluaciones se restringen a simulaciones y no presentan evidencia de integración del acelerador de optimización a nivel del algoritmo de control. Además, tanto el nivel de especialización de la arquitectura de hardware como la falta de detalles de implementación hacen que las soluciones propuestas resulten difíciles de reproducir y extender a otras aplicaciones o problemas de distinta escala.

Posteriormente, el desarrollo de herramientas de HLS ha simplificado el proceso de diseño de aceleradores de hardware, al permitir generar descripciones de arquitecturas de hardware a partir de descripciones en lenguajes de alto nivel como C/C++. A partir de las descripciones funcionales de alto nivel, es posible explorar rápidamente diferentes arquitecturas por medio del uso de pragmas o directivas de optimización, las que permiten, por ejemplo, controlar el paralelismo, representación numérica, particiones de memoria e interfaces de comunicación entre módulos. El uso de herramientas de HLS para aceleradores de MPC implícito fue explorado inicialmente en [44]. Desde entonces, el uso de HLS enfocado en el *solver* del problema QP ha sido aplicado a algoritmos

como *quadprog* [31] y ADMM [45–47]. Recientemente, el uso eficiente de directivas de optimización en HLS ha sido estudiado en mayor detalle [46, 47], derivándose directrices generales para aplicar eficientemente opciones de paralelización en ADMM y reducir el tiempo de ejecución de lazos de MPC implícito.

En cuanto a MPC explícito, se han desarrollado diferentes estrategias para habilitar su implementación en FPGA, considerando tanto herramientas de HLS como codificación a bajo nivel. Varias de estas estrategias permiten aplicar MPC explícito mediante una aproximación de la ley de control [7], estableciendo un compromiso entre el desempeño del controlador, el tiempo de ejecución y el uso de recursos. En términos de tiempo de ejecución, los controladores explícitos presentados en [7] y [4] presentan latencias en el orden de las centenas de nanosegundos (ns). Sin embargo, ambos problemas son de baja dimensionalidad (por ejemplo, tres estados, horizonte de cinco muestras, dos restricciones), y no presentan un análisis sobre como las implementaciones escalan a problemas de mayor dimensionalidad.

De acuerdo con lo revisado, se concluye que las herramientas de HLS han ganado relevancia en aplicaciones de MPC en FPGA, ya que facilitan el diseño de aceleradores para algoritmos en lenguajes de alto nivel sin la necesidad de escribir código en HDL manualmente. Además, se dispone de datos de latencia cuyos órdenes de magnitud se utilizarán como referencia para comparar las implementaciones logradas a través del flujo propuesto en el siguiente capítulo. Finalmente, además de la latencia del controlador las implementaciones revisadas son evaluadas mediante el uso de recursos, dividido en LUTs, FFs, BRAMs y DSPs.

2.4.2. Aproximación de MPC con redes neuronales

La revisión de literatura en este aspecto apunta a identificar el tipo de red neuronal comúnmente utilizado para aproximar la ley de control y las condiciones más adecuadas para su entrenamiento. Además, se apuntó a identificar métricas de desempeño y buenas prácticas empleadas en la aplicación de redes neuronales en el contexto de MPC desde el punto de vista analítico y algorítmico, sin necesariamente abordar aún los aspectos de implementación en hardware.

Entre las topologías de ANNs empleadas para aproximar la ley de control, las redes fully-connected con activaciones ReLU son ampliamente utilizadas [12, 13, 22, 48–51], principalmente debido a que representan una función definida por tramos, lo cual resulta similar a la ley de control que se obtendría mediante una formulación explícita exacta [15]. Complementariamente, en [12], los autores derivan los requisitos que una DNN con activaciones ReLU debe cumplir para lograr una representación exacta de una ley de control lineal, proporcionando así las mismas garantías teóricas que un controlador explícito. Otros autores han optado también por utilizar DNNs con activaciones tanh, dado que esta función también proporciona una capacidad de aproximación universal de funciones [19, 52, 53]. Dado que tanto ReLU como tanh son funciones de activación ampliamente efectivas en aplicaciones de MPC, en este trabajo se optará exclusivamente por el uso de activaciones ReLU para las redes que aproximarán la ley de control.

Dado que el conjunto de datos de entrenamiento solo corresponde a una muestra representativa del espacio de estados, los controladores aproximados basados en ANNs presentan la limitación de no poder asegurar el cumplimiento de garantías teóricas de desempeño en todo el rango de operación. Esto incluye el cumplimiento de restricciones, la estabilidad en estado estacionario y la optimalidad de la actuación [28]. Para abordar esta limitación, los autores de [13] presentan un método para evaluar y ajustar una red con activaciones ReLU para garantizar la estabilidad y el cumplimiento de restricciones. En [48] se propone una DNN con activaciones ReLU que aproxima la ley de control, donde se derivan garantías estadísticas de cumplimiento de restricciones y estabilidad para la aplicación específica. Otros estudios, han propuesto métodos para garantizar estas propiedades, como el uso de capas basadas en operaciones de optimización [49, 54]. Sin embargo, estos métodos requieren resolver un problema de optimización durante la inferencia, lo que implica una carga computacional significativa en comparación con las operaciones de capas densas. Como alternativa, [52] introduce un algoritmo que asegura el cumplimiento de las restricciones de actuación descartando las actuaciones inviables generadas por la red.

Otros autores han flexibilizado el cumplimiento de las restricciones, enfocándose en maximizar el grado de representatividad de la ley de control en los datos de entrenamiento para, de esta manera, acotar el grado de incumplimiento de las restricciones de actuación. En este contexto, la

Tabla 2.1: Resumen de implementaciones de MPC con redes neuronales en FPGA.

Paper	**Hiperparámetros de red** ($L \times M$)	**Función de activación**	**Precisión numérica**	**Latencia** [μs]
[19]	5×10	ReLU	32 bits punto flotante	2.1
[19]	5×10	ReLU	16 bits punto fijo	1
[20]	2×5	—	24 bits punto fijo	—
[21]	2×100	ReLU	—	126
[22]	7×256	ReLU	—	—
[53]	1×17	tanh	—	1.12
[56]	1×8	tanh	—	0.86

literatura identifica dos enfoques principales para la generación de datos en MPC: en lazo abierto, donde la ley de control se evalúa para un conjunto explícito de estados posibles [12, 49, 50] buscando capturar una mayor variedad de regiones de la ley de control; y en lazo cerrado, donde se obtiene una secuencia de actuaciones y estados a partir de un estado inicial [51, 53, 55], enfocándose en capturar el comportamiento en estado estacionario. En [50], los autores reportan una reducción del error de aproximación de la ley de control tras complementar los datos obtenidos en lazo abierto con datos en lazo cerrado, demostrando que ambos enfoques pueden combinarse eficazmente para caracterizar la ley de control.

Como conclusión de esta revisión del estado del arte, se observan ciertas tendencias en el uso de redes neuronales para aprender leyes de control de MPC. En primer lugar, para la generación de datos de entrenamiento, no importa si estos provienen de una formulación explícita o implícita, ya que ambos enfoques abordan el mismo problema. Esto le da al diseñador de la red neuronal la libertad de elegir la formulación que considere más adecuada para obtener datos representativos de MPC para el entrenamiento. En segundo lugar, dado que el controlador operará la mayor parte del tiempo en el punto de operación o en su vecindad, incluir una mayor cantidad de datos en esta zona dentro del conjunto de datos favorecerá un mejor desempeño del controlador en estado estacionario. En tercer lugar, aunque la elección de los hiperparámetros de las redes para aproximar MPC dependen de la ley de control específica del problema, las DNNs con activaciones ReLU son las más utilizadas. Finalmente, el MSE es la métrica de desempeño más empleada para el entrenamiento de las redes empleadas que aproximan la ley de control.

2.4.3. Implementaciones en FPGA de ANNs para aproximar MPC

En esta sección se revisan trabajos recientes que consideran el uso de ANNs para aproximar la ley de control y la FPGA para implementar el controlador. Esta revisión apunta a identificar directrices de diseño que se hayan usado en este contexto para incorporarlas en el flujo de trabajo a diseñar y evaluar, además de obtener datos reportados para comparar el desempeño de los controladores basados en ANNs implementados en este trabajo.

En la literatura, existen implementaciones de redes neuronales en RTL de tipo *shallow* de una sola capa oculta, ya que estas redes presentan una operatoria más sencilla en comparación con las redes *deep*. Dado que las redes *shallow* demandan menos recursos y permiten simplificar la lógica de implementación en RTL, se han vuelto una opción viable para algunas aplicaciones de MPC [53, 56]. Sin embargo, los métodos empleados para implementar estas redes no son escalables al agregar capas ocultas, lo que hace necesario adoptar otra estrategia de implementación para aplicaciones con leyes de control más complejas que requieren el uso de redes *deep* para capturar adecuadamente sus dinámicas.

Como estrategia alternativa a la descripción manual de RTL, el uso de herramientas de HLS ha permitido implementar redes con un mayor número de capas ocultas [19–22]. En [20], los autores reportan un *trade-off* relevante entre los hiperparámetros de la red neuronal, el error de aproximación y el uso de recursos, destacando que ciertas combinaciones de hiperparámetros no son factibles de implementar debido a limitaciones en los recursos disponibles. Como una técnica para reducir los recursos requeridos por la implementación, en [19] se propone el uso de representaciones cuantizadas de 16 bits, logrando una disminución del 80 % en el uso de recursos en comparación con

representaciones en punto flotante. Sin embargo, aunque se muestra que la cuantización permite reducir significativamente el uso de recursos, no se exploraron en detalle otros niveles de cuantización ni su efecto sobre el error de aproximación de la ley de control, lo cual restringe las opciones de personalización que entregan las FPGAs para optimizar el uso de recursos en base a un objetivo de desempeño.

La Tabla 2.1 muestra un resumen de la revisión de implementaciones de MPC con ANNs en FPGA, destacando aspectos como los hiperparámetros, la función de activación, la precisión numérica y la latencia. En la tabla, se puede observar que todas las redes revisadas presentan activaciones ReLU, tanh o Leaky ReLU, siendo ReLU la más utilizada en estas implementaciones. Además, en la tabla se puede notar que la latencia de las implementaciones en FPGA disminuye a medida que la red neuronal se vuelve más compacta, siendo la configuración más pequeña ($L = 1$, $M = 8$) la que alcanza la menor latencia (0.86 μs), lo cual sugiere una relación directa entre el tamaño de la red neuronal y su latencia. Adicionalmente, se destaca que hay pocos reportes sobre el uso de punto fijo en aplicaciones de MPC con redes neuronales, observándose en algunos casos el uso de 16 y 24 bits de punto fijo.

Adicionalmente, en la Tabla 2.1 se observa que los tamaños de las redes neuronales varían considerablemente, lo que refleja la necesidad de ajustar la arquitectura a los requisitos específicos de cada problema. Por ejemplo, en [56] se emplea una red *shallow* con 8 neuronas, mientras que en [22] se emplea una DNN de 256 neuronas y 7 capas ocultas. Esta variabilidad en los tamaños de red hace necesario contar con un *framework* que facilite la exploración de hiperparámetros de red y su factibilidad de implementación.

En relación con la viabilidad de la implementación, al comparar la complejidad de las redes neuronales para MPC implementadas por [22] y [20] se revela un panorama discrepante en cuanto a la utilización de recursos. Mientras que [22] logra implementar una red considerablemente grande, de 256 neuronas y 7 capas, sin reportar restricciones en el uso de recursos, [20] enfrenta limitaciones que impiden la implementación de redes con más de 5 neuronas y 2 capas. Esta discrepancia sugiere una falta de exploración exhaustiva del espacio de diseño en ambos trabajos, lo que impide establecer comparaciones concluyentes en términos de factibilidad y desempeño. Estos resultados subrayan la necesidad de un flujo de diseño más estandarizado que permita una evaluación más uniforme y reproducible del mapeo de redes neuronales en hardware.

En términos de los *frameworks* empleados para el diseño e implementación de redes neuronales para MPC en FPGA, varios trabajos coinciden en el uso de Matlab, tanto para generar datos de entrenamiento como para validar el hardware implementado, principalmente mediante la técnica *Hardware-in-the-Loop* (HIL) [20–22]. Aunque Matlab también permite generar descripciones en RTL a partir de algoritmos de alto nivel, en [21] se propone un flujo que integra Vitis HLS, Vivado y posteriormente Vitis para desarrollar la implementación en una FPGA, que forma parte de un SoC con CPU. Alternativamente, en [19] se propone un flujo basado en Keras [57] para el entrenamiento de la red, permitiendo ajuste fino mediante cuantización con Ristretto [58]. Sin embargo, al momento de escribir este documento, el repositorio oficial de Ristretto no presenta actividad desde agosto de 2017, lo cual desincentiva su uso ante la constante actualización de todas las herramientas de diseño de redes neuronales y FPGA.

Finalmente, de la última etapa de la revisión se extrae que, aunque Matlab es la herramienta más ampliamente usada para desarrollar todo el diseño del algoritmo MPC, existen discrepancias en las herramientas utilizadas para la implementación en FPGA. Además, se identifica la ausencia de un estudio exhaustivo sobre los efectos que tienen la elección de hiperparámetros y la cuantización de la red en la implementación en hardware. Asimismo, los trabajos revisados carecen de detalles completos sobre sus implementaciones, así como de repositorios con códigos, lo que produce que los resultados presentados en general tengan problemas de reproducibilidad.

Capítulo 3

Flujo de diseño de controladores MPC aproximados por ANNs

En este capítulo se presenta un flujo sistemático para diseñar e implementar controladores MPC aproximados mediante ANNs en hardware programable. El presente capítulo detalla cada fase del flujo, desde la generación del conjunto de datos y el diseño de la red neuronal, hasta la implementación de un sistema de hardware completo y su posterior validación en lazo cerrado. Además, serán tratados aspectos relevantes para lograr un balance adecuado entre latencia, error de aproximación del controlador y uso de recursos en FPGA. Los códigos utilizados y detalles técnicos sobre cada uno de los pasos descritos en este capítulo están disponibles en el repositorio que complementa este informe.

3.1 Organización del flujo de diseño propuesto

La revisión de la literatura presentada en el Capítulo 2.4 muestra un consenso general sobre los pasos necesarios para implementar ANNs aplicadas a MPC en FPGA. Estos pasos, comúnmente adoptados, son los siguientes:

1. Generación de un conjunto de datos basado en un lazo de referencia de MPC.
2. Selección y ajuste del modelo de red neuronal utilizando los datos generados.
3. Generación de una descripción de hardware a partir de la red neuronal entrenada.
4. Implementación e integración del acelerador de hardware con una CPU.
5. Validación del hardware mediante una planta simulada.

Para ejecutar estos pasos, es posible emplear diversas estrategias o flujos de diseño. En la literatura revisada, se identifican principalmente dos enfoques: un flujo general, que combina *frameworks* especializados para cada paso, y un flujo basado íntegramente en Matlab, donde esta herramienta se utiliza en todos los pasos del proceso. La Tabla 3.1 resume ambos enfoques, detallando los *frameworks* empleados en el flujo general y los *toolboxes* utilizados en el flujo Matlab.

Dado que Matlab es el *framework* más utilizado para el diseño y validación de controladores MPC, también se emplea en la generación de conjuntos de datos representativos de estos controladores, como se observa en ambas columnas de la Tabla 3.1. Sin embargo, los flujos comienzan a diferenciarse al momento de diseñar y entrenar una red neuronal para aproximar la ley de control. Mientras que el flujo basado en Matlab aprovecha el Deep Learning Toolbox, el flujo general utiliza *frameworks* especializados, como Keras, Tensorflow, PyTorch o Caffe, que ofrecen herramientas y optimizaciones específicas para esta tarea.

Posteriormente, es necesario convertir el modelo de la red neuronal a una descripción de hardware. En un flujo basado en HLS, esta descripción se obtiene en dos pasos: (i) la traducción de la red

Tabla 3.1: Frameworks reportados en la literatura para cada tarea en la implementación de una aproximación de MPC con redes neuronales en FPGA.

Paso	**Flujo general**	**Flujo Matlab**
Generación de conjunto de datos	Matlab	Matlab
Diseño y entrenamiento de red neuronal	Keras, PyTorch, Tensorflow, Caffe	Matlab (Toolbox DL)
Traducción de red neuronal a código HLS	—	HDL Coder
Generación de código RTL	Vitis HLS	HDL Coder
Síntesis e implementación de hardware	Vivado	HDL Coder*
Validación de hardware	Matlab, Simulink	Matlab, Simulink

*HDL Coder actúa como interfaz en la etapa de síntesis e implementación de hardware.

neuronal a código compatible con HLS (o simplemente, código HLS), y (ii) la generación asistida o automatizada de código RTL.

La Tabla 3.1 muestra que, en el flujo de Matlab, HDL Coder se utiliza para la totalidad del proceso de generación de código RTL. En contraste, en el flujo general no se reportan *frameworks* que asistan o automaticen la traducción de los modelos de las redes neuronales a código HLS, por lo que este paso suele realizarse manualmente, lo que lo convierte en una tarea extensa y propensa a errores.

A partir del código HLS, es común usar herramientas como Vitis HLS para generar la descripción en RTL. Posteriormente, el diseño RTL pasa por un proceso de síntesis lógica e implementación física, etapas que pueden realizarse mediante *frameworks* como Vivado o a través de HDL Coder, en el caso del flujo de Matlab.

Finalmente, en ambos flujos, el hardware implementado se valida a nivel de aplicación mediante la emulación de un lazo de control con una planta simulada, utilizando Matlab o Simulink.

Con base en los flujos de diseño e implementación identificados en la literatura, se propone un flujo general para la generación de controladores MPC aproximados mediante redes neuronales, implementados en hardware programable a partir de un controlador de referencia. La Figura 3.1 ilustra el proceso completo, que comienza con el diseño del controlador MPC para una aplicación específica y continúa con la generación del conjunto de datos necesario para el entrenamiento de las ANNs. Luego de esta etapa de generación de datos, el flujo se organiza en tres fases principales: el diseño de la red cuantizada, el diseño del hardware y la validación del hardware. Cada una de estas etapas requiere enfoques y herramientas específicas, y los detalles de cada una se discuten en las secciones siguientes.

3.2 Generación de conjunto de datos

Antes de la primera fase del flujo, es necesario generar un conjunto de datos representativo de la ley de control para entrenar las redes neuronales. Dado que las redes aprenden a partir de datos, la capacidad de la red neuronal para aproximar la ley de control puede verse comprometida si el conjunto de datos de entrenamiento no representa adecuadamente dicha ley, lo que subraya la importancia de una adecuada generación del conjunto de datos. Este paso previo se destaca con fondo verde en la Fig. 3.1. Para la generación del conjunto de datos, se asume la disponibilidad de un algoritmo MPC, en adelante denominado controlador de referencia, que ha sido diseñado y validado de acuerdo con los requerimientos específicos de la aplicación.

La generación de un conjunto de datos representativo de la ley de control es una tarea que puede abordarse de diversas formas y constituye un área de investigación en sí misma [12]. En términos generales, la generación de datos se puede realizar de dos maneras: en lazo abierto, donde se seleccionan estados específicos y se obtiene solo una actuación correspondiente a cada estado; y en lazo cerrado, donde también se eligen varios estados iniciales, pero se captura la secuencia de actuaciones y los estados alcanzados en el lazo hasta lograr el equilibrio en estado estacionario [50].

A grandes rasgos, una generación de datos en lazo abierto permite controlar completamente la distribución de las muestras en el conjunto de datos, facilitando la cobertura de distintas regiones de la ley de control. En cambio, la generación en lazo cerrado concentra los datos en torno a un punto

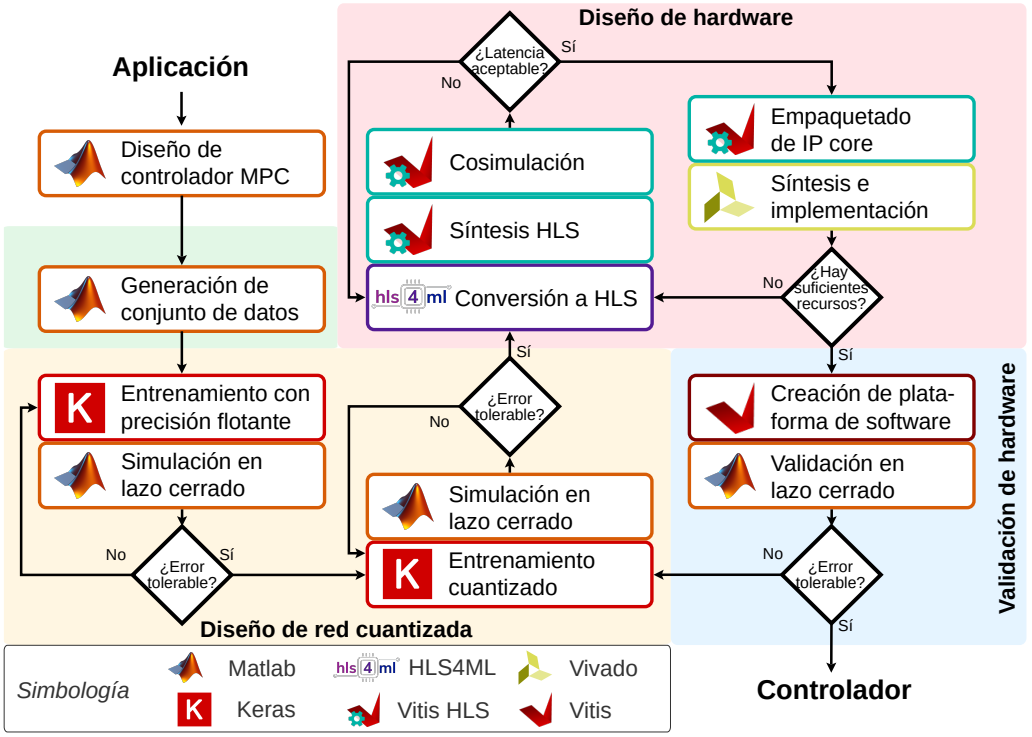


Figura 3.1: Flujo de diseño propuesto. Además de la generación de datos (en verde), el flujo se organiza en tres fases: (i) el diseño de la red cuantizada (en amarillo), (ii) el diseño del hardware (en rojo), y (iii) la validación del hardware (en azul).

de operación, lo que favorece la captura de las regiones correspondientes a los estados alcanzados durante la simulación. Ambos métodos no son excluyentes, y se han reportado resultados favorables en la aproximación de MPC al utilizarlos conjuntamente [50].

Para el flujo de diseño propuesto, se emplea una generación de datos en lazo abierto por simplicidad. Aunque este método puede conducir a un sobremuestreo de estados que rara vez se alcanzan durante la operación del controlador, presenta la ventaja de solo requerir información sobre las restricciones de estados del sistema, lo que facilita tanto su implementación como su extensión a diferentes aplicaciones.

Para determinar los estados que serán muestreados y añadidos al conjunto de datos, se utiliza una distribución uniforme dentro del conjunto de estados factibles para el sistema. Este enfoque permite que la concentración de datos en cada región sea proporcional al tamaño relativo de dicha región en la ley de control. Con esta distribución, cada estado muestreado se evalúa utilizando el controlador de referencia, obteniendo un par $(\mathbf{x}, \mathbf{u})$, donde $\mathbf{x}$ es el estado y $\mathbf{u}$ la acción de control correspondiente para la siguiente muestra. Este proceso de muestreo de estados y obtención de acciones de control se repite hasta alcanzar el tamaño deseado del conjunto de datos.

Dado que Matlab es la herramienta más utilizada para el diseño y simulación de controladores MPC, se opta por su uso para facilitar la generación del conjunto de datos. Este conjunto de datos se guarda posteriormente en un archivo `.csv`, que es un formato compatible con la mayoría de los *frameworks* de entrenamiento de redes neuronales.

3.3 Diseño de red cuantizada.

La primera fase del flujo tiene como objetivo entrenar una red cuantizada que pueda aproximar adecuadamente la ley de control de MPC. Para ello, dos decisiones de diseño afectan directamente el desempeño final de la red neuronal: la elección de los hiperparámetros y la cantidad de bits empleados para su cuantización. En la Fig. 3.1, esta fase se destaca con fondo amarillo.

3.3.1. Entrenamiento de la red neuronal

El objetivo de esta etapa es encontrar una estructura de red neuronal que permita aproximar la ley de control manteniendo el error de salida dentro de un rango predeterminado. Como se mencionó previamente, se consideran redes del tipo *fully-connected* con activaciones ReLU. Las redes neuronales consideradas en esta fase tienen precisión de punto flotante, lo que permite que su capacidad de aprendizaje dependa únicamente de sus hiperparámetros, sin estar limitada por la precisión numérica. El rango aceptable para el error de aproximación de la red depende de los requerimientos específicos de cada aplicación y se establece en función del comportamiento observado al reemplazar el algoritmo MPC de referencia por la red neuronal entrenada en la simulación del lazo de control.

3.3.1.1. Configuración de entrenamiento de punto flotante

El diseño y entrenamiento de la red neuronal se realiza utilizando Keras 2 [57], un *framework* popular de alto nivel para el desarrollo de modelos de redes neuronales profundas. Keras es una herramienta de código abierto enfocada en la productividad, que ofrece una interfaz simple y amigable para facilitar la exploración y validación funcional de modelos de redes complejas. Además, una característica importante es que los modelos descritos en Keras 2 son compatibles con las herramientas utilizadas en la fase posterior del diseño de la red cuantizada.

Antes de entrenar la red, es necesario preprocesar el conjunto de datos mediante la normalización de las entradas y salidas. La normalización es una práctica recomendada que permite homogeneizar los datos y facilita el proceso de aprendizaje de la red neuronal [32]. Además, normalizar las entradas y salidas ayuda a equilibrar los órdenes de magnitud en las operaciones en las capas de la red, lo cual facilitará su posterior cuantización.

Una vez normalizado, el conjunto de datos se divide en un conjunto de entrenamiento y un conjunto de *test*, que se utiliza para evaluar el desempeño de la red. Dentro del conjunto de entrenamiento, en cada *epoch* se reserva una parte de los datos para realizar una evaluación del error de aproximación de la red neuronal al final del *epoch*, lo cual permite detectar oportunamente la presencia de *overfitting*.

Para el entrenamiento, dado que se trata de un problema de regresión, se utiliza la función de pérdida MSE junto con una penalización de norma L^2 en los pesos de la red neuronal. Esta penalización ayuda a evitar el *overfitting* [14,32], especialmente cuando la red está sobredimensionada en relación con la cantidad de regiones que debe aprender de la ley de control. En este contexto, la penalización evita que la red aprenda a interpolar regiones adicionales que no están en el conjunto de datos ni forman parte de la ley de control. Además, mantener los pesos de la red en valores bajos permite que su desempeño sea comparable con el de la versión cuantizada que se entrenará posteriormente.

El entrenamiento de la red neuronal finaliza en cualquiera de los siguientes casos:

- Se alcanza el número máximo de *epochs* de entrenamiento.
- El MSE calculado en al final de cada *epoch* se mantiene sin disminución durante un número predefinido de *epochs* consecutivos, denominado paciencia.

En cualquiera de estos casos, se conservan los parámetros que permitieron a la red obtener el menor error de aproximación en el conjunto de datos reservado durante el entrenamiento.

3.3.1.2. Exploración de hiperparámetros y evaluación de la red

En esta etapa de diseño, el desafío consiste en encontrar los hiperparámetros de la red neuronal, definidos en términos de la cantidad de capas ocultas L y unidades por capa M, que permitan el mejor balance el entre error de aproximación y el tamaño de la arquitectura de la red. La arquitectura de la red neuronal tiene tres efectos principales sobre la implementación:

- **La cantidad de unidades por capa y de capas ocultas afectan la capacidad de aprendizaje de la red.** La Figura 2.3 muestra que el aprendizaje de la red neuronal mejora en mayor medida al incrementar la cantidad de capas ocultas, en comparación con el aumento

de unidades por capa. Según [15], este resultado se cumple siempre que $M \geq n_x$, es decir, que haya al menos tantas unidades por capa como entradas en la red.

- **El número de capas ocultas afecta directamente la latencia de inferencia de la red.** Dado que cada capa en la red neuronal depende de las salidas calculadas por la capa anterior (ver Fig. 2.2), estas deben procesarse secuencialmente. Por lo tanto, añadir capas ocultas incrementa la latencia de inferencia de la red.
- **La cantidad de operaciones de multiplicación en la estructura de la red neuronal está directamente relacionada con el uso de recursos.** La cantidad de multiplicaciones en una red neuronal de L capas con M unidades se determina mediante la expresión:

$$N_{\text{mult}} = (L-1)M^2 + M(n_x + n_u) \tag{3.1}$$

donde n_x y n_u representan el número de entradas y salidas, respectivamente. La ecuación (3.1) muestra que el número de multiplicaciones depende cuadráticamente de la cantidad de unidades por capa y linealmente de la cantidad de capas ocultas. Aunque el mapeo de estas operaciones en el hardware se discutirá más adelante, es importante considerar en esta etapa que una gran cantidad de multiplicaciones podría limitar las alternativas de implementación debido a la disponibilidad de recursos.

La búsqueda de la arquitectura de red se realiza exhaustivamente dentro de un rango de valores heurísticamente definidos para L y M, cumpliendo con $M \geq n_x$ y priorizando siempre los valores más pequeños para obtener una red compacta. Si se dispone del conteo de las regiones de la ley de control, el rango de búsqueda de los hiperparámetros puede ajustarse adicionalmente utilizando la ecuación (2.33), que permite hacer una relación entre la cantidad de regiones y los hiperparámetros que podría tener una red que aproxima esa ley de control.

Para finalizar la exploración de hiperparámetros, se evalúa el desempeño de la red neuronal mediante una simulación en lazo cerrado. Esta simulación permite observar el comportamiento de la planta conectada a la red neuronal y verificar si esta puede controlar los estados del sistema y cumplir con las restricciones. Dado que el controlador MPC de referencia satisface estos requisitos, el desempeño de control a nivel de aplicación se evalúa en función de la diferencia de la red con respecto al controlador de referencia. Como el controlador de referencia está implementado en Matlab, se opta por realizar las simulaciones en el mismo entorno, cargando la red a través del intérprete de Python para Matlab [59]. Los resultados de esta simulación permitirán decidir si es necesario continuar con la búsqueda de hiperparámetros o si el error de aproximación de la red es adecuado para avanzar con el flujo.

Los hiperparámetros encontrados en este paso se utilizarán en la siguiente etapa. Opcionalmente, también se pueden emplear los pesos entrenados para acelerar el entrenamiento de la red cuantizada.

3.3.2. Entrenamiento cuantizado

El objetivo de esta etapa es aproximar nuevamente la ley de control, pero ahora utilizando una red neuronal cuantizada con activaciones ReLU y los hiperparámetros determinados en la etapa anterior. En esta etapa, la red neuronal se comprime utilizando representaciones numéricas en punto fijo, manteniendo el error de aproximación dentro de un rango tolerable para la aplicación.

3.3.2.1. Configuración de entrenamiento cuantizado

Para que el desempeño de la red neuronal cuantizada sea comparable al de la red en punto flotante, se utilizan las mismas configuraciones de entrenamiento, incluida la penalización de norma L^2 en los pesos. Además, dado que las entradas y salidas de la red están normalizadas, se prevé un uso reducido de los bits asignados a la parte entera en la cuantización, lo que permite enfocar el ajuste principalmente en la cantidad de bits para la parte fraccionaria. Para facilitar la exploración de niveles de cuantización, se asume un nivel uniforme en todas las capas de la red neuronal, es decir, todos los parámetros y activaciones tienen el mismo nivel de cuantización.

El entrenamiento cuantizado se realiza con el *framework* QKeras, que es una extensión de Keras que permite diseñar y entrenar redes neuronales con cuantización en pesos, *bias* y funciones

de activación. Durante el entrenamiento, el nivel de cuantización se define en términos de W, la cantidad total de bits, y Q, la cantidad de bits asignados al signo y a la parte entera. Es importante considerar representaciones con signo para la red neuronal, ya que los valores negativos permiten que la función ReLU inhabilite la actividad de la neurona cuando sea necesario; por lo tanto, es deseable que $Q \geq 1$.

3.3.2.2. Exploración de niveles de cuantización y evaluación de la red

En este paso, es necesario explorar los niveles de cuantización para la red neuronal, utilizando los hiperparámetros previamente seleccionados. Esta exploración se realiza entrenando varias instancias de la red con los mismos hiperparámetros definidos en la etapa anterior (ver Sección 3.3.1), aplicando diferentes niveles de cuantización y evaluando el error de aproximación de la ley de control mediante una simulación en lazo cerrado. Se recomienda utilizar Matlab para facilitar la evaluación del desempeño a nivel de aplicación. En general, al elegir el nivel de cuantización, se debe considerar que, aunque un mayor número de bits reduce el error de aproximación de la red, también implica un mayor uso de recursos en el hardware. Por lo tanto, se recomienda comenzar con un número bajo de bits para la representación en punto fijo e ir incrementándolo gradualmente hasta que la simulación en lazo cerrado muestre un comportamiento aceptable en comparación con el lazo de referencia, lo cual debe determinarse para cada aplicación.

De este paso, se obtiene un modelo de red cuantizada con hiperparámetros definidos y pesos entrenados, que cumple con los requerimientos del lazo de control de la aplicación. El modelo entrenado se exporta como un archivo `.keras`, que servirá como entrada para el primer paso de la siguiente fase. Con esto, se interrumpe el uso de Matlab y Keras en el flujo de diseño, ya que la siguiente etapa se centra en la implementación de la red entrenada en una FPGA, lo cual es un problema fundamentalmente distinto y requiere de nuevas herramientas especializadas.

3.4 Diseño de hardware

La segunda fase del flujo tiene como objetivo implementar la red neuronal previamente entrenada e integrarla en una plataforma de hardware adecuada para su uso en una aplicación de MPC. Además de las decisiones tomadas en la fase anterior, la principal decisión de diseño en esta fase es la elección del factor de reutilización (ver Sección 2.3.2). En la Fig. 3.1, esta fase de diseño se destaca con fondo rojo.

A diferencia del diseño de redes neuronales, el diseño de hardware para la aceleración en FPGA es un proceso diferente que requiere herramientas especializadas en desarrollo de hardware. Por esta razón, para implementar el hardware se utilizarán conjuntamente herramientas dedicadas como HLS4ML, Vitis HLS y Vivado.

3.4.1. Creación de IP-Package

En este paso, el modelo de alto nivel de la red neuronal cuantizada con QKeras se convierte en una descripción de hardware funcionalmente equivalente. El objetivo es balancear el uso de recursos y la latencia mediante la elección del factor de reutilización en HLS4ML, teniendo en cuenta las restricciones de tiempo de la aplicación y los recursos disponibles en la plataforma.

3.4.1.1. Condiciones de síntesis de alto nivel

Aunque HLS4ML permite configurar los niveles de cuantización para cada capa, en este paso no se aplicará esta optimización, ya que la red fue previamente cuantizada en QKeras. Por lo tanto, la configuración de cuantización en HLS4ML se mantiene igual, utilizando los mismos parámetros W y Q definidos durante el entrenamiento cuantizado.

HLS4ML permite trabajar con varios *backends* para realizar simulación, síntesis e implementación. En este flujo, se ha elegido el *backend* Vitis HLS. Aunque durante el desarrollo de este trabajo el soporte para Vitis HLS se encontraba en fase experimental, resultó completamente funcional

y compatible para las evaluaciones realizadas, sin presentar impedimentos en la funcionalidad explorada.

Al emplear el *backend* Vitis HLS, HLS4ML genera por defecto una interfaz AP, que es el protocolo de comunicación predeterminado para los IPs generados en Vitis HLS. Dado que HLS4ML opera sobre un proyecto de Vitis HLS, cambiar la interfaz AP a una interfaz AXI-lite es un proceso sencillo, automatizado mediante un script disponible en el repositorio asociado a esta tesis.

Una vez definida la configuración para el modelo de la red neuronal, se procede a la simulación, síntesis HLS y cosimulación del diseño en Vitis HLS mediante HLS4ML. La simulación y cosimulación se realizan para verificar que la red se comporte de acuerdo con una *golden reference*, construida a partir del mismo conjunto de datos utilizado en el entrenamiento de la red.

Si la descripción de hardware es funcionalmente equivalente a la red neuronal, el error obtenido en la cosimulación será comparable al observado en la simulación previa a la síntesis.

3.4.1.2. Exploración de reutilización y reportes de síntesis

Durante este paso, se exploran las configuraciones posibles para el mapeo de la red neuronal a hardware, de modo que se logre una latencia adecuada para la aplicación con los recursos disponibles para la implementación. Para alcanzar este objetivo, se realiza la síntesis de alto nivel de la red neuronal, evaluando distintos factores de reutilización, desde uno (*full unroll*) hasta M^2.

Al explorar los factores de reutilización, es importante tener en cuenta que un mayor factor de reutilización reduce el uso de recursos a costa de un aumento en la latencia. Además, la reutilización máxima está limitada por la cantidad de multiplicaciones requeridas en las capas ocultas de la red neuronal. En una red con M unidades por capa oculta, se requieren M^2 multiplicaciones por capa, por lo que el mayor factor de reutilización posible será M^2. Por otro lado, la cantidad mínima de multiplicadores para la red está acotada inferiormente por el número total de capas, es decir, toda red implementada en hardware tendrá al menos un multiplicador por cada capa.

Tras la síntesis, la latencia y el uso de recursos estimados se obtienen a través de los reportes de Vitis HLS. Aunque la latencia (en ciclos de reloj) y la cantidad de bloques DSP utilizados son precisos, la herramienta tiende a ser conservadora al reportar el *timing* y el uso de FFs y LUTs [24]. Por lo tanto, no cumplir con estos últimos criterios en esta etapa no necesariamente impedirá la implementación final.

Los diseños sintetizados que cumplen al menos con el requisito de latencia para la aplicación se exportan en formato de IP-Package, lo cual permite su posterior incorporación en una plataforma de hardware. El IP-Package es una descripción de hardware funcionalmente equivalente al modelo de la red neuronal cuantizada y está listo para su integración e implementación en el sistema.

3.4.2. Plataforma de hardware

En este paso, se construye la plataforma de hardware que interactuará directamente con el IP-Package creado en HLS4ML, generando un archivo que permitirá programar la FPGA.

Dado que se utilizarán plataformas ZYNQ, que incorporan una CPU conectada a la FPGA, se considera un sistema compuesto por el procesador integrado del SoC, conectado al IP que contiene la red neuronal. Este sistema se construye utilizando los *Block Diagrams* de Vivado, que permiten abstraer la elaboración del sistema como diagramas de bloques interconectados. Aunque el uso del IP no está limitado a sistemas con CPU, en este flujo se opta por integrar una CPU, ya que facilita la validación en la siguiente fase.

Para realizar la comunicación entre el procesador y el IP, se utiliza el bus AXI con la configuración por defecto proporcionada por Vivado. Es importante asegurarse de que la sección de memoria asignada al IP dentro del bus AXI sea lo suficientemente grande para contener todos sus registros.

Después de conectar el bus de comunicación, se deben realizar la síntesis lógica y la implementación del diseño. Los reportes generados durante la implementación permitirán ajustar las estimaciones de *timing* y uso de recursos realizadas previamente por Vitis HLS. Si el uso de recursos excede los recursos disponibles según los reportes, será necesario volver a la etapa anterior para seleccionar un factor de reutilización mayor, como se indica en la Fig. 3.1. Al finalizar esta fase, la FPGA estará programada con el hardware diseñado, quedando pendiente solo la validación.

3.5 Validación de hardware

En general, es esperable que existan diferencias entre la red cuantizada en software y la red implementada en hardware. Aunque la red en software esté cuantizada, sigue ejecutándose en un entorno optimizado para punto flotante, lo que puede generar discrepancias numéricas respecto a una implementación en hardware dedicado. Estas diferencias pueden provocar un error acumulativo durante la simulación, afectando el comportamiento de ambos controladores. Por esta razón, es necesaria una validación final del hardware implementado mediante simulaciones, con el fin de verificar que cumple con los requerimientos de control de la aplicación.

3.5.1. Plataforma de software

Para utilizar el hardware diseñado, se requiere software que facilite la interacción entre el módulo de la red neuronal y la CPU del sistema, además de permitir la comunicación con un *host*, lo cual posibilita la validación del hardware mediante una planta simulada.

El desarrollo del software puede realizarse de diferentes maneras, como por ejemplo, mediante una aplicación *Bare Metal* o utilizando bibliotecas PYNQ en Jupyter Notebook.

3.5.1.1. Aplicación *Bare Metal*

Cuando el software es *bare metal*, la aplicación, escrita en C/C++, se ejecuta directamente sobre el procesador sin un sistema operativo ni otra capa de abstracción. Esto permite un control directo de los periféricos y una interacción más cercana con el hardware. Además, se puede configurar fácilmente el uso de interrupciones desde el IP, para indicar al procesador cuando el procesamiento de datos ha concluido.

Dada la memoria limitada de la aplicación, la tarjeta de desarrollo se conecta a través de USB, que se utiliza como interfaz serial para comunicarse con un computador personal o *host*. De este modo, se puede emplear la misma estrategia de evaluación mediante una *golden reference* utilizada en la simulación y cosimulación.

Los datos recibidos a través de la interfaz serial del procesador se envian por AXI a la red neuronal. Una vez iniciado el procesamiento de los datos de entrada, el procesador espera la finalización del cálculo utilizando la señal de interrupción del IP.

Una vez que se indica la finalización del procesamiento de los datos en la red neuronal, el procesador lee la salida y la envía a través de la interfaz serial al *host*, que la comparará con la *golden reference* antes de enviar el siguiente dato.

3.5.1.2. Aplicación en Python (PYNQ)

Otra alternativa para desarrollar la capa de software es utilizar el *framework* PYNQ, que facilita el desarrollo e integración de aplicaciones de software con hardware diseñado en FPGA. En términos prácticos, es una extensión del sistema operativo que incluye librerías para resolver automáticamente los mapas de memoria física a virtual. Estas librerías permiten que el hardware sea accesible y utilizado desde aplicaciones de más alto nivel, como Python o Jupyter Notebook.

El soporte del sistema operativo en PYNQ permite desarrollar aplicaciones más complejas con mayor facilidad. Sin embargo, tiene la desventaja de que las interrupciones provenientes del bloque IP no son tan accesibles como en una aplicación *bare metal*, lo que obliga a utilizar estrategias menos eficientes, como el *polling*, para detectar la disponibilidad de los datos de salida del módulo.

En resumen, existen distintas alternativas para el desarrollo de plataformas de software que permiten el uso del hardware implementado, entre las cuales se encuentran PYNQ y *bare metal*. PYNQ facilita la integración de hardware y software al proporcionar un manejo de alto nivel del IP, simplificando el desarrolla a costa de un menor control sobre la ejecución, el manejo de datos y la comunicación. En contraste, *bare metal* permite un acceso a bajo nivel al IP, otorgando mayor control en comparación con PYNQ, pero con una mayor complejidad en el desarrollo y la depuración del software. Dado que estas diferencias pueden influir en el comportamiento del sistema, es necesario realizar evaluaciones para cuantificar los *overheads* asociados a cada plataforma de software y determinar su impacto en aplicaciones de MPC.

3.5.2. Validación en lazo cerrado

Finalmente, aprovechando la conectividad proporcionada por la interfaz serial USB del software, se utiliza un *host* para realizar una simulación de una planta en Matlab.

Para evaluar el desempeño del hardware a nivel de aplicación de control, se emula el comportamiento de la planta a controlar y se envían los estados medidos a través de la interfaz serial. Luego, la acción de control generada por el hardware implementado se recibe y se aplica a la planta simulada. Este proceso se repite de manera iterativa durante un número definido de pasos de simulación.

Los resultados de la simulación se analizan en términos del comportamiento de los estados y las actuaciones a lo largo de las muestras simuladas. Los datos obtenidos se comparan con los del controlador de referencia para evaluar el desempeño de la red neuronal implementada en hardware como controlador. Si la diferencia en el comportamiento entre la red y el controlador de referencia son admisibles, entonces se puede dar por terminado el flujo de diseño e implementación. En caso contrario, es necesario considerar el uso de una red con diferente nivel de cuantización, como indica la Figura 3.1.

Capítulo 4

Exploración de optimizaciones de hardware usando HLS4ML

Este capítulo presenta diversas exploraciones dentro del espacio de diseño disponible para mapear una red neuronal al hardware de una FPGA, empleando directivas proporcionadas por el framework HLS4ML para síntesis de alto nivel. El objetivo es establecer lineamientos para seleccionar hiperparámetros y niveles de cuantización en redes neuronales aplicadas a MPC, evaluando cómo estas decisiones afectan el uso de recursos y la latencia de inferencia en hardware en base a la dimensión de la red a implementar. Estos lineamientos complementan el flujo de diseño propuesto en el Capítulo 3, facilitando al diseñador la búsqueda de implementaciones que satisfagan los requerimientos específicos del problema. Las evaluaciones presentadas en este capítulo consideran solo la estructura de la red en base a los hiperparámetros como el número de neuronas por capa y el número de capas, sin considerar una fase de entrenamiento para ajustar sus parámetros. Los lineamientos derivados de esta exploración se aplicarán posteriormente en el Capítulo 5 para validar su funcionalidad en casos de estudio concretos.

4.1 Metodología de exploración

El objetivo de este capítulo es determinar las dimensiones de las redes neuronales que son implementables en FPGA y obtener una estimación preliminar del desempeño esperado para distintas configuraciones de hardware. Dado que el análisis se centra en evaluar métricas de recursos y latencia, independientemente de la precisión del modelo, no se considera el error de aproximación de las redes evaluadas. Esto permite asignar pesos de manera aleatoria, simplificando el análisis al eliminar la necesidad de entrenamiento previo. A pesar de esta simplificación, se verifica la equivalencia funcional de cada red implementada mediante inferencias realizadas con un conjunto de datos aleatorio, asegurando que su comportamiento sea coherente durante el proceso de diseño.

En particular, utilizando HLS4ML, Vitis HLS y Vivado, se analizan los resultados de síntesis e implementación bajo las siguientes condiciones de diseño, cada una enfocándose en distintos aspectos que se esperan sobre el hardware resultante:

- **Redes neuronales con diferentes combinaciones de hiperparámetros, pero igual cantidad de neuronas totales.** Este análisis examina cómo la distribución de las neuronas entre capas afecta el uso de recursos y la latencia. Dado que el número de operaciones de cada neurona depende del tamaño de la capa anterior, se busca entender cómo la estructura interna de la red influye en su desempeño.

- **Redes neuronales con distintas configuraciones de hiperparámetros, pero igual cantidad de multiplicaciones.** Este estudio evalúa cómo los hiperparámetros impactan el uso de recursos y la latencia, manteniendo constante el total de multiplicaciones de la red. Esto permite analizar el efecto de los hiperparámetros manteniendo estable el costo computacional asociado a las multiplicaciones.

Tabla 4.1: Recursos disponibles en la tarjeta de desarrollo ZCU104.

LUTs	FFs	DSP	BRAM
230,400	460,800	1,728	624

- **Redes neuronales con distintas configuraciones de hiperparámetros y diferente cantidad de multiplicaciones.** Esta exploración apunta a evaluar cómo el número total de multiplicaciones impacta el uso de recursos y la latencia en una implementación en hardware. A diferencia de los análisis previos, este estudio no está limitado por la disposición o el número de neuronas, permitiendo un análisis más extensivo del efecto del costo computacional asociado a las multiplicaciones en las métricas de las propiedades del hardware resultante.

- **Redes neuronales con diferentes niveles de cuantización, pero mismos hiperparámetros.** El nivel de cuantización, definido por la cantidad de bits utilizados para representar valores numéricos en la red, impacta directamente los requerimientos de cómputo y almacenamiento en el hardware. Esta exploración analiza cómo dicho nivel afecta el uso de recursos y la latencia de inferencia.

- **Redes con diferentes factores de reutilización y niveles de cuantización, pero mismos hiperparámetros.** Esta exploración se centra en cómo el factor de reutilización, definido como la cantidad de veces que una instancia de multiplicación se reutiliza dentro de una capa, impacta el uso de recursos y la latencia de inferencia. Además, se evalúa cómo estos efectos varían con distintos niveles de cuantización.

- **Redes reportadas en la literatura con variaciones en el factor de reutilización y los niveles de cuantización.** Esta exploración evalúa la factibilidad de implementar estas redes con las herramientas utilizadas y analiza posibles mejoras en latencia y uso de recursos respecto de los resultados previamente reportados.

Para delimitar el espacio de diseño, se seleccionó la tarjeta de desarrollo ZCU104 de AMD/Xilinx [43] como plataforma de referencia para las implementaciones. Esta tarjeta ha demostrado ser adecuada para aplicaciones de MPC en trabajos previos [45, 47], lo que respalda su elección en este estudio. Aunque los resultados obtenidos son específicos para este hardware, los patrones observados proporcionan directrices aplicables a otras plataformas con características similares. De este modo, el análisis presentado facilita la búsqueda de implementaciones que cumplan con los requisitos de distintas aplicaciones. La Tabla 4.1 resume los recursos disponibles en la tarjeta ZCU104.

4.2 Comportamientos técnicos esperados en HLS4ML

La implementación en FPGA de modelos de redes neuronales mediante HLS4ML, Vitis HLS y Vivado presenta una serie de comportamientos y limitaciones documentados que derivan de las especificaciones técnicas de estas herramientas. Estas características influyen directamente en las decisiones de diseño y en la evaluación de las implementaciones. A continuación, se destacan los aspectos más relevantes que guían las exploraciones de este capítulo:

- **Arquitectura de streaming:** Al usar HLS4ML, los valores de salida de cada capa en la red se calculan de manera secuencial e independiente, utilizando un *pipeline* en el que la salida de cada capa se registra para ser utilizada por la siguiente en el siguiente ciclo de reloj. Debido a que cada capa se implementa como un motor de cómputo independiente, no es posible compartir recursos, como multiplicadores, entre capas [60].

- **Hardware de control:** Vitis HLS infiere automáticamente hardware de control para gestionar la coordinación de las operaciones en la red neuronal, incluyendo el manejo del flujo de datos y la sincronización entre las distintas capas [60]. Este hardware representa un *overhead* en términos de LUTs y FFs que se espera varíe en función del número de capas de la red.

- **Uso de DSPs para multiplicaciones:** La mayoría de FPGAs modernas cuentan con un número limitado de bloques DSP dedicados para realizar multiplicaciones. En su configuración por defecto, Vitis HLS utiliza DSPs para implementar multiplicadores de 10 o más bits [61]; para anchos menores, los multiplicadores se implementan exclusivamente con LUTs. En casos donde el ancho de la multiplicación excede las capacidades del DSP, la herramienta complementa con LUTs o utiliza DSPs adicionales, priorizando el desempeño [62]. Para la tarjeta de desarrollo utilizada, los DSPs son de 18×27 bits, por lo que, cuando una de las entradas exceda los 18 bits, se espera que la capacidad del DSP se vea superada, requiriendo LUTs o DSPs adicionales.

- **Uso de LUTs para activaciones:** Las redes neuronales consideradas en este estudio emplean funciones de activación ReLU (ver Sección 2.2), que suelen implementarse utilizando exclusivamente LUTs. Esto genera una relación directa entre el uso de LUTs y las funciones de activación en las implementaciones.

- **Preferencia por FFs frente a BRAM:** La estrategia *Latency* de HLS4ML utilizada con Vitis HLS, resulta en una implementación de los pesos de la red con partición completa en registros, lo que evita el uso de BRAM en las implementaciones de este estudio [60]. Esta estrategia es la configuración utilizada por defecto en HLS4ML.

- **Independencia del valor de la parte entera Q:** En términos de la implementación en hardware, las representaciones de operaciones en punto fijo son equivalentes a las operaciones con números enteros, y la ubicación del punto decimal afecta la interpretación del resultado, pero no la operatoria asociada. En este sentido, al evaluar el costo computacional de una operación de punto fijo solo importa conocer la cantidad total de bits (W) utilizados para representar los datos, sin que la distribución entre parte entera y fraccionaria tenga impacto en la implementación. Por lo tanto, el valor de Q es irrelevante desde la perspectiva del diseño de hardware.

Estos comportamientos documentados proporcionan información relevante para definir las condiciones de las exploraciones y establecer expectativas claras sobre los resultados de las implementaciones. En particular, dado que los pesos y bias de las redes se almacenan en FFs, y las funciones ReLU no requieren de tablas de búsqueda su implementación, las redes evaluadas en este capítulo no utilizan BRAM. Por lo tanto, los reportes de uso de BRAM se omitirán a lo largo de este capítulo.

4.3 Redes neuronales con diferentes hiperparámetros e igual número de neuronas

4.3.1. Objetivo del experimento

El objetivo de este estudio apunta a evaluar cómo varía el uso de recursos y la latencia en modelos de redes neuronales con una cantidad fija de neuronas, pero distribuidas en diferentes cantidades de capas ocultas. Este estudio se considera relevante, ya que desde una perspectiva de alto nivel, suele asumirse intuitivamente que el costo computacional de implementar una red neuronal depende únicamente del número total de neuronas. Sin embargo, desde el punto de vista de hardware, lo relevante es el número de operaciones y las dependencias entre estas, lo cual puede variar significativamente dependiendo del número de capas y las neuronas por capa.

Al utilizar representaciones numéricas con más de 10 bits y un factor de reutilización fijo que permita explotar el paralelismo en cada capa, se espera que al distribuir las neuronas en más capas disminuya el número de DSPs necesarios para su implementación, siguiendo la misma tendencia del número de operaciones. Por otro lado, la cantidad de funciones de activación en una red neuronal coincide con el número total de neuronas, y asumiendo que estas funciones suelen implementarse con LUTs, se espera que el uso de LUTs se mantenga similar independiente del número de capas.

Tabla 4.2: Parámetros utilizados para la exploración de redes con diferentes hiperparámetros y la misma cantidad de neuronas.

Parámetro	**Valor**
Período	10 [ns]
Estrategia	*Latency*
R	1
W	32
Q	12
n_u	1
n_x	{ 2, 4, 8 }
$\text{Neuronas}_{n_x=2}$	{ 13, 25, 47, 61 }
$\text{Neuronas}_{n_x=4}$	{ 25, 37, 61, 73 }
$\text{Neuronas}_{n_x=8}$	{ 49, 61, 73, 91 }

Del mismo modo, como se mencionó en la Sección 4.2, dado que la salida de cada neurona debe registrarse antes de ser utilizada por la siguiente capa, se espera que el uso de FFs se mantenga estable al redistribuir las neuronas. Por último, debido a la dependencia de datos, se prevé que la latencia tenga una correlación positiva con la cantidad de capas ocultas.

4.3.2. Condiciones de exploración

Las condiciones de la exploración para esta configuración se resumen en la Tabla 4.2. El reloj objetivo para la síntesis se fija en su valor por defecto que es 10 [ns], y la estrategia de optimización de HLS4ML se fija en *Latency*. El factor de reutilización R se establece en 1 para aprovechar el paralelismo en las operaciones de una misma capa. El nivel de cuantización W se establece en 32 bits, que es un valor comúnmente utilizado para las operaciones en CPUs y fuerza el uso de DSPs para las multiplicaciones. Además, se asignan arbitrariamente 12 bits para la parte entera Q como referencia, lo cual no tiene mayores efectos en la implementación en hardware. Finalmente, para mantener el enfoque en la redistribución del número de neuronas por capa M y la cantidad de capas ocultas L, el número de salidas n_u de la red neuronal se fija en 1.

Para complementar el análisis, como se muestra en la Tabla 4.2, se consideran tres valores para el número de entradas n_x: 2, 4 y 8. Aunque este parámetro no afecta directamente el total de neuronas de la red, en la práctica establece un mínimo para la cantidad necesaria de neuronas por capa para que la red pueda aprender de manera efectiva, según la expresión (2.33). Considerando esta restricción y el hecho de que el total de neuronas de una red se define como $L \cdot M + n_u$, se seleccionan combinaciones de M y L que permitan al menos cinco redistribuciones de neuronas.

Para redes con dos entradas, las configuraciones seleccionadas son de 13, 25, 37 y 61 neuronas. En el caso específico de 13 neuronas, las redistribuciones incluyen 1×12, 2×6, 3×4, 4×3 y 6×2, dejando una neurona fija en la capa de salida. Para redes con cuatro entradas, las configuraciones incluyen 25, 37, 61 y 73 neuronas, mientras que para redes con ocho entradas, las configuraciones seleccionadas son de 49, 61, 73 y 91 neuronas.

4.3.3. Análisis de uso de recursos

La Figura 4.1 muestra el uso de recursos reportado por Vitis HLS luego de la síntesis lógica de las redes consideradas para esta exploración. En particular, la Fig. 4.1a considera la redistribución de neuronas utilizando dos entradas, donde se pueden observar tendencias similares en el uso de recursos para las cuatro cantidades de neuronas consideradas. Además, se observa que respecto de la cantidad de recursos disponibles en la tarjeta, el recurso utilizado más intensivamente son los bloques DSP, seguido de las LUTs, para terminar con un uso de FFs que es considerablemente menor en términos relativos en todos los casos. Las tendencias observadas en el uso de recursos muestran correlación con la cantidad de multiplicaciones de las redes, siendo los casos de dos capas ocultas los que presentan el mayor uso de recursos, al igual que la cantidad de multiplicaciones. Más aún, si la cantidad de neuronas es una constante $L \cdot M$, el total de multiplicaciones aumenta a medida que disminuye la cantidad de capas ocultas, alcanzando su máximo con dos capas ($L = 2$).

Esto se cumple siempre que $L \cdot M > 2(n_x + n_u)$, es decir, que el total de neuronas en las capas ocultas exceda el doble de la suma de entradas y salidas de la red. Para más detalles, consulte el Anexo A.

La Fig. 4.1b muestra que las redes que tienen cuatro entradas siguen las mismas tendencias observadas en la Fig. 4.1a, con la excepción del caso de 73 neuronas, donde la red con 3 capas ocultas alcanzó el uso del 100 % de los bloques DSP disponibles y presenta un mayor uso de LUTs de lo esperado en relación a la tendencia observada en los demás casos.

Para el caso de las redes con ocho entradas, en la Fig. 4.1c se observa que estas también siguen, en general, las mismas tendencias en el uso de recursos. Sin embargo, hay más casos donde se alcanzó el uso del 100 % de los DSPs, lo que llevó a la herramienta de síntesis al uso de LUTs adicionales para suplir estos requerimientos. Como resultado, las implementaciones de las redes de 73 neuronas con 3 capas ocultas y de 91 neuronas con 3 capas ocultas se volvieron inviables debido a que el requerimiento de LUTs superó al total disponible.

Finalmente, la Figura 4.2 ilustra el impacto de la cantidad de entradas en el uso de recursos para las redes de 61 neuronas. En particular, se observa un aumento en el uso de recursos al incrementar el número de entradas. Este efecto es más pronunciado en redes con menos capas ocultas, especialmente en aquellas con una sola capa, donde las entradas contribuyen directamente al número de multiplicaciones. En redes más profundas, el impacto de las entradas se reduce, ya que las operaciones asociadas a estas representan una proporción menor en comparación con las realizadas en las demás capas.

4.3.4. Análisis de latencia

A diferencia del uso de recursos, la latencia tiende a aumentar al redistribuir las neuronas en redes con un mayor número de capas ocultas, como se observa en la Fig. 4.3. Este comportamiento se debe a que HLS4ML implementa las redes siguiendo una arquitectura de *streaming*, donde las salidas de cada capa se conectan en las entradas de la siguiente, formando un *pipeline* de procesamiento cuyo tamaño depende directamente de la cantidad de capas ocultas. Dado que la frecuencia de reloj se mantiene fija, un mayor número de capas implica directamente más ciclos de reloj para completar la inferencia, ya que cada capa requiere al menos un ciclo para procesar y transferir sus resultados a la siguiente.

Por otro lado, aunque en la Figura 4.4 se observa un ligero incremento en la latencia al aumentar la cantidad de entradas, este comportamiento puede explicarse por el mayor número de operaciones realizadas en la primera capa de la red. Sin embargo, esta variación tiene un efecto menor en comparación con el impacto del aumento en el número de capas ocultas.

4.3.5. Observaciones generales

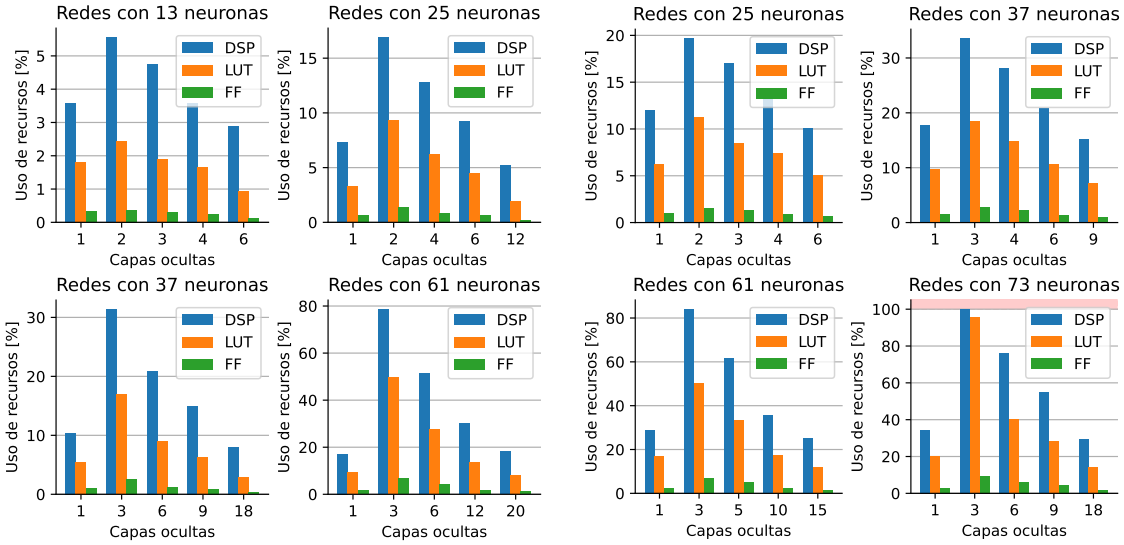
A partir de las observaciones realizadas en esta sección, se concluye que el uso de recursos depende principalmente de la cantidad de multiplicaciones más que del número total de neuronas en la red. Al comparar redes con la misma cantidad de neuronas, aquellas con un mayor número de capas ocultas presentan un menor número de multiplicaciones, lo que implica un menor uso de recursos. La única excepción a esta tendencia son las redes de una sola capa oculta, que presentan un menor uso de recursos que las redes de dos capas. No obstante, aunque no es abordado en este capítulo, es relevante considerar que una red con una sola capa oculta podría no poseer la capacidad suficiente, o requerir de un alto número de neuronas, para aprender la ley de control en ciertas aplicaciones [14].

Asimismo, el uso de recursos también se vio afectado por el número de entradas de las redes, ya que estas modifican la cantidad de multiplicaciones en la primera capa sin alterar el número total de neuronas. Por otro lado, aunque la relación entre el uso de DSPs y las multiplicaciones era previsible, los usos de LUTs y FFs también siguieron una tendencia similar, aunque con menor magnitud. Una posible explicación para esta tendencia en el uso de LUTs es la presencia de operaciones lógicamente más demandantes dentro de la red implementada, como las máquinas de estado inferidas que son necesarias para sincronizar el flujo de datos entre neuronas y otras señales internas, las cuales enmascaran la relación entre el número de neuronas y el uso de LUTs. De manera similar, en el caso de los FFs, aunque las señales se registran después de cada capa, la herramienta de síntesis

infiere registros adicionales según lo considere necesario para garantizar que las señales se propaguen correctamente dentro de los límites temporales establecidos por el período de reloj.

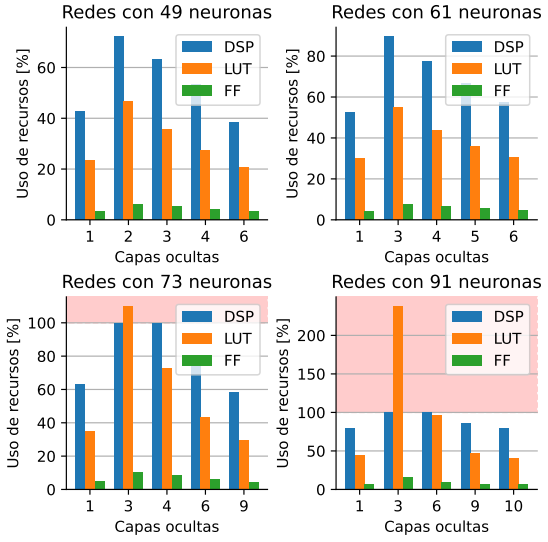
En términos de latencia, se observa una correlación positiva promedio de 0.956, lo que concuerda con la relación esperada con respecto al número de capas ocultas. Esta tendencia confirma que las redes más compactas, es decir, aquellas con menos capas ocultas, presentan una latencia menor en comparación con redes más profundas. Esto se debe a que un mayor número de capas incrementa las etapas necesarias para procesar y registrar las señales a lo largo de la red.

En resumen, para una cantidad constante de neuronas, existe un compromiso de diseño asociado al número de capas ocultas: un mayor número de capas implica un menor uso de recursos debido a la reducción en el número de multiplicaciones, pero también un incremento en la latencia. Adicionalmente, las evaluaciones realizadas destacan que, para niveles de cuantización mayores de 10 bits e implementaciones paralelas, los DSPs se identifican como el recurso crítico, representando consistentemente el mayor porcentaje de utilización en comparación con otros recursos.



(a) Redes neuronales de dos entradas.

(b) Redes neuronales de cuatro entradas.



(c) Redes neuronales de ocho entradas.

Figura 4.1: Uso de recursos para redes con neuronas redistribuidas y diferente número de entradas.

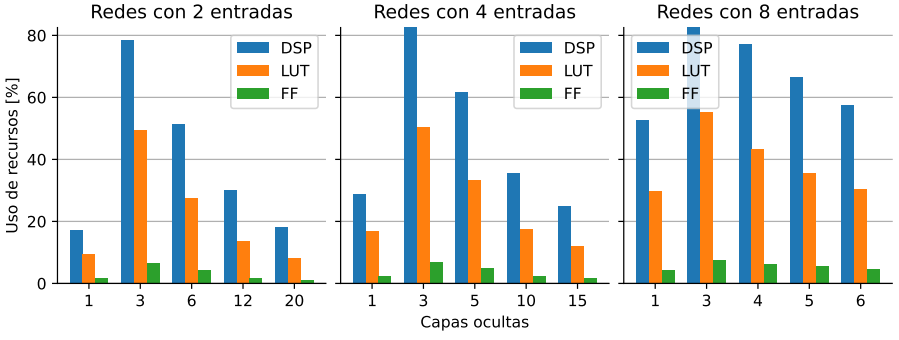


Figura 4.2: Uso de recursos para redes con 61 neuronas y diferentes entradas.

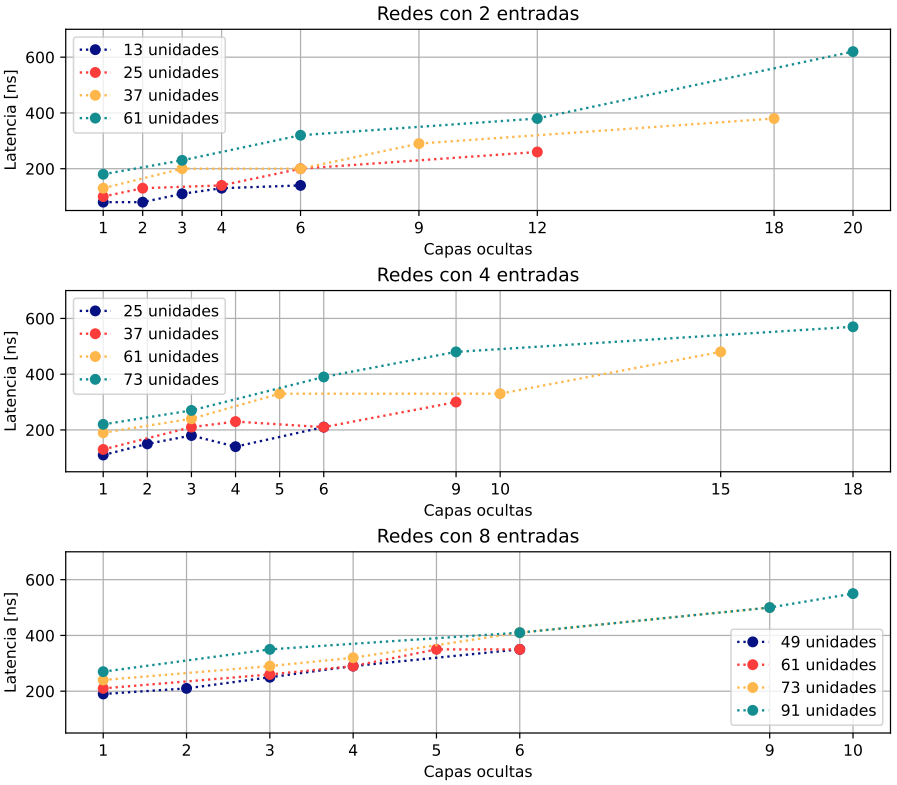


Figura 4.3: Latencia para redes con igual cantidad de neuronas distribuidas de forma diferente.

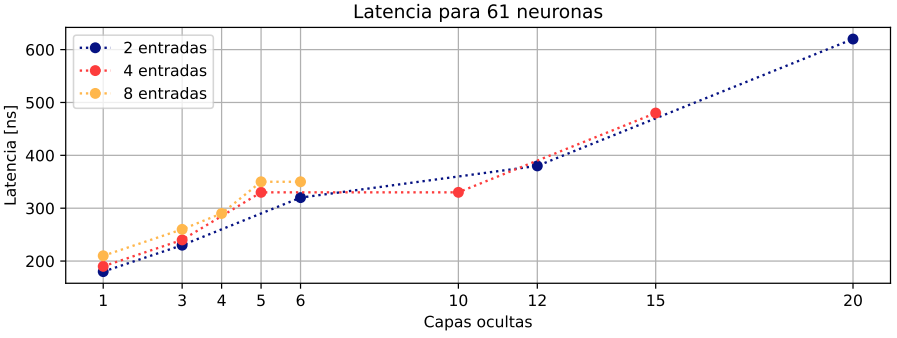


Figura 4.4: Latencia para redes con 61 neuronas y diferentes entradas.

4.4 Redes con diferentes hiperparámetros e igual número de multiplicaciones

4.4.1. Objetivo del experimento

Este estudio tiene como objetivo evaluar cómo varían el uso de recursos y la latencia en modelos de redes neuronales con una cantidad fija de operaciones de multiplicación, distribuidas de diferentes formas mediante combinaciones de hiperparámetros. Este análisis es relevante porque, desde el punto de vista del hardware, las multiplicaciones son la operación más costosa computacionalmente en las redes neuronales utilizadas en este estudio. Por lo tanto, se busca determinar si el uso de recursos y la latencia dependen únicamente del número total de multiplicaciones, o si también están influenciados por otros factores relacionados con los hiperparámetros, como el número de capas ocultas o la cantidad de neuronas por capa.

Dado que en la Sección 4.3 se observó que los tres recursos analizados (DSPs, LUTs y FFs) mostraron tendencias consistentes respecto al número de multiplicaciones, se espera que las redes estudiadas en esta sección tengan un uso de recursos similar. Sin embargo, el uso de FFs podría ser más sensible al número de capas ocultas, ya que cada etapa representa una etapa de procesamiento que debe registrarse.

En cuanto a la latencia, se espera que esta también crezca con el número total de multiplicaciones ya que las observaciones de la Sección 4.3 mostraron que la latencia aumentó con el número de entradas debido al incremento en el número de multiplicaciones. Sin embargo, la cantidad de capas ocultas debería tener un impacto más significativo sobre la latencia de la red implementada.

4.4.2. Condiciones de exploración

Como se muestra en la Tabla 4.3, la exploración para este escenario se llevó a cabo considerando un reloj con período de 10 [ns] y la estrategia *Latency*. Además, se utilizó un factor de reutilización R igual a 1 y un nivel de cuantización W de 32 bits, con una parte entera Q de 8 bits. Adicionalmente, se consideraron únicamente redes con dos entradas (n_x=2) y una salida (n_u=1).

Para la exploración se seleccionaron combinaciones de hiperparámetros que resultaran en el mismo número de multiplicaciones. Así, se analizaron cuatro grupos diferentes con 18, 54, 90 y 270 multiplicaciones, utilizando tres redes distintas para cada caso. Específicamente, para 18 multiplicaciones, las redes seleccionadas fueron 1×6, 2×3 y 4×2; para 54 multiplicaciones, 1×18, 2×6 y 6×3; para 90 multiplicaciones, 1×30, 3×6 y 4×5; y para 270 multiplicaciones, 2×15, 4×9 y 8×6.

Tabla 4.3: Parámetros utilizados para la exploración de redes con diferentes hiperparámetros y la misma cantidad de multiplicaciones.

Parámetro	**Valor**
Período	10 ns
Estrategia	*Latency*
R	1
W	32
Q	8
n_x	2
n_u	1
Multiplicaciones	{18, 54, 90, 270}
Hiperparámetros $(L{\times}M)_{\text{mult=18}}$	{1×6, 2×3, 4×2}
Hiperparámetros $(L{\times}M)_{\text{mult=54}}$	{1×18, 2×6, 6×3}
Hiperparámetros $(L{\times}M)_{\text{mult=90}}$	{1×30, 3×6, 4×5}
Hiperparámetros $(L{\times}M)_{\text{mult=270}}$	{2×15, 4×9, 8×6}

Tabla 4.4: Resultados de síntesis lógica para redes neuronales con igual cantidad de multiplicaciones.

Multiplicaciones	**Arquitectura** ($L \times M$)	**DSPs** - (%)	**FFs** - (%)	**LUTs** - (%)	**Latencia** (ciclos)
18	1×6	36 (2.08)	540 (0.12)	1,447 (0.63)	5
18	2×3	36 (2.08)	574 (0.12)	1,419 (0.62)	8
18	4×2	36 (2.08)	450 (0.10)	1,354 (0.59)	10
54	1×18	108 (6.25)	2,235 (0.49)	4,324 (1.88)	9
54	2×6	108 (6.25)	1,531 (0.33)	4,590 (1.99)	8
54	6×3	108 (6.25)	1,478 (0.32)	4,153 (1.80)	20
90	1×30	178 (10.30)	3,829 (0.83)	7,742 (3.36)	12
90	3×6	178 (10.30)	2,549 (0.55)	7,822 (3.39)	11
90	4×5	180 (10.42)	2,559 (0.56)	7,697 (3.34)	15
270	2×15	530 (30.67)	9,165 (1.99)	25,339 (11.00)	14
270	4×9	538 (31.13)	8,931 (1.94)	24,154 (10.48)	22
270	8×6	538 (31.13)	7,431 (1.61)	23,694 (10.28)	26

4.4.3. Análisis de uso de recursos

Los reportes de uso de recursos mostrados en la Tabla 4.4 indican que, para una cantidad fija de operaciones de multiplicación, las redes neuronales tienden a utilizar una cantidad similar de bloques DSP. En los ejemplos analizados, el uso de DSPs es aproximadamente el doble del número de multiplicaciones de la red, lo que indica que se utilizan dos DSPs por multiplicador de 32 bits. Además, se observa que el uso de FFs aumenta conforme incrementa el número de multiplicaciones. Aunque el uso de este recurso es similar entre redes con el mismo número de multiplicaciones, tiende a ser ligeramente mayor en las configuraciones con mayor número de neuronas por capa. Este aumento probablemente está relacionado con la cantidad de salidas generadas por cada capa, las cuales suelen registrarse para ser utilizadas por la siguiente capa en el próximo ciclo de reloj. En contraste, el uso de LUTs no muestra una tendencia consistente al aumentar el número de neuronas por capa, lo que impide establecer una relación directa entre el uso de LUTs y la implementación de funciones de activación dentro de la red.

4.4.4. Análisis de latencia

En términos de latencia, la Tabla 4.4 muestra un aumento con la cantidad de multiplicaciones, además de verse influenciada por el número de capas ocultas de la red neuronal. Si bien el incremento en la latencia con respecto al número de capas es esperable, ya que un mayor número de capas introduce más etapas de procesamiento y aumenta el tiempo total de inferencia, no lo es con respecto al número de multiplicaciones. Esto se debe a que el factor de reutilización empleado ($R = 1$) indica paralelización completa, lo que debería evitar que la latencia de la capa dependa del número de multiplicaciones ejecutada.

Un aspecto llamativo se observa en los grupos de 54 y 90 multiplicaciones. En el caso de 54 multiplicaciones, al pasar de 1 a 2 capas ocultas, y en el caso de 90 multiplicaciones, al pasar de 1 a 3 capas ocultas, se registra una reducción de un ciclo en la latencia de la red. Este comportamiento puede explicarse porque distribuir las multiplicaciones en un mayor número de capas simplifica la planificación de las operaciones, lo que podría haber permitido reducir las etapas de procesamiento dentro de cada capa. Esta simplificación compensaría el efecto de los registros adicionales introducidos por el incremento en el número de capas, resultando en una latencia general menor.

4.4.5. Observaciones generales

A partir de la exploración realizada, se confirma que redes con diferentes hiperparámetros, pero igual número de multiplicaciones, tienen un uso similar de DSPs, LUTs y FFs. Sin embargo, esta conclusión está condicionada a que el nivel de cuantización sea constante y el factor de reutilización sea igual a uno. Para factores de reutilización mayores que uno, esta afirmación podría no ser

válida, incluso si dicho factor es uniforme para todas las redes. Un análisis de esto se realiza en la Sección 4.7. Además, considerando redes con la misma cantidad de multiplicaciones, aunque el uso de FFs tiende a aumentar con el número de neuronas por capa, no se observa una tendencia equivalente en el uso de LUTs. A pesar de estas diferencias, las variaciones en el uso de recursos no son significativas, lo que permite concluir que el uso de recursos de una red neuronal depende principalmente del número de multiplicaciones que esta realice.

Finalmente, a diferencia de lo observado en el uso de recursos, la latencia no permanece acotada entre redes con el mismo número de multiplicaciones. No obstante, se confirma una correlación positiva de 0.745 entre la latencia y el número de multiplicaciones. Esta relación, se suma al impacto de la cantidad de capas ocultas de la red, que es la principal causa de las diferencias de latencia entre redes con igual número de multiplicaciones.

4.5 Redes con diferentes hiperparámetros y número de multiplicaciones

4.5.1. Objetivo del experimento

El objetivo de este estudio es analizar cómo el número de multiplicaciones influye en el uso de recursos y la latencia. A diferencia del experimento descrito en la Sección 4.3, este análisis no se limita a redes con el mismo número de neuronas, sino que considera diferentes combinaciones de hiperparámetros para evaluar el impacto del número de multiplicaciones en las implementaciones de hardware.

Basándose en los resultados de las Secciones 4.3 y 4.4, se conoce de antemano que existe una correlación positiva entre número de multiplicaciones y el uso de DSPs, LUTs y FFs. Este estudio busca confirmar dicha tendencia ampliando el rango de casos de estudio y, además, identificar con mayor precisión el punto en que la tendencia cambia debido al agotamiento de los DSPs disponibles.

En cuanto a la latencia, se espera observar un incremento asociado al aumento de la cantidad de capas ocultas, y en menor medida del número de multiplicaciones, manteniendo la correlación positiva identificada en experimentos previos.

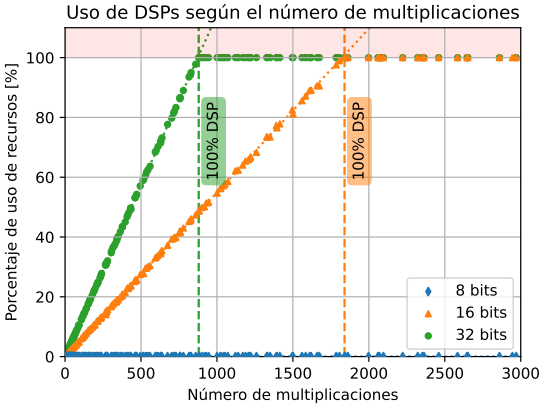
4.5.2. Condiciones de exploración

La Tabla 4.5 presenta los parámetros utilizados en las implementaciones de este experimento. En particular, se emplea un período de reloj de 10 [ns] y la estrategia *Latency*, que corresponden a los valores por predeterminados de Vitis HLS y HLS4ML, respectivamente. Para evitar que el efecto de la reutilización enmascare el uso de recursos, se fija el factor de reutilización R igual a uno.

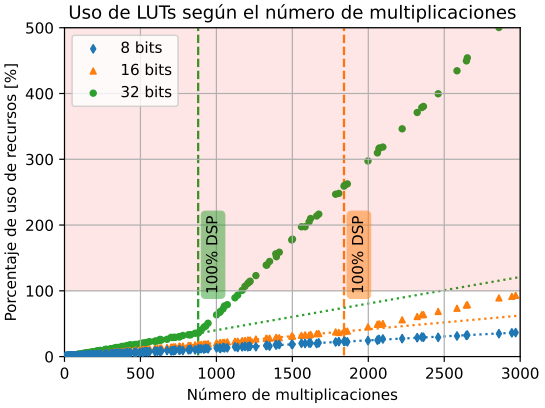
Como parte del estudio, la caracterización incluye tres niveles de cuantización W: 8, 16 y 32 bits, con tamaños de parte entera Q de 2, 4 y 8 bits, respectivamente. Las redes implementadas se configuran con dos entradas (n_x=2) y una salida (n_u=1).

Tabla 4.5: Parámetros utilizados para la exploración de redes con diferentes hiperparámetros y diferente cantidad de multiplicaciones.

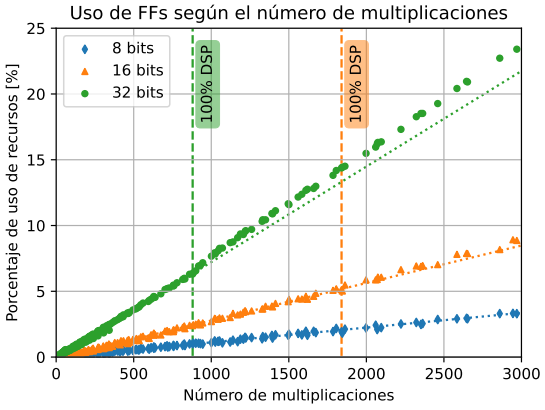
Parámetro	**Valor**
Período	10 ns
Estrategia	*Latency*
R	1
W	{8, 16, 32}
Q	{2, 4, 8}
n_x	2
n_u	1
L	[1, 10]
M	[2, 30]



(a) Uso de DSP.



(b) Uso de LUTs.



(c) Uso de FFs.

Figura 4.5: Uso de recursos de redes neuronales según la cantidad de multiplicaciones y nivel de cuantización.

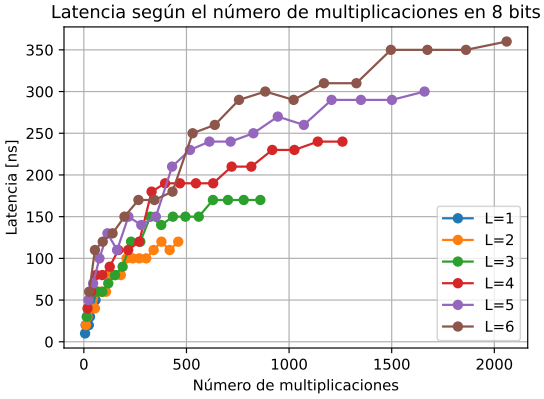
La exploración se lleva a cabo mediante un barrido de combinaciones de hiperparámetros, que abarca desde 1 hasta 10 capas ocultas (L=$[1, 10]$) y entre 2 a 30 neuronas por capa (M=$[2, 30]$) para cada cantidad de capas. Este enfoque permite cubrir un rango significativo de números de multiplicaciones, abarcando los tamaños más frecuentes observados en redes del estado del arte.

4.5.3. Análisis de uso de recursos

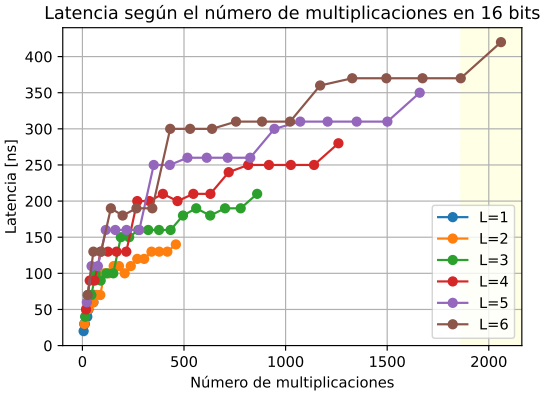
En la Figura 4.5 se muestra el comportamiento del uso de cada recurso a medida que incrementa la cantidad de operaciones de multiplicación de la red. Las etiquetas verticales en cada gráfica indican el número de multiplicaciones a partir del cual los bloques DSP alcanzan su máxima utilización. Para destacar este comportamiento, las líneas punteadas de cada gráfico corresponden a las tendencias calculadas a partir de los datos, excluyendo los puntos donde existe saturación.

En la Figura 4.5a se observa que, para 8 bits, no se utilizan DSPs, lo que refleja el comportamiento de la herramienta al emplear LUTs para multiplicaciones con menos de 10 bits. Para 32 bits de cuantización, el uso de bloques DSP es aproximadamente el doble que para 16 bits, alcanzando la saturación al llegar al 100 % de la capacidad de la tarjeta.

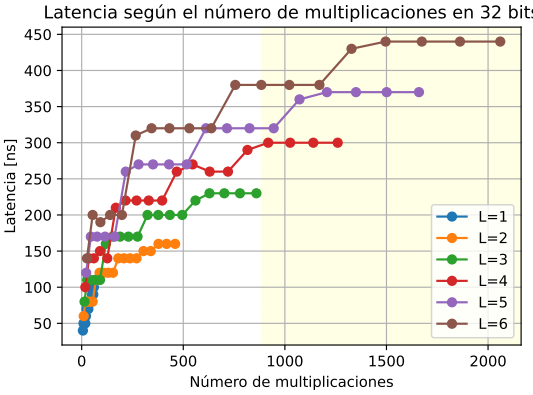
En la Figura 4.5b, se aprecia un efecto similar al observado en la Figura 4.1c, donde la saturación de bloques DSP incrementa rápidamente el requerimiento de LUTs, haciendo inviable la implementación de la red neuronal. Este fenómeno se identifica en los casos de 16 y 32 bits de la Figura 4.5b, que muestra un punto en que se alcanza la capacidad máxima de DSPs, tras lo cual la tendencia del uso de LUTs se quiebra y aumenta su pendiente. Para el caso de 32 bits, este fenómeno limita el número máximo de multiplicaciones antes de agotar las LUTs, siendo este valor



(a) 8 bits.



(b) 16 bits.



(c) 32 bits.

Figura 4.6: Latencia según el número de multiplicaciones.

considerablemente menor que el proyectado si no considerara la saturación de los bloques DSP. Por otro lado, la Figura 4.5b también muestra que, incluso sin utilizar bloques DSP, las redes con 8 bits tienen un menor uso de LUTs en comparación con las de 16 bits, una diferencia que se acentúa tras el agotamiento de los DSP en el caso de 16 bits.

Finalmente, la Figura 4.5c muestra la relación entre el uso de FFs y la cantidad de multiplicaciones en las redes neuronales. Aunque este recurso también experimenta un leve aumento en su pendiente tras el agotamiento de los DSP, el efecto es mucho menos pronunciado que en el caso de las LUTs. Además, este incremento no representa una restricción para la implementación, ya que, en todos los casos observados, el uso de FFs crece más lentamente que el de DSPs y LUTs.

4.5.4. Análisis de latencia

En relación con la latencia, la Figura 4.6 muestra, por separado, los resultados de la latencia obtenidos para cada nivel de cuantización considerado. En las gráficas, se resalta con sombra amarilla la región donde el uso de DSPs alcanza su capacidad máxima. Debido a la influencia, observada en secciones anteriores, del número de capas ocultas sobre la latencia, los resultados se agrupan según la cantidad de capas ocultas. Para evitar una sobrecarga visual, las gráficas se han limitado a mostrar hasta seis capas ocultas, aunque durante el experimento se consideraron hasta diez. No obstante, los experimentos que no fueron ilustrados no aportan información adicional para el análisis realizado en esta sección.

En la Figura 4.6a, los datos evidencian una relación aproximadamente logarítmica entre el número de multiplicaciones y la latencia. La Figura 4.6b muestra una tendencia similar, aunque comienzan a aparecer rangos donde la latencia no varía significativamente. Por su parte, la Figura 4.6c también presenta una relación creciente entre el número de multiplicaciones y la latencia, pero con un

comportamiento escalonado. Este patrón sugiere que, para un número fijo de capas ocultas, existen rangos de multiplicaciones donde el incremento en la latencia no es considerable. Finalmente, a diferencia del análisis de uso de recursos, no se observa una alteración en la tendencia de la latencia al alcanzar la zona que indica el uso completo de los DSPs, marcada con un fondo amarillo claro en las Figuras 4.6b y 4.6c.

4.5.5. Observaciones generales

En general, las curvas observadas en este experimento evidencian una relación aproximadamente lineal entre el número de multiplicaciones y el uso de DSPs, LUTs y FFs. Sin embargo, esta tendencia se quiebra al alcanzar el punto en que se agotan los bloques DSP. En este punto, la herramienta de síntesis comienza a implementar los multiplicadores utilizando los demás recursos disponibles.

Conociendo la relación entre el número de multiplicaciones y el uso de recursos, es posible estimar el uso de recursos de una red neuronal a partir del número de operaciones de multiplicación. Además, los resultados indican que trabajar con 32 bits de cuantización agota los recursos disponibles antes que hacerlo con 16 bits, y de manera similar, las configuraciones de 8 bits permiten un uso más eficiente de los recursos. Sin embargo, si bien se puede extraer un rango de redes factibles de implementar desde la Fig. 4.5, este rango podría ampliarse con los análisis adicionales que se presentarán en las siguientes secciones.

En cuanto a la latencia, se confirmó la tendencia al alza con respecto al número de multiplicaciones. En particular, la influencia del número de multiplicaciones se manifiesta más claramente en el rango extendido considerado, lo que permitió observar casos donde redes con menos capas ocultas presentan una mayor latencia que redes más profundas, pero con un menor número total de multiplicaciones. Este comportamiento sugiere que, aunque el factor de reutilización es igual a uno, las operaciones dentro de cada capa no se ejecutan completamente en paralelo.

4.6 Redes con diferentes niveles de cuantización

4.6.1. Objetivo del experimento

Este estudio apunta a analizar el impacto que tiene el nivel de cuantización elegido sobre el uso de recursos y la latencia de una red neuronal implementada. Dado que el flujo de diseño propuesto en el Capítulo 3 contempla un entrenamiento cuantizado de la red neuronal con los hiperparámetros previamente escogidos, este análisis también ofrece información sobre los niveles de cuantización factibles, así como las latencias que puede alcanzar el hardware bajo la combinación de hiperparámetros seleccionada. Una elección guiada de la cuantización permite al diseñador de hardware evitar un barrido exhaustivo de alternativas, enfocándose en opciones con alta factibilidad y resultados prometedores.

A partir de los datos obtenidos, se busca establecer directivas generales sobre el grado de reducción en el uso de recursos a medida que disminuye el nivel de cuantización. Adicionalmente, para el caso de los bloques DSP, se busca determinar los puntos en que aumenta la cantidad de DSP por multiplicación.

En términos de latencia, también se espera observar una disminución a medida que se reduce la cantidad de bits empleados para la cuantización.

4.6.2. Condiciones de exploración

La Tabla 4.6 resume los parámetros empleados en las implementaciones realizadas en esta sección. Como en los experimentos anteriores, las redes se implementaron utilizando un período de reloj de 10 [ns], una estrategia *Latency* y un factor de reutilización R igual a uno. Para este experimento, se consideraron redes con cuatro entradas (n_x=4) y una salida (n_u=1).

Adicionalmente, se definieron ocho combinaciones de hiperparámetros para analizar los reportes de síntesis lógica en términos de uso de recursos y latencia. Estas combinaciones corresponden a redes de 3 capas ocultas, con 8, 12, 20 y 24 neuronas por capa, y redes de 6 capas ocultas, con 4, 6,

Tabla 4.6: Parámetros utilizados para la exploración de redes con diferentes niveles de cuantización.

Parámetro	**Valor**
Período	10 [ns]
Estrategia	*Latency*
R	1
n_x	4
n_u	1
L	{ 3, 6}
$M_{L=3}$	{8, 12, 20, 24}
$M_{L=6}$	{4, 6, 10, 12}
W	[2, 32]
Q	$\lceil \frac{W}{4} \rceil$

10 y 12 neuronas por capa. Para cada combinación de hiperparámetros, se evaluaron niveles de cuantización W desde 2 hasta 32 bits, asegurando que la cantidad de bits de la parte entera Q fuera al menos igual a una cuarta parte del total de bits.

4.6.3. Análisis de uso de recursos

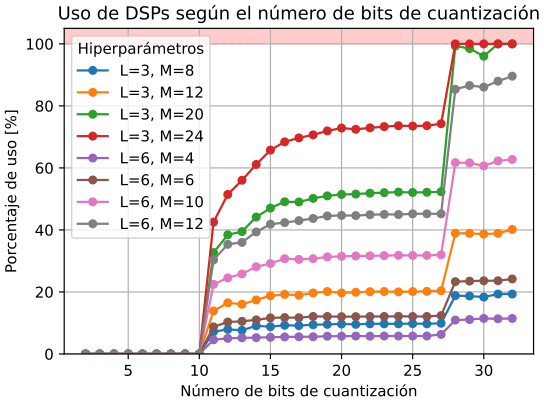
Los reportes de uso de recursos para las redes consideradas se presentan en la Figura 4.7. Los recursos se expresan como porcentajes respecto a la disponibilidad indicada en la Tabla 4.1, destacándose con una sombra roja las regiones donde los requerimientos superan los recursos disponibles. La Figura 4.7a detalla el uso de bloques DSP, mostrando que estos no se utilizan cuando la cuantización es de 10 bits o menos. Aunque la documentación más reciente encontrada que menciona este comportamiento data de 2017 [61], donde se indica que los multiplicadores de 10 bits o más se implementan con DSPs, los experimentos realizados sugieren que, en la práctica, las multiplicaciones deben superar los 10 bits para que la herramienta asigne DSPs. Este comportamiento no ha sido mencionado explícitamente en la documentación más reciente revisada [62], pero los resultados obtenidos confirman que se mantiene en la versión de la herramienta utilizada para este trabajo.

Aunque es esperable que la cantidad de bloques DSP se mantenga constante, dado que el número de multiplicaciones no varía con la cuantización, en la Figura 4.7a se nota un ligero incremento en el uso de bloques DSP a medida que aumenta el número de bits de cuantización. Este crecimiento progresivo parece ser resultado de una optimización de Vivado, que distribuye el uso de recursos de manera gradual para evitar un incremento abrupto en la utilización de bloques DSP, implementando algunos multiplicadores completamente con LUTs.

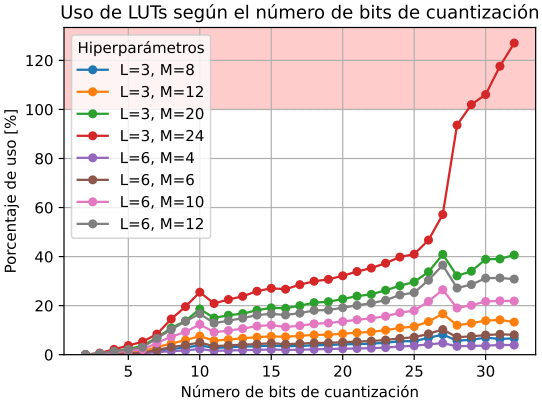
Según la documentación, Vitis HLS complementa los DSPs con LUTs o más DSPs para implementar multiplicadores cuando las entradas exceden sus capacidades. En el caso específico de la tarjeta utilizada, los bloques DSP48E tienen entradas de 27×18 [63]. Como se observa en la Figura 4.7a, el uso de DSPs no aumenta drásticamente con al superar los 18 bits, pero sí al superar los 27 bits. Esto sugiere que, después de los 18 bits, la herramienta complementó los DSPs con LUTs para implementar multiplicadores, y que desde los 28 bits, comenzó a emplear un segundo DSP por multiplicador.

Además, la Figura 4.7a muestra que la red con hiperparámetros L=3 y M=24 (en rojo) utiliza completamente los bloques DSP disponibles a partir de los 28 bits. Según la ecuación (3.1), esta red requiere realizar 1272 multiplicaciones. Sin embargo, al necesitar el doble de bloques DSP por multiplicación, el requerimiento aumenta a 2544 bloques DSP, excediendo en 816 la cantidad disponible en el hardware. Este déficit obliga a la herramienta a emplear LUTs para implementar los multiplicadores restantes.

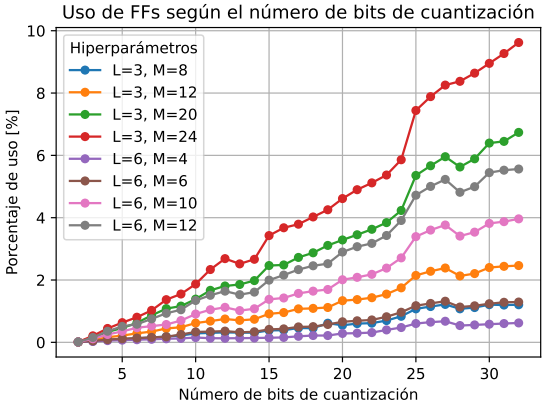
La Figura 4.7b muestra cómo crece el uso de LUTs a medida que se emplea una mayor cantidad de bits para la representación numérica de los datos en la red. En general, el aumento en el uso de LUTs es gradual con la cantidad de bits, pero se dispara al agotarse los bloques DSP. Esto puede observarse en el caso de la red con hiperparámetros L=3 y M=24 (en rojo), que, al saturar el uso de bloques DSP, comienza a utilizar LUTs para implementar las instancias de multiplicación



(a) Uso de bloques DSP según el nivel de cuantización para diferentes redes neuronales.



(b) Uso de LUTs según el nivel de cuantización para diferentes redes neuronales.



(c) Uso de FFs según el nivel de cuantización para diferentes redes neuronales.

Figura 4.7: Uso de recursos para redes neuronales según el número de bits de cuantización.

adicionales requeridas por la red neuronal. Este elevado requerimiento de LUTs vuelve inviable la red mencionada cuando se cuantiza con más de 28 bits.

Por otro lado, la red con hiperparámetros L=3 y M=20 (en verde) también supera ligeramente la cantidad de bloques DSP disponibles, requiriendo 72 bloques DSP adicionales. Sin embargo, el impacto sobre las LUTs en este caso es mucho menor que en el de la red con L=3 y M=24. Adicionalmente, se observa una ligera disminución en el uso de LUTs para las cuantizaciones de 11 y 28 bits, coincidiendo con los escalones en el uso de bloques DSP. Esto se debe a que, a partir de esos niveles de cuantización, el uso de bloques DSP permitió liberar parte de las LUTs previamente utilizadas para implementar multiplicadores.

La Figura 4.7c muestra que el uso de FFs también experimenta algunos cambios en función del nivel de cuantización. Aunque los aumentos observados no ocurren con la misma cantidad de bits que en los casos anteriores (Figs. 4.7a y 4.7b), la disminución registrada con 28 bits si está relacionada con las demás figuras. Esto probablemente se deba al incremento en el uso de bloques DSP, los cuales incluyen registros que pueden sustituir algunos FFs.

4.6.4. Análisis de latencia

El análisis del comportamiento de la latencia en función de la cantidad de bits de cuantización para las redes estudiadas se presenta en la Figura 4.8. En general, a partir de los 4 bits, el nivel de cuantización afecta la latencia de la red neuronal de manera leve. Sin embargo, cuando la cuantización utiliza menos de 4 bits, se nota una variación significativa en la latencia con respecto

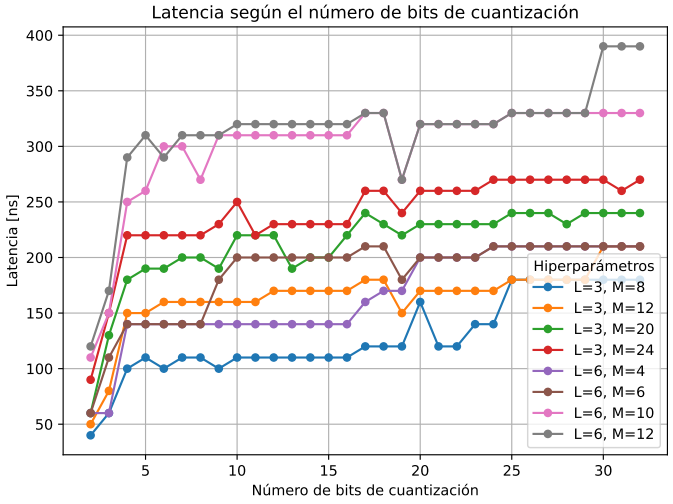


Figura 4.8: Latencia para redes neuronales según el número de bits de cuantización.

al número de bits. Esta disminución puede explicarse porque, con niveles tan bajos de cuantización, las operaciones de cada capa se aceleran al punto de permitir agrupar una mayor cantidad de cálculos en cada ciclo de reloj, reduciendo así la necesidad de registros intermedios y disminuyendo la latencia.

Aunque, en términos generales, la latencia aumenta con la cantidad de bits de cuantización, este incremento se manifiesta de forma abrupta en ciertos rangos de cuantización, como entre 17-20 bits y 28-30 bits. Además, como se analizó en la sección anterior, la Figura 4.8 confirma que, aunque la latencia de la red implementada depende del nivel de cuantización, está influenciada principalmente por la cantidad de capas ocultas y el número de multiplicaciones realizadas.

El comportamiento de la latencia en la Figura 4.8 probablemente se deba a cambios en la implementación de los multiplicadores. Desde 19 bits, el bloque DSP debe complementarse con LUTs para ejecutar la multiplicación, y a partir de los 28 bits los reportes indican que es necesario utilizar dos DSP por multiplicador. Además, es observable que la latencia no aumenta de manera continua con el número de bits, ya que en algunos casos se notan caídas puntuales. Estas caídas podrían explicarse por una mejor adaptación del diseño sintetizado a los recursos disponibles en la tarjeta, ya sea debido a una mayor compatibilidad entre ambos o a ajustes en los algoritmos de optimización de la herramienta de síntesis.

4.6.5. Observaciones generales

En general, los bloques DSP se identifican como el recurso mayormente afectado por variaciones en el nivel de cuantización. Una vez que se ha ocupado la capacidad máxima de DSPs, la herramienta utiliza LUTs para implementar las operaciones de multiplicación adicionales. Esta solución incrementa rápidamente el uso de LUTs, pudiendo volverse inviable la implementación. El análisis muestra que, a partir de 28 bits de cuantización, se utilizan dos bloques DSP por multiplicación, lo que incrementa significativamente el uso de estos recursos y hace más probable alcanzar la saturación de los DSPs disponibles.

En contraste, se observa que las redes con requerimientos más extensivos pueden obtener una reducción significativa del uso de recursos al disminuir su nivel de cuantización por debajo de los 10 bits. En este rango, los DSPs no se utilizan, y las operaciones de multiplicación se implementan completamente con LUTs, lo que contribuye a un uso más eficiente de los recursos disponibles. Sin embargo, este análisis no considera la precisión del modelo, por lo cuál la factibilidad de esta optimización queda sujeta a la tolerancia de la aplicación.

Por otro lado, el uso de FFs no resulta tan crítico como el de los bloques DSP y las LUTs, ya que, en los casos estudiados, su utilización no superó el 10 % de la capacidad total, incluyendo los casos inviables de implementar.

En relación con la latencia, aunque utilizar un nivel de cuantización más bajo puede reducirla, los datos muestran que esta disminución es leve y no se observa consistentemente en todos los niveles de cuantización. Por el contrario, si la aplicación requiere una mayor precisión, es posible emplear un nivel de cuantización más alto sin esperar un incremento significativo en la latencia.

4.7 Redes con diferente arquitectura de hardware

4.7.1. Objetivo del experimento

El objetivo de este análisis es evaluar la efectividad del factor de reutilización como técnica para reducir el uso de recursos y su impacto en la latencia de la inferencia bajo diferentes niveles de cuantización y combinaciones de hiperparámetros.

El factor de reutilización en HLS4ML controla cuántas veces se utiliza un multiplicador dentro de una misma capa, definiendo indirectamente el nivel de paralelismo espacial del hardware. Al usar un valor de reutilización igual a uno se espera que la implementación sea completamente paralela, mientras que, con el valor máximo, equivalente al número de multiplicaciones en la capa (M^2), se espera un único multiplicador que opere de forma secuencial. Es importante destacar que este comportamiento se aplica de manera independiente en cada capa, lo que significa que el número mínimo de multiplicadores requeridos para una red corresponde al número de capas de esta. Este análisis amplía la exploración propuesta en el Capítulo 3, considerando una mayor diversidad de condiciones para evaluar con más detalle el comportamiento del factor de reutilización.

Así, se espera que el incremento del factor de reutilización permita reducir efectivamente el uso de DSPs, y en menor medida LUTs y FFs, a la vez que incrementa significativamente la latencia.

4.7.2. Condiciones de exploración

La Tabla 4.7 resume los parámetros utilizados para este experimento. Al igual que en los experimentos anteriores, se considera un período de reloj de 10 [ns] para la síntesis y una estrategia *Latency* para HLS4ML. Además, se evalúan los niveles de cuantización W de 8, 16 y 32 bits, asignando 2, 4 y 8 bits a la parte entera Q, respectivamente.

Las redes consideradas tienen cuatro entradas (n_x=4) y una salida (n_u=1), con los hiperparámetros 3×24 y 6×12. Estos hiperparámetros son escogidos debido a que representan tamaños similares a los utilizados en aplicaciones de MPC [12, 19]. Dos casos adicionales a esta exploración se incorporan en el Anexo C.

Para cada red, se explora una lista de factores de reutilización que incluyen valores consecutivos desde 1 hasta el número de neuronas por capa (M), junto con los resultados de dividir el total de multiplicaciones por capa (M^2) entre cada entero desde 1 hasta $M-1$, aproximando cada división al entero superior. En el caso de 3×24, los factores de reutilización abarcan valores desde 1 y 576, mientras que para 6×12, el rango incluye desde 1 hasta 144. Los valores seleccionados en ambos

Tabla 4.7: Parámetros utilizados para la exploración de factores de reutilización para redes neuronales.

Parámetro	**Valor**
Período	10 ns
Estrategia	*Latency*
W	$\{8, 16, 32\}$
Q	$\lceil \frac{W}{4} \rceil$
n_x	4
n_u	1
Hiperparámetros ($L \times M$)	{ 3×24, 6×12 }
$R_{3\times24}$	$[1, 24] \cup \{ \lceil \frac{576}{x} \rceil \mid x \in [1, 23] \}$
$R_{6\times12}$	$[1, 12] \cup \{ \lceil \frac{144}{x} \rceil \mid x \in [1, 11] \}$

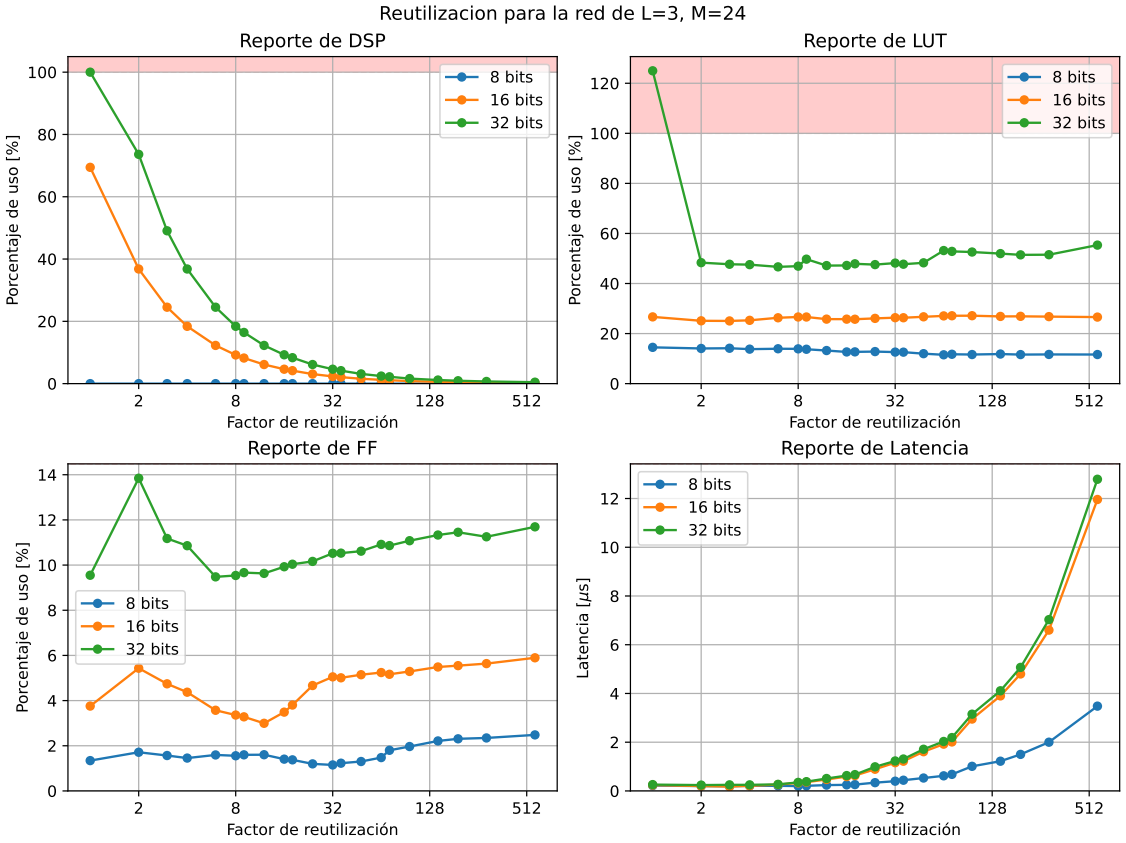


Figura 4.9: Reportes de implementación de la red de 3×24 con diferentes factores de reutilización.

casos están diseñados para evaluar diferentes cantidades de multiplicadores inferidos, con el objetivo de obtener datos representativos sobre el uso de recursos y la latencia.

4.7.3. Análisis de reportes para una red neuronal de 3×24

Para la red de 3×24, la Fig. 4.9 presenta los usos de recursos y la latencia reportados por la implementación, considerando factores de reutilización desde uno hasta 576. Complementariamente, la Tabla 4.8 muestra algunos de los puntos representados en la Fig. 4.9, destacando con colores las variaciones porcentuales en los reportes a medida que aumenta el factor de reutilización. En particular, a partir de la Tabla 4.8 y la Figura 4.9, se observa lo siguiente:

- En términos del uso de bloques DSP, la Figura 4.9 muestra una relación inversamente proporcional entre el factor de reutilización y el uso de DSPs. Aunque con 8 bits no se utilizan DSPs, es evidente que con 32 bits se emplea el doble de bloques que con 16 bits, como se puede corroborar en la Tabla 4.8. A pesar de que el rango de factores de reutilización es extenso debido a la cantidad de neuronas por capa de la red, la Tabla 4.8 muestra que un factor de reutilización igual a 16 es suficiente para reducir el uso de DSPs en más de un 90 %.

  Cabe destacar el caso de 32 bits con reutilización uno, donde se alcanza el máximo uso de bloques DSP. En este escenario, la red neuronal debe realizar 1272 multiplicaciones y, dado que cada multiplicador de 32 bits utiliza dos bloques DSP, existe un déficit de 816 bloques, los cuales deben implementarse utilizando LUTs. Al aumentar el factor de reutilización a dos, el déficit de bloques DSP desaparece, y el uso de DSPs sigue una tendencia inversamente proporcional al factor de reutilización.

  Para el caso de 16 bits, aunque no se ha utilizado completamente la capacidad de la tarjeta, la implementación ha ahorrado 72 bloques DSP. Es interesante que este tipo de optimización

Tabla 4.8: Reportes de implementación para la red de 3×24 bajo diferentes factores de reutilización y niveles de cuantización (8, 16 y 32 bits), mostrando el impacto en el uso de DSPs, LUTs, FFs y latencia. Las columnas coloreadas representan las variaciones porcentuales respecto al caso con factor de reutilización $R = 1$. En todos los casos, el color verde indica una reducción y el rojo un aumento.

W bits	***R***	**DSPs**	**Δ DSPs** % de R=1	**LUTs**	**Δ LUTs** % de R=1	**FFs**	**Δ FFs** % de R=1	**Latencia** *ns*	**Δ Latencia** % de R=1
8	1	0 (0.0 %)	—	33396 (14.5 %)	—	6188 (1.3 %)	—	220	—
8	2	0 (0.0 %)	0.0 %	32372 (14.1 %)	-3.1 %	7893 (1.7 %)	27.6 %	200	-9.1 %
8	4	0 (0.0 %)	0.0 %	31743 (13.8 %)	-4.9 %	6705 (1.5 %)	8.4 %	210	-4.5 %
8	8	0 (0.0 %)	0.0 %	32013 (13.9 %)	-4.1 %	7175 (1.6 %)	16.0 %	200	-9.1 %
8	16	0 (0.0 %)	0.0 %	29167 (12.7 %)	-12.7 %	6486 (1.4 %)	4.8 %	250	13.6 %
8	32	0 (0.0 %)	0.0 %	29030 (12.6 %)	-13.1 %	5304 (1.2 %)	-14.3 %	400	81.8 %
8	64	0 (0.0 %)	0.0 %	26538 (11.5 %)	-20.5 %	6805 (1.5 %)	10.0 %	620	181.8 %
8	144	0 (0.0 %)	0.0 %	27234 (11.8 %)	-18.5 %	10200 (2.2 %)	64.8 %	1220	454.5 %
8	576	0 (0.0 %)	0.0 %	26790 (11.6 %)	-19.8 %	11434 (2.5 %)	84.8 %	3480	1481.8 %
16	1	1200 (69.4 %)	—	61444 (26.7 %)	—	17314 (3.8 %)	—	240	—
16	2	636 (36.8 %)	-47.0 %	57831 (25.1 %)	-5.9 %	25039 (5.4 %)	44.6 %	200	-16.7 %
16	4	318 (18.4 %)	-73.5 %	58269 (25.3 %)	-5.2 %	20144 (4.4 %)	16.3 %	210	-12.5 %
16	8	159 (9.2 %)	-86.8 %	61341 (26.6 %)	-0.2 %	15483 (3.4 %)	-10.6 %	330	37.5 %
16	16	80 (4.6 %)	-93.3 %	59351 (25.8 %)	-3.4 %	16078 (3.5 %)	-7.1 %	590	145.8 %
16	32	40 (2.3 %)	-96.7 %	60699 (26.3 %)	-1.2 %	23274 (5.1 %)	34.4 %	1160	383.3 %
16	64	21 (1.2 %)	-98.2 %	62339 (27.1 %)	1.5 %	24154 (5.2 %)	39.5 %	1920	700.0 %
16	144	10 (0.6 %)	-99.2 %	61863 (26.9 %)	0.7 %	25287 (5.5 %)	46.0 %	3900	1525.0 %
16	576	4 (0.2 %)	-99.7 %	61249 (26.6 %)	-0.3 %	27165 (5.9 %)	56.9 %	11960	4883.3 %
32	1	1728 (100 %)	—	287958 (125 %)	—	44006 (9.5 %)	—	260	—
32	2	1272 (73.6 %)	-26.4 %	111363 (48.3 %)	-61.3 %	63798 (13.8 %)	45.0 %	240	-7.7 %
32	4	636 (36.8 %)	-63.2 %	109522 (47.5 %)	-62.0 %	50032 (10.9 %)	13.7 %	250	-3.8 %
32	8	318 (18.4 %)	-81.6 %	108130 (46.9 %)	-62.4 %	43959 (9.5 %)	-0.1 %	350	34.6 %
32	16	160 (9.3 %)	-90.7 %	108770 (47.2 %)	-62.2 %	45739 (9.9 %)	3.9 %	630	142.3 %
32	32	80 (4.6 %)	-95.4 %	110958 (48.2 %)	-61.5 %	48484 (10.5 %)	10.2 %	1230	373.1 %
32	64	42 (2.4 %)	-97.6 %	122500 (53.2 %)	-57.5 %	50294 (10.9 %)	14.3 %	2030	680.8 %
32	144	20 (1.2 %)	-98.8 %	119694 (52.0 %)	-58.4 %	52211 (11.3 %)	18.6 %	4110	1480.8 %
32	576	8 (0.5 %)	-99.5 %	127505 (55.3 %)	-55.7 %	53882 (11.7 %)	22.4 %	12790	4819.2 %

no se manifieste en otros casos con requerimientos similares, como el de 32 bits con factor de reutilización dos.

- Al analizar el uso de LUTs, la Figura 4.9 evidencia un problema crítico para el caso de 32 bits y un factor de reutilización igual a uno, ya que la implementación requiere una cantidad de LUTs superior a la disponible en la tarjeta utilizada. El déficit de 816 bloques DSP con reutilización uno en 32 bits requirió utilizar LUTs para implementar multiplicadores, incrementando drásticamente su uso. Sin embargo, al aumentar el factor de reutilización y eliminar el déficit de DSPs, el uso de LUTs disminuye significativamente, permitiendo que la red neuronal sea viable de implementar.

  El drástico cambio en el uso de LUTs entre los factores de reutilización uno y dos en el caso de 32 bits dificulta identificar una tendencia clara en los niveles de cuantización estudiados. No obstante, la Tabla 4.8 muestra un ligero aumento en el uso de LUTs después de la caída asociada al factor de reutilización dos. En el caso de 16 bits, también se observa una disminución en el uso de LUTs desde el factor de reutilización dos, aunque esta reducción es menor y se desvanece gradualmente a medida que el factor de reutilización aumenta. Incluso, en algunos casos, el uso de LUTs supera el reportado con reutilización uno, como ocurre con los factores 64 y 144, que muestran incrementos del 1.5 % y 0.7 %, respectivamente.

  Por último, para 8 bits, la tendencia es opuesta a los casos de 16 y 32 bits, ya que el uso de LUTs disminuye con el incremento del factor de reutilización, alcanzándose reducciones de hasta el 20.5 % con un factor de reutilización de 64. Aunque el factor de reutilización permite ahorrar LUTs en este caso, no resulta tan efectivo como lo es con los bloques DSP en los casos de mayor nivel de cuantización.

- En cuanto a los FFs, la Figura 4.9 evidencia una tendencia general al alza con el aumento del

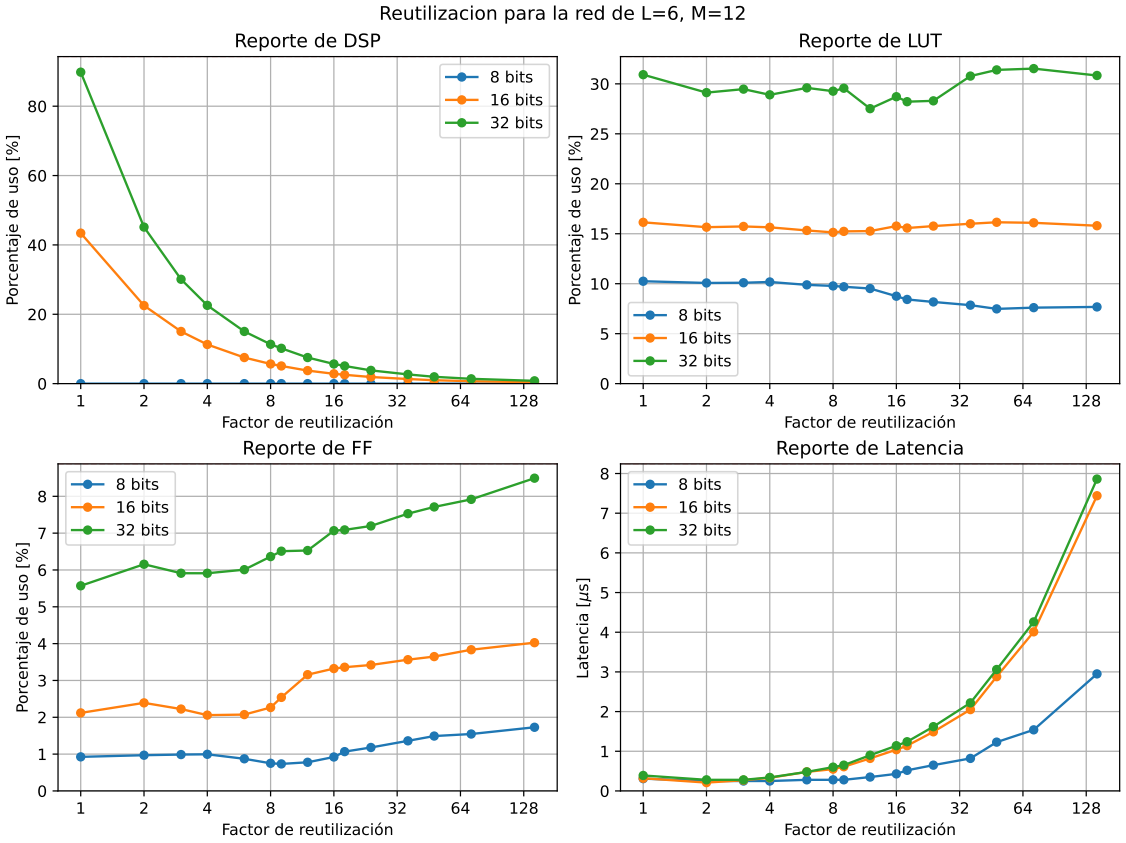


Figura 4.10: Reportes de implementación de la red de 6×12 con diferentes factores de reutilización.

factor de reutilización. Sin embargo, en los tres niveles de cuantización analizados se observa un *peak* en el caso de reutilización dos. De manera similar, también se identificaron puntos en los tres niveles de cuantización donde el uso de FFs fue menor que en la implementación con factor de reutilización uno.

- Por último, con respecto a la latencia, la Figura 4.9 muestra una relación proporcional entre la latencia y el factor de reutilización, con valores significativamente altos para la latencia en el caso de la reutilización máxima considerada con 16 y 32 bits. En el caso de 8 bits, aunque la latencia también aumenta, este incremento es menos pronunciado en comparación con los demás niveles de cuantización. Además, la Tabla 4.8 revela que los factores de reutilización dos y cuatro lograron reducir ligeramente la latencia en comparación con la implementación con reutilización uno, siendo observable en el caso de 8 bits también una reducción con factor de reutilización ocho.

4.7.4. Análisis de reportes para una red neuronal de 6×12

Los reportes de implementación de la red neuronal de 6×12 se analizan a partir de los usos de recursos y la latencia graficados en la Figura 4.10. Algunos de estos datos también se presentan en la Tabla 4.9, que incluye una columna con las variaciones respecto al caso base (factor de reutilización igual a uno) para el mismo nivel de cuantización. Para esta red, los factores de reutilización varían entre uno y 144. De la Figura 4.10 y la Tabla 4.9, se derivan las siguientes observaciones:

- Para el uso de DSPs, los datos confirman la relación inversamente proporcional esperada entre el factor de reutilización y el uso de bloques DSP, además de su ausencia en los casos de 8 bits, en línea con los análisis previos. Aunque la red en este caso requiere realizar 780 multiplicaciones, la Tabla 4.9 muestra que, con un factor de reutilización igual a uno, se

Tabla 4.9: Reportes de implementación para la red de 6×12 bajo diferentes factores de reutilización y niveles de cuantización (8, 16 y 32 bits), mostrando el impacto en el uso de DSPs, LUTs, FFs y latencia. Las columnas coloreadas representan las variaciones porcentuales respecto al caso con factor de reutilización $R = 1$. En todos los casos, el color verde indica una reducción y el rojo un aumento.

W bits	***R***	**DSPs**	**Δ DSPs** % de R=1	**LUTs**	**Δ LUTs** % de R=1	**FFs**	**Δ FFs** % de R=1	**Latencia** ns	**Δ Latencia** % de R=1
8	1	0 (0.0 %)	—	23609 (10.2 %)	—	4264 (0.9 %)	—	310	—
8	2	0 (0.0 %)	0.0 %	23215 (10.1 %)	-1.7 %	4470 (1.0 %)	4.8 %	250	-19.4 %
8	4	0 (0.0 %)	0.0 %	23443 (10.2 %)	-0.7 %	4584 (1.0 %)	7.5 %	250	-19.4 %
8	8	0 (0.0 %)	0.0 %	22528 (9.8 %)	-4.6 %	3460 (0.8 %)	-18.9 %	280	-9.7 %
8	16	0 (0.0 %)	0.0 %	20157 (8.7 %)	-14.6 %	4247 (0.9 %)	-0.4 %	430	38.7 %
8	144	0 (0.0 %)	0.0 %	17686 (7.7 %)	-25.1 %	7966 (1.7 %)	86.8 %	2950	851.6 %
16	1	750 (43.4 %)	—	37190 (16.1 %)	—	9764 (2.1 %)	—	320	—
16	2	389 (22.5 %)	-48.1 %	36072 (15.7 %)	-3.0 %	11021 (2.4 %)	12.9 %	210	-34.4 %
16	4	195 (11.3 %)	-74.0 %	36042 (15.6 %)	-3.1 %	9484 (2.1 %)	-2.9 %	330	3.1 %
16	8	98 (5.7 %)	-86.9 %	34869 (15.1 %)	-6.2 %	10437 (2.3 %)	6.9 %	550	71.9 %
16	16	49 (2.8 %)	-93.5 %	36325 (15.8 %)	-2.3 %	15313 (3.3 %)	56.8 %	1040	225.0 %
16	144	7 (0.4 %)	-99.1 %	36409 (15.8 %)	-2.1 %	18551 (4.0 %)	90.0 %	7440	2225.0 %
32	1	1552 (89.8 %)	—	71236 (30.9 %)	—	25667 (5.6 %)	—	390	—
32	2	780 (45.1 %)	-49.7 %	67099 (29.1 %)	-5.8 %	28364 (6.2 %)	10.5 %	280	-28.2 %
32	4	390 (22.6 %)	-74.9 %	66592 (28.9 %)	-6.5 %	27236 (5.9 %)	6.1 %	340	-12.8 %
32	8	196 (11.3 %)	-87.4 %	67433 (29.3 %)	-5.3 %	29323 (6.4 %)	14.2 %	600	53.8 %
32	16	98 (5.7 %)	-93.7 %	66155 (28.7 %)	-7.1 %	32566 (7.1 %)	26.9 %	1140	192.3 %
32	144	14 (0.8 %)	-99.1 %	71047 (30.8 %)	-0.3 %	39133 (8.5 %)	52.5 %	7860	1915.4 %

ahorran 30 bloques DSP con 16 bits y 8 bloques DSP con 32 bits, reflejando la optimización observada en el caso anterior en la asignación de recursos para estos niveles de cuantización.

- En el caso de las LUTs, la Tabla 4.9 muestra una disminución general del uso de LUTs para factores de reutilización superiores a uno en comparación con el caso base. Sin embargo, en la Figura 4.10 no se aprecia una tendencia clara asociada al factor de reutilización. La excepción corresponde al nivel de cuantización de 8 bits, donde se evidencia una reducción progresiva en el uso de LUTs a medida que aumenta el factor de reutilización.

- En cuanto a los FFs, la Figura 4.10 indica un incremento general en su uso conforme aumenta el factor de reutilización. No obstante, este crecimiento no sigue un comportamiento lineal. La Tabla 4.9 muestra casos específicos, como el de factor de reutilización igual a ocho con 8 bits, donde se observa una reducción en el uso de FFs respecto al caso base.

- Respecto a la latencia, el menor valor se observa consistentemente con un factor de reutilización igual a dos, en lugar del caso que permite ejecución completamente paralela (factor de reutilización igual a uno). En los demás casos, se cumple la tendencia esperada de una relación proporcional entre la latencia y el factor de reutilización, como se analizó previamente.

4.7.5. Discusión y observaciones generales

En términos generales, la latencia y el uso de bloques DSP responden a una relación marcada con el factor de reutilización. Aunque la latencia generalmente aumenta proporcionalmente al factor de reutilización, en varios de los casos estudiados, la latencia mínima se alcanzó con un factor de reutilización $R = 2$ o incluso con $R = 4$. Estas observaciones sugieren que el mejor balance entre latencia y uso de recursos no siempre se logra con una implementación completamente paralela ($R = 1$). Según lo señalado en [24], la herramienta de síntesis presenta diferentes grados de libertad para realizar optimizaciones automáticas en configuraciones con $R = 1$ en comparación con aquellas con $R > 1$, lo que podría ser una de las causas del comportamiento observado en la latencia.

La Figura 4.11 ilustra cómo Vitis HLS planifica la ejecución de las capas en una red neuronal de 3×8 de 32 bits. Cada columna (*Control Step*) representa un ciclo de reloj, mientras que cada fila corresponde a una tarea, como la lectura de datos, las operaciones de cada capa y las funciones

de activación. Los rectángulos verdes indican el momento y duración de cada tarea, y la línea segmentada representa el rango de incertidumbre asociado a la propagación de señales, que será ajustado posteriormente en Vivado durante la etapa de *Place & Route*.

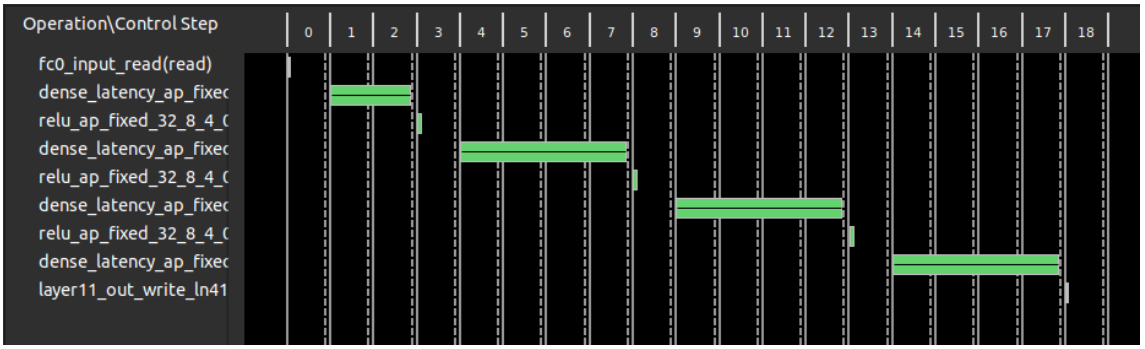
En la Figura 4.11 se comparan los casos R=1, R=2 y R=4. Aunque R=1 utiliza recursos dedicados que permiten una operación completamente paralela, la latencia por capa, salvo en la primera, es mayor que en R=2. Esto ocurre porque que las tareas de cada capa se planifican en ciclos separados en lugar de ejecutarse simultáneamente, lo que limita la eficiencia del hardware. En contraste, en R=2, todas las capas tienen un tiempo de ejecución uniforme de 3 ciclos de reloj. De manera similar, en R=4, las capas requieren cinco ciclos de reloj, con la excepción de la primera.

Además, en la Fig. 4.11 se nota que, a pesar de que la última capa debe calcular 8 multiplicaciones y la primera 32, el tiempo de ejecución de la última capa es mayor. Esto puede atribuirse a las decisiones de planificación de operaciones realizadas automáticamente por Vitis HLS.

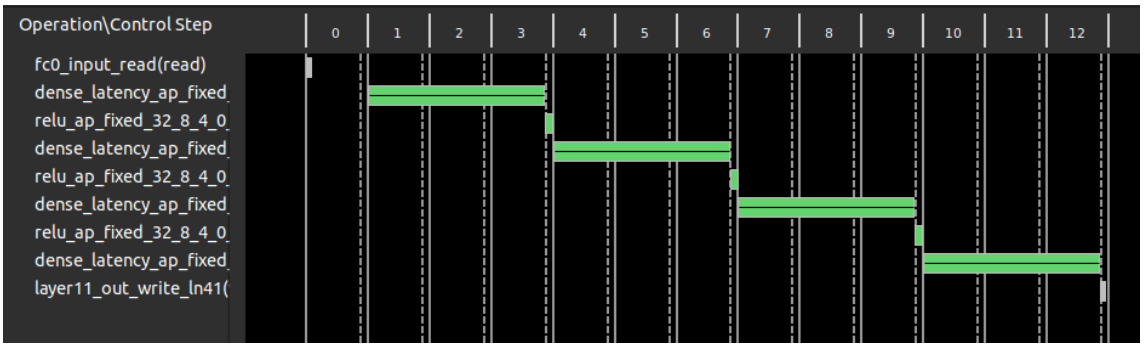
Asimismo, se identifica una variación en la planificación de las funciones de activación. En el caso de R=1, todas las funciones de activación se ejecutan en un ciclo de reloj adicional después de completar las operaciones de la capa correspondiente. En contraste, para R=2, estas funciones se realizan dentro del tiempo restante de la ejecución de las operaciones de su respectiva capa.

La combinación de la optimización en la planificación de las activaciones, y el menor tiempo de ejecución de las capas de la red, permite que la implementación resultante con factor de reutilización dos tenga un menor tiempo de ejecución que la implementación lograda con factor de reutilización uno.

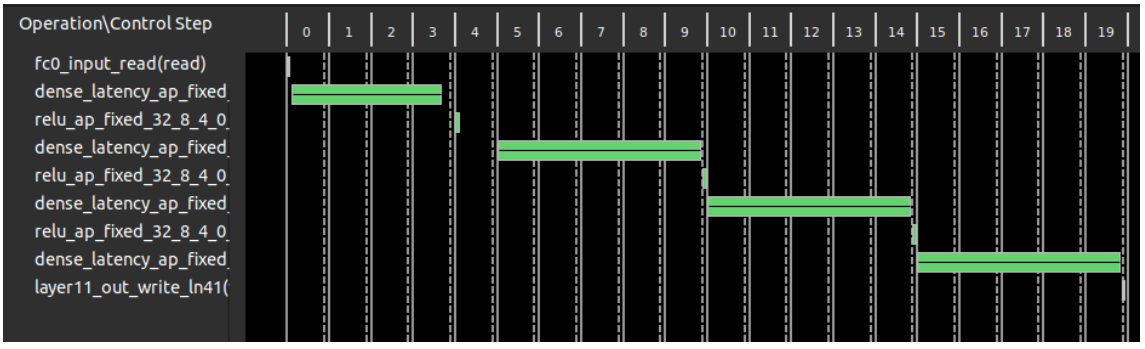
En síntesis, el factor de reutilización permite balancear principalmente el uso de bloques DSP y la latencia de la red neuronal. Para los casos de redes que agotan completamente los recursos disponibles, una adecuada elección del factor de reutilización puede hacer viable su implementación. Por otro lado, aunque la latencia de la red incrementa significativamente con factores de reutilización altos, el factor de reutilización R=2 se identificó repetidamente como el que logra la menor latencia posible para el nivel de cuantización deseado. Además, salvo en los casos que no utilizan bloques DSP, el uso de LUTs, aunque se reduce con factores de reutilización mayores, permanece relativamente constante y no muestra un aumento significativo con valores de reutilización más altos. Finalmente, el uso de FFs sigue generalmente una tendencia de crecimiento con forma logarítmica respecto al factor de reutilización. Sin embargo, debido a la relación entre el uso de FFs y el resto de los recursos, estos no representan un recurso crítico para la implementación de las redes neuronales.



(a) Planificación con factor de reutilización uno.



(b) Planificación con factor de reutilización dos.



(c) Planificación con factor de reutilización cuatro.

Figura 4.11: Planificación de operaciones por capa sintetizadas por Vitis HLS para una red de 3×8 de 32 bits.

4.8 Implementación de redes utilizadas en la literatura

4.8.1. Objetivo del experimento

Para finalizar los experimentos, se consideran dos redes con combinaciones de hiperparámetros reportadas en trabajos relacionados de redes para MPC implementadas en FPGA, de esta manera, es posible comparar el desempeño en términos de latencia de la herramienta contra otras técnicas utilizadas en otros trabajos a través de la misma red neuronal implementada con diferente factor de reutilización.

Además, a partir de la replicación de las implementaciones, se busca identificar alternativas de optimización que no hayan sido consideradas por los autores y que sean posibles utilizando HLS4ML.

Tabla 4.10: Parámetros utilizados para la implementación de redes neuronales presentes en la literatura.

Parámetro	**Valores para red de [19]**	**Valores para red de [21]**
Período	10 [ns]	10 [ns]
Estrategia	*Latency*	*Resources*
W	$\{8, 16, 32\}$	8
Q	$\frac{W}{4}$	2
n_x	3	12
n_u	2	4
L	5	2
M	10	100
R	$[1, 10] \cup \{\left\lceil \frac{100}{x} \right\rceil \mid x \in [1, 9]\}$	{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024}

4.8.2. Condiciones de exploración

La Tabla 4.10 resume las condiciones utilizadas para las implementaciones de las redes consideradas. En particular, para ambas redes se utiliza un período de síntesis de 10 [ns], mientras que las estrategias de HLS4ML se fijan a *Latency* para la red de [19] y *Resources* para la red de [21]. La elección de la estrategia *Resources* busca favorecer la viabilidad de la red de [21], ya que, debido al gran número de multiplicaciones que conlleva, es esperable que agote rápidamente los recursos disponibles, como se observó en las tendencias de la Sección 4.5.

Para la red de [19] se analizaron niveles de cuantización W de 8, 16 y 32 bits. Sin embargo, debido al alto tiempo de síntesis requerido (aproximadamente 7 días por cada implementación con 16 bits), la red de [21] se limitó a una cuantización de 8 bits. En ambos casos, se asignó una cuarta parte de los bits de cuantización a la parte entera Q.

Respecto a las cantidades de entradas y salidas, se respetaron los valores reportados por los autores: 3 entradas y 2 salidas para la red de [19], y 12 entradas y 4 cuatro salidas para la red de [21]. Los hiperparámetros de las redes son 5×10 y 2×100, respectivamente.

Finalmente, los factores de reutilización considerados para la red de [19] se eligieron tomando en cuenta aquellos que generan diferentes cantidades de multiplicadores, mientras que para la red de [21] se limitaron a potencias de dos, desde 1 hasta 1024.

4.8.3. Análisis de resultados

4.8.3.1. Red de 5×10

La primera red a analizar corresponde a la reportada por [19], que posee hiperparámetros $L = 5$ y $M = 10$. En su estudio, los autores reportan dos implementaciones de la misma red: una con punto flotante y otra con 16 bits de cuantización, destacando que esta última ofrece un mejor desempeño en términos de uso de recursos y latencia, con una latencia mínima de 1 [μs]. La Figura 4.12 muestra los reportes de síntesis lógica para esta red utilizando HLS4ML. Las regiones coloreadas en la figura representan comparaciones relativas con los reportes de la literatura: las zonas rojas indican un mayor uso de recursos o latencia en comparación con la implementación en punto flotante, las zonas amarillas corresponden a valores intermedios entre las implementaciones en punto flotante y 16 bits, y las zonas verdes reflejan un uso de recursos o latencia menor que el reporte para 16 bits.

En cuanto a los DSPs, en la Figura 4.12 se muestra que el uso de DSPs es considerablemente mayor que el reporte en punto flotante de la literatura hasta $R = 10$ para 16 bits, y $R = 20$ para 32 bits. Posteriormente, el uso de DSPs sigue siendo mayor que lo reportado por la literatura en 16 bits (10 DSPs), excepto cuando se emplean 8 bits (que no utilizan DSPs) y en el caso de 16 bits con $R = 100$, donde la red requiere 6 DSPs.

Para las LUTs, los niveles de cuantización de 32 y 16 bits no logran reducir el uso de recursos por debajo del reporte de la literatura. Sin embargo, con 8 bits, es posible lograr una reducción en el uso de recursos respecto a la implementación en punto flotante, aunque el valor reportado en [19] para 16 bits sigue siendo inalcanzable.

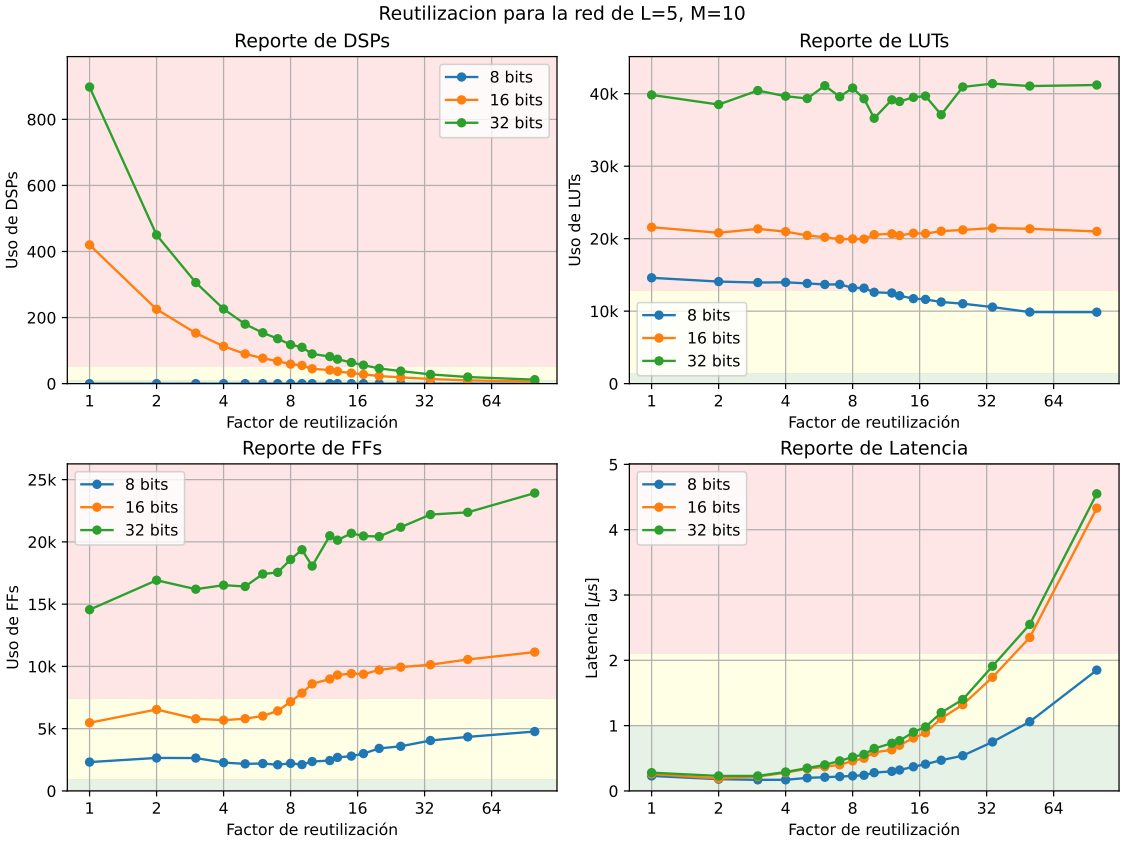


Figura 4.12: Resultados de la exploración de factores de reutilización con diferentes versiones de la red utilizada por Lucia et al. [19].

Un comportamiento similar al de las LUTs ocurre con los FFs: con 8 bits, el uso de recursos se mantiene dentro del rango de los reportes de la literatura. Con 16 bits, el uso de FFs supera al reporte de punto flotante a partir de $R = 9$, y con 32 bits, el uso de FFs es consistentemente mayor que el reporte de punto flotante.

Aunque las implementaciones logradas con HLS4ML son menos eficientes en términos de uso de recursos que las reportadas en la literatura, la latencia alcanzada es significativamente menor, logrando una latencia mínima de 170 [ns] con 8 bits y 190 [ns] con 16 bits.

4.8.3.2. Red de 2×100

La segunda implementación considerada corresponde a la red presentada en [21], caracterizada por su alta exigencia computacional, con hiperparámetros $L = 2$ y $M = 100$, lo que requiere 11.600 operaciones de multiplicación. Los resultados reportados para esta red incluyen una latencia 126 [μs].

La Figura 4.13 presenta los reportes de las implementaciones realizadas con HLS4ML para la red de [21]. Las regiones coloreadas en la figura reflejan comparaciones relativas con los resultados de la literatura: las zonas en rojo indican un mayor uso de recursos o latencia, mientras que las regiones en verde indican que el uso de recursos o la latencia es menor que lo reportado en la literatura.

Aunque con 8 bits no se utilizaron bloques DSP, lo que implica un ahorro significativo de recursos, los reportes de LUTs y FFs muestran que no se alcanzó un uso comparable al reportado en la literatura en ninguno de los casos analizados. Sin embargo, la latencia lograda con HLS4ML fue considerablemente menor en todos los casos estudiados, alcanzando un mínimo de 430 [ns] y un máximo de 2.6 [μs].

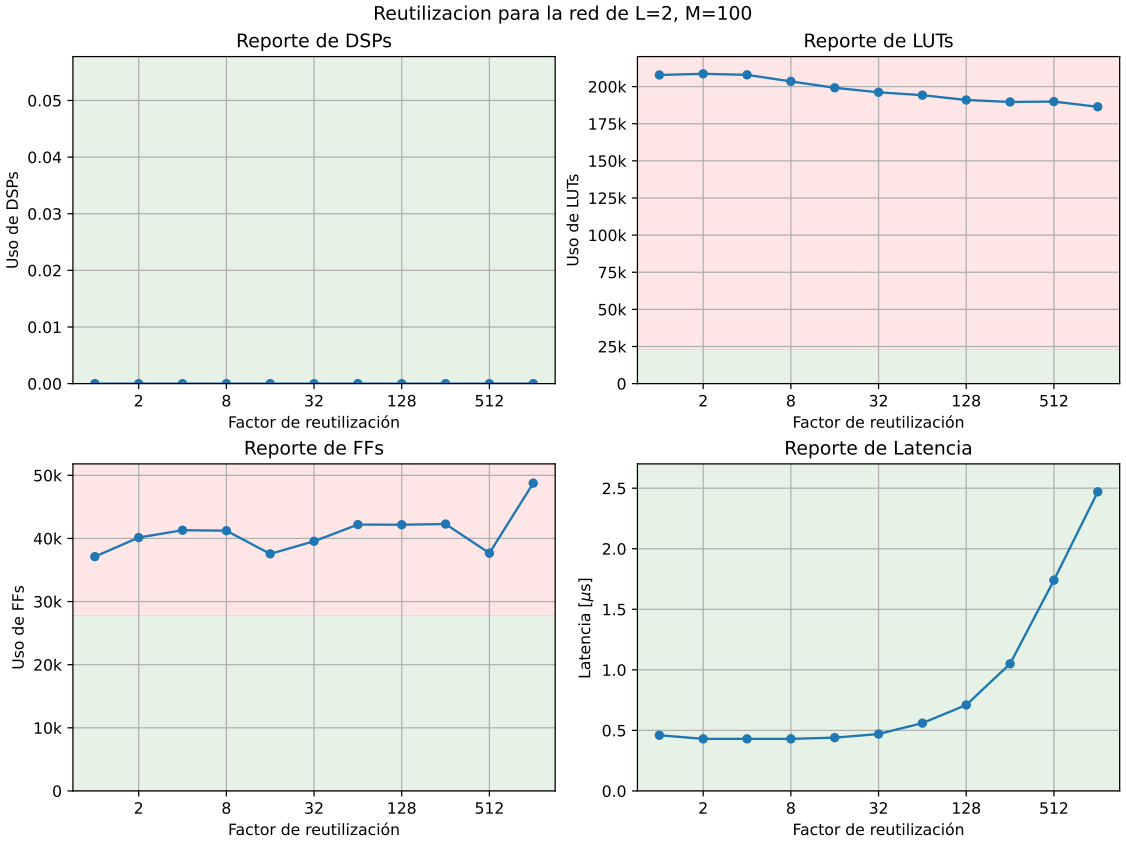


Figura 4.13: Resultados de la exploración de factores de reutilización de la red de 8 bits similar a la utilizada por Dong et al. [21].

4.8.4. Observaciones generales

A partir de los resultados obtenidos, se concluye que existe una discrepancia entre las arquitecturas de hardware utilizadas con HLS4ML y las empleadas por los autores en las implementaciones revisadas. Esto se refleja en un claro *trade-off* entre el uso de recursos y la latencia. En particular, HLS4ML permite alcanzar desempeños significativamente mejores en términos de latencia en comparación con los enfoques reportados en la literatura. Sin embargo, el uso de recursos con HLS4ML fue mayor en ambas redes estudiadas.

Por otro lado, aunque el desempeño se analizó considerando diferentes niveles de cuantización, es importante tener en cuenta el impacto del nivel de precisión del modelo, que no ha sido considerado en este estudio. Si bien las implementaciones con menor nivel de cuantización permiten un mejor compromiso entre uso de recursos y latencia, su funcionalidad queda condicionada por los requisitos específicos de la aplicación de MPC.

No obstante, estos resultados destacan el potencial de las técnicas exploradas para ofrecer alternativas viables a desafíos relevantes en la comunidad. Sin embargo, para comprender mejor los reportes y el tipo de *trade-off* adquirido, es fundamental considerar el tipo de arquitectura de hardware empleado. Esto permite identificar con mayor claridad las implicaciones de diseño en términos de uso de recursos y latencia, así como las limitaciones específicas asociadas a cada enfoque.

Capítulo 5

Casos de estudio

Este capítulo presenta dos casos de estudio de lazos de control reportados en literatura previa, en los cuales se aplica el flujo descrito en el Capítulo 3 y las directrices de diseño extraídas de las exploraciones realizadas en el Capítulo 4. El objetivo es facilitar el balance entre los *trade-offs* de diseño involucrados durante el proceso, considerando tanto las características del hardware utilizado como los requisitos de la aplicación. Específicamente, los casos de estudio considerados incluyen:

1. Un controlador diseñado para un motor DC con una carga inercial, cuyo modelo se presenta en [64]. El modelo corresponde a un sistema relativamente simple, que opera con un período de muestreo de 10 [ms].

2. Un controlador diseñado para un sistema de recursos energéticos distribuidos (DER, por sus siglas en inglés), cuyo modelo se presenta en [45] y que requiere operar con un período de muestreo de 200 [μs].

Para cada uno de los casos de estudio, se detallan los resultados obtenidos en las distintas etapas del flujo, descrito en la Figura 3.1, desde el entrenamiento con precisión de punto flotante, hasta la validación del hardware en lazo cerrado. Las simulaciones del flujo se comparan con el controlador MPC implementado en Matlab y usado como referencia, mostrando el comportamiento de los estados y las actuaciones a lo largo del proceso, con el fin de evaluar el desempeño del controlador diseñado desde el punto de vista de control.

5.1 Caso de estudio: Motor DC

5.1.1. Planteamiento MPC

El primer caso de estudio considerado para validar la aplicabilidad del flujo descrito en el Capítulo 3, corresponde a un sistema compuesto por un motor de corriente continua (DC) con carga inercial presentado en [64]. Este sistema se utiliza debido a que es un ejemplo bien documentado y sencillo de una aplicación de MPC, lo que permite ilustrar de manera clara el flujo propuesto. La salida del sistema corresponde a la velocidad angular de la carga, mientras que la entrada es el voltaje aplicado al motor. El sistema opera con un período de muestreo de 10 [ms] y, siguiendo la notación presentada en el Capítulo 2, está definido en tiempo discreto por las siguientes matrices:

$$\boldsymbol{A} = \begin{bmatrix} 0.9662 & -0.0339 \\ 0.0509 & 0.7398 \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} 0.6554 \\ 0.0179 \end{bmatrix}, \qquad \boldsymbol{C} = \begin{bmatrix} 0 & 1 \end{bmatrix}. \tag{5.1}$$

Las matrices de costos del problema se definen como $\boldsymbol{Q} = \boldsymbol{C}^\intercal \boldsymbol{C}$ para los estados y $\boldsymbol{R} = [0.1]$ para la actuación. El costo del estado estacionario, representado por $\boldsymbol{P}$, se obtiene como la solución

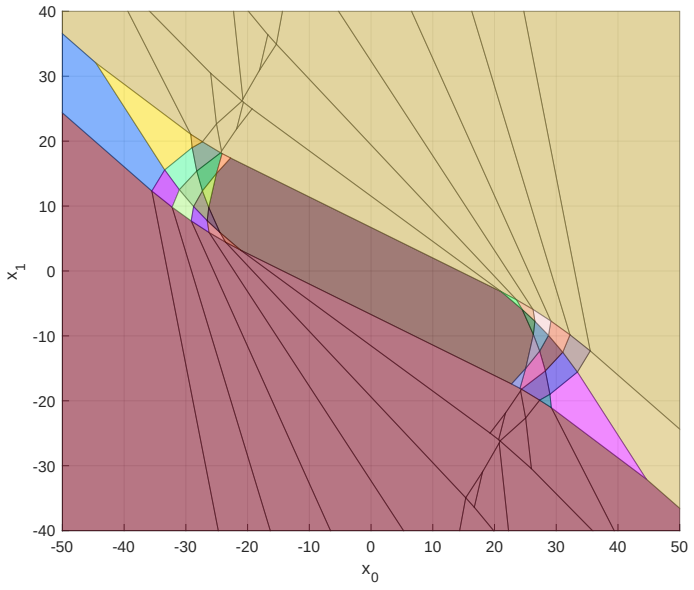


Figura 5.1: Regiones de la PWA de MPC explícito para el motor DC.

de la ecuación de Riccati. Siguiendo lo documentado en [64], se considera un horizonte de predicción de $N = 7$. Las restricciones en los estados y las acciones de control se establecen como:

$$\mathcal{X} = \left\{ \boldsymbol{x} \in \mathbb{R}^2 : \begin{bmatrix} -50 \\ -40 \end{bmatrix} \leq \boldsymbol{x} \leq \begin{bmatrix} -50 \\ -40 \end{bmatrix} \right\}, \tag{5.2}$$

$$\mathcal{U} = \{ \boldsymbol{u} \in \mathbb{R} : -5 \leq \boldsymbol{u} \leq 5 \}. \tag{5.3}$$

donde $\boldsymbol{x} = [x_0 \quad x_1]^\intercal$ son los estados del sistema, y $\boldsymbol{u} = [u_0]$ es la actuación. Las restricciones en la actuación son las mismas reportadas en [64]. Sin embargo, los límites en los estados han sido adaptados en este trabajo con el propósito de mantener la simplicidad del ejemplo.

Para generar los datos de entrenamiento de las redes neuronales, es necesario disponer de un controlador de referencia, que en este caso corresponde a un controlador explícito obtenido mediante la herramienta Hybrid Toolbox [65] en Matlab. La Figura 5.1 muestra la partición del espacio de estados de la ley de control en 67 regiones, donde los colores repetidos agrupan regiones con ganancias similares. A partir de esta ley de control, y siguiendo lo expuesto en el Capítulo 3, se genera un conjunto de datos para el entrenamiento de las redes neuronales, que se detalla en la siguiente sección.

5.1.2. Diseño de red neuronal cuantizada

Esta sección presenta los resultados del entrenamiento de redes neuronales con precisión de punto flotante, y redes cuantizadas, junto con sus respectivas simulaciones en lazo cerrado, como se ilustra en la Figura 3.1, que esquematiza el flujo de diseño. Para mayor detalle sobre los scripts utilizados para obtener los resultados presentados, ver el repositorio asociado a este documento.

5.1.2.1. Entrenamiento con precisión flotante

Para determinar una combinación de hiperparámetros adecuada para la red neuronal, se exploran combinaciones dentro de los rangos $L \in [1, 5]$ para la cantidad de capas ocultas y $M \in [2, 15]$ para la cantidad de neuronas por capa. Estos valores fueron seleccionados a partir de la aproximación de la expresión (2.33) que sugiere que una red con cuatro capas y tres neuronas por capa puede aprender la ley de control. Sin embargo, el rango final se determinó de manera empírica tras un proceso iterativo de ajuste.

La Figura 5.2 muestra los resultados de la exploración de combinaciones de hiperparámetros de red, donde se observa que la red de cuatro capas y tres neuronas por capa sugerida por la

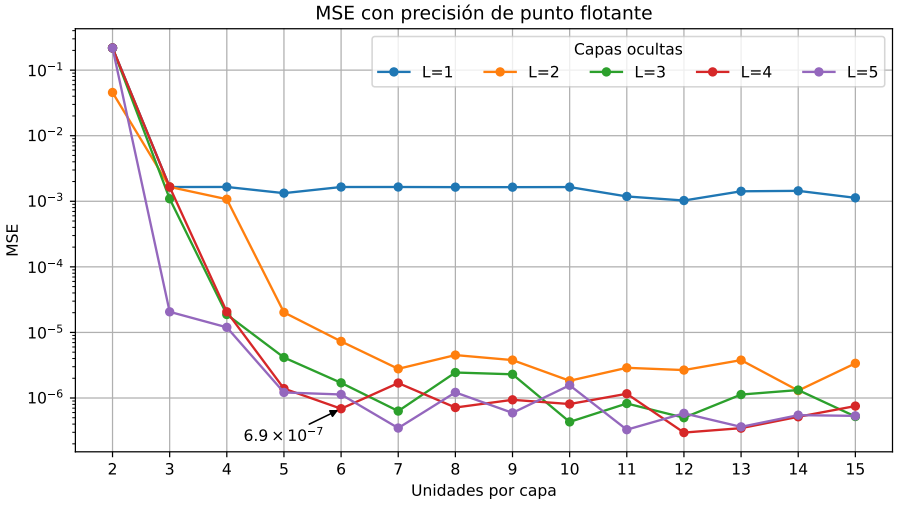


Figura 5.2: Resultados de la exploración de arquitecturas de red neuronal.

expresión (2.33) alcanza un MSE cercano a 10^{-3}, el cual se reduce considerablemente al incrementar la cantidad de capas y neuronas por capa, subrayando la importancia de la exploración de hiperparámetros.

Como era esperable, en la Figura 5.2 se observa que un mayor número de neuronas por capa mejora la capacidad de la red para aproximar la ley de control. Además, los resultados indican que una red con una sola capa oculta presenta limitaciones en su capacidad de aprendizaje en comparación con configuraciones más profundas.

La Figura 5.2 también evidencia que las redes más profundas requieren menos neuronas por capa para alcanzar una misma precisión. Por ejemplo, para un MSE de 10^{-5}, una red con dos capas ocultas necesita seis neuronas por capa, mientras que una red con tres capas ocultas alcanza un MSE menor con solo cinco neuronas por capa.

A partir de los valores observados en la Figura 5.2, se selecciona una red con cuatro capas ocultas y seis neuronas por capa (4×6) como un compromiso adecuado entre compacidad y precisión para continuar con el flujo de diseño. Esta configuración alcanza un MSE de $6.9 \cdot 10^{-7}$. Aunque la configuración con cinco capas ocultas y siete neuronas por capa logra un MSE menor ($3.5 \cdot 10^{-7}$), también implica un aumento en la cantidad de capas ocultas. De acuerdo con las directrices del Capítulo 4, esto se traduce en una mayor latencia de inferencia. Además, este incremento en el número de neuronas de la red conlleva un aumento en la cantidad de operaciones de multiplicación, lo que, según lo descrito en la Sección 4.4, tiende a incrementar el uso de recursos en la implementación en hardware.

Los resultados de la inferencia de la red seleccionada para la cuantización se presentan en la Figura 5.3. En ella, cada punto azul representa una actuación inferida en función del valor esperado obtenido a partir del controlador explícito para el mismo estado. La línea segmentada gris indica el origen de la actuación, cuyo rango está normalizado. La recta roja representa la relación ideal entre la predicción de la red y el valor esperado. De esta manera, la precisión de la red puede evaluarse observando la distancia de los puntos a la recta roja. Se destaca la baja dispersión de los puntos con respecto a la recta roja y la ausencia de valores fuera del rango $[0, 1]$ en el eje de las ordenadas, lo que indica que la red cumple con las restricciones de actuación en todo el conjunto de datos de prueba.

La Figura 5.4 muestra la evolución temporal de los estados y la actuación en simulaciones realizadas con la red neuronal y el controlador de referencia, ambos partiendo del mismo estado inicial. El comportamiento de los estados y la actuación es prácticamente idéntico en ambas simulaciones. Además, en cada gráfica se incluye el error cuadrático de la red con respecto al controlador de referencia (Eje derecho, en rojo). Se observa un error constante en estado estacionario en los estados, cuya magnitud no es significativa, puesto que corresponde a un error esperable en los controladores basados en redes neuronales [28] y no influye en la capacidad de la red para llevar

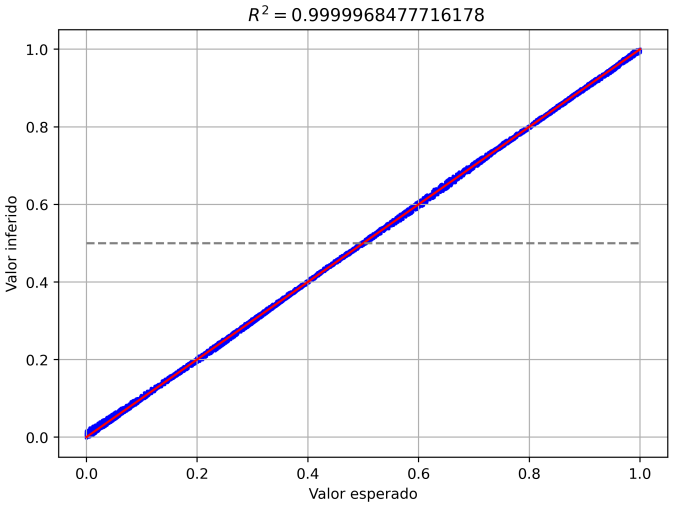


Figura 5.3: Relación entre valor de predicción y valor esperado para la red de 4 capas y 6 neuronas de precisión flotante.

los estados del sistema al origen. Además, en la Figura 5.4b se aprecia que el mayor error en la actuación ocurre durante la fase transitoria de la simulación, donde la variabilidad de la actuación es mayor.

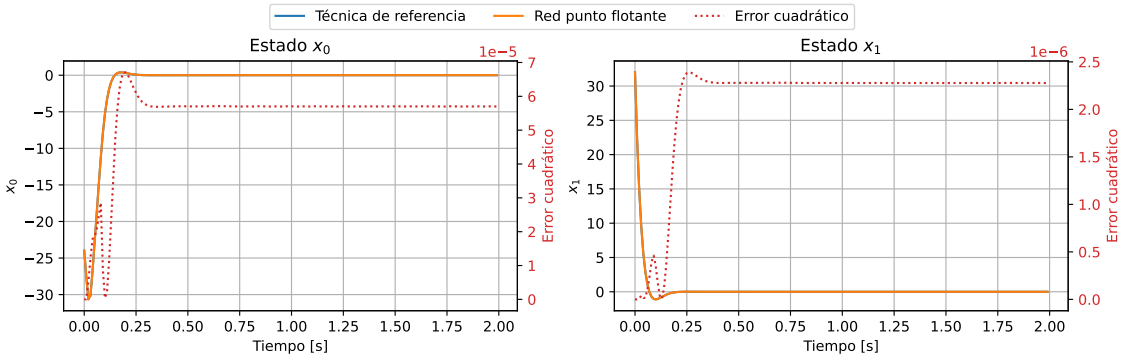
Dado el bajo error de aproximación de la red y la similitud de su comportamiento en simulación con respecto al controlador de referencia, se seleccionan los hiperparámetros $L = 4$ y $M = 6$, junto con los pesos de la red entrenada, para su uso en el entrenamiento cuantizado.

5.1.2.2. Entrenamiento cuantizado

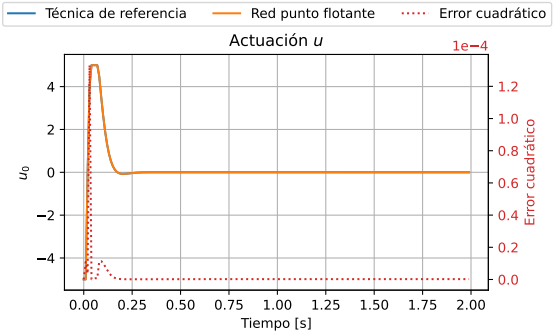
Una vez definida la combinación de hiperparámetros de la red, el siguiente paso es determinar un nivel de cuantización adecuado para su implementación en FPGA. Para ello, se realiza una exploración de alternativas considerando la cantidad total de bits utilizados en la cuantización, definida como $W \in [2, 20]$, y la cantidad de bits asignada a la parte entera y el signo, definida como $Q \in [1, 4]$. El objetivo de esta exploración es identificar la menor cantidad de bits necesarios para mantener la funcionalidad alcanzada por la red con precisión de punto flotante. Cabe mencionar, que tanto los pesos como las entradas de la red están normalizados, por lo que se espera que el rango de bits para la parte entera sea apenas utilizado. En base a experimentos preliminares, se limita el rango hasta 3 bits (Q=4), dado que en ellos se notó que el uso de una cantidad mayor de bits no afecta el error de aproximación de la red.

Los resultados de la exploración de niveles de cuantización se presentan en la Figura 5.5. Cabe mencionar, que el menor nivel de cuantización admisible para cada curva presentada, corresponde a la cantidad de bits para la parte entera más el signo, lo que explica los rangos diferenciados para cada una de las curvas. Como es esperable, a medida que aumenta la cantidad de bits utilizados para la cuantización, el error de aproximación disminuye. En la misma figura, la línea segmentada indica el MSE obtenido con la red de punto flotante, por lo que se espera que los niveles de cuantización con mayor precisión converjan a este valor. En este sentido, la figura muestra que desde los 14 bits, el error se estabiliza cerca del de la red de punto flotante. Incluso, se observan algunos casos donde la red cuantizada logra un menor error, posiblemente debido a que el ruido de cuantización afecta la optimización que se realiza durante el entrenamiento, facilitando el hallazgo de mínimos locales más favorables.

Como se observó en el Capítulo 4, el uso de recursos aumenta con el nivel de cuantización, por lo que se selecciona el menor nivel de cuantización necesario para lograr un error de aproximación comparable al de la red de punto flotante, la cual ya ha demostrado un desempeño adecuado en simulación. Por este motivo, se selecciona el nivel de cuantización con 10 bits para la parte fraccionaria y 1 bit para la parte entera, correspondientes a los parámetros $W = 12$ y $Q = 2$.



(a) Comportamiento de estados.



(b) Comportamiento de actuación.

Figura 5.4: Resultados de simulación de la red neuronal de punto flotante.

La red cuantizada seleccionada alcanza un MSE de $1.1 \cdot 10^{-6}$, utilizando menos de la mitad de los bits de la red en punto flotante (32 bits) y logrando una aproximación de la ley de control con un error cuadrático similar. Debido a esta similitud entre los errores, es esperable que la red cuantizada sea adecuada para la implementación. Sin embargo, el impacto del error en la aplicación se determinará en la simulación. La Figura 5.6 muestra el ajuste de la actuación de la red cuantizada para estados no presentes en el conjunto de entrenamiento. En comparación con la Figura 5.3, que muestra el ajuste de la red en punto flotante, en la red cuantizada se observa una dispersión ligeramente mayor de las actuaciones inferidas. Esto sugiere que un mayor número de estados genera actuaciones que difieren de las esperadas, lo que podría traducirse en una mayor diferencia en el comportamiento del controlador en simulación. Esta diferencia de precisión entre la red cuantizada y la versión sin cuantizar también se refleja en el coeficiente de determinación (R^2), el cual, si bien sigue siendo cercano a la unidad, presenta una diferencia en el sexto dígito decimal. Esta diferencia indica que, términos prácticos, ambas redes son iguales.

Los resultados de la simulación de la red cuantizada seleccionada se presentan en la Figura 5.7. Aunque visualmente el comportamiento de la red cuantizada es indistinguible del controlador de referencia, en comparación con la simulación de la red en punto flotante, se observa un incremento en la magnitud de los errores cuadráticos en los estados y la actuación. La Figura 5.7a muestra que, en estado estacionario, los errores en los estados no convergen a un valor fijo, sino que oscilan dentro de un rango acotado. Este comportamiento es esperable a aplicar cuantización y no introduce diferencias significativas entre la red cuantizada y el controlador de referencia. Por otro lado, la Figura 5.7b evidencia una mayor cantidad de *peaks* de error en la actuación durante el transitorio. En estado estacionario, se observa que la oscilación en los estados es causada por pequeños *peaks* de actuación de la red cuantizada en respuesta a desviaciones de los estados respecto del origen.

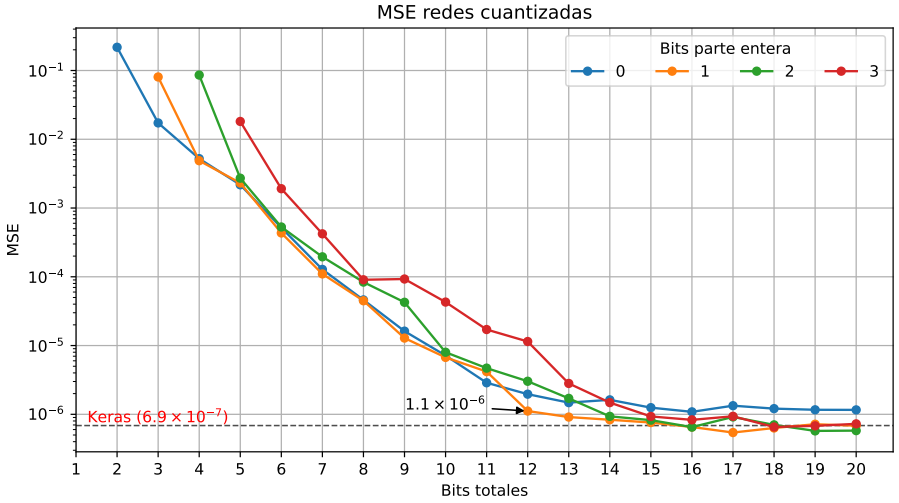


Figura 5.5: Resultados de la exploración de cuantizaciones para la red de 4 capas ocultas y 6 unidades por capa.

Tabla 5.1: Estimación de uso de bloques DSP y resultados de síntesis lógica para cada factor de reutilización.

R	**Latencia** [ns]	**DSPs** estimados	**DSPs** [-] (%)	**LUTs** [-] (%)	**FFs** [-] (%)
1	250	126	54 (24.5 %)	2489 (4.7 %)	1979 (1.9 %)
2	200	63	33 (15.0 %)	2624 (4.9 %)	1771 (1.7 %)
3	240	42	22 (10.0 %)	2712 (5.1 %)	1835 (1.7 %)
4	260	32	18 (8.2 %)	2664 (5.0 %)	1876 (1.8 %)
5	260	29	18 (8.2 %)	2648 (5.0 %)	1873 (1.8 %)
6	310	21	12 (5.5 %)	2714 (5.1 %)	1831 (1.7 %)
8	300	18	12 (5.5 %)	2728 (5.1 %)	1874 (1.8 %)
9	340	15	10 (4.5 %)	2704 (5.1 %)	1875 (1.8 %)
12	390	11	8 (3.6 %)	2666 (5.0 %)	1891 (1.8 %)
18	520	8	6 (2.7 %)	2677 (5.0 %)	1909 (1.8 %)
36	600	5	5 (2.3 %)	2696 (5.1 %)	1917 (1.8 %)

5.1.3. Diseño de hardware

Esta sección abarca el proceso de diseño de hardware, desde la conversión a HLS hasta el proceso de síntesis e implementación, ilustrados en la Figura 3.1.

La implementación en hardware se lleva a cabo en la tarjeta de desarrollo PYNQ-Z1 de Digilent, utilizando un reloj de 100 [MHz], que corresponde a la frecuencia de operación por defecto de la plataforma. La estrategia de HLS4ML usada es *Latency*, por lo que, como se mencionó en el Capítulo 4, las implementaciones no utilizarán BRAMs.

El modelo de red neuronal seleccionada requiere 126 multiplicaciones según la ecuación (3.1). Este cálculo se obtiene considerando que la primera capa realiza $2 \cdot 6$ multiplicaciones, las tres capas siguientes requieren de 6^2 cada una, y la capa de salida suma 6 multiplicaciones adicionales. Dado que el nivel de cuantización utilizado es de 12 bits, como se concluyó en el Capítulo 4.6, cada multiplicación puede ejecutarse en un único bloque DSP, por lo que el uso máximo estimado de estos recursos asciende a 126 bloques DSP.

Dado que cada capa de la red neuronal contiene hasta M^2 multiplicaciones, se evalúan factores de reutilización en el rango de 1 a 36. La Figura 5.8a ilustra el uso de recursos en la exploración de factores de reutilización R, según los resultados de síntesis lógica en Vitis HLS para la red cuantizada. En particular, en las arquitecturas con una reutilización más alta, se observa una marcada reducción en la cantidad de DSPs requeridos para la implementación de la red, mientras

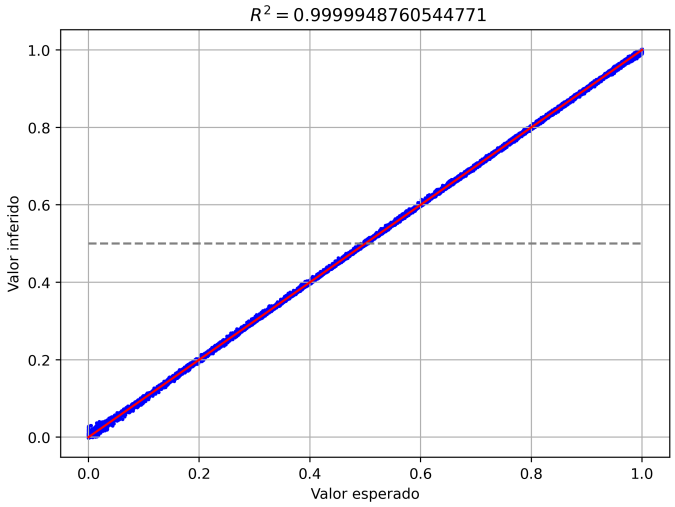


Figura 5.6: Relación entre valor de predicción y valor esperado para la red de 4 capas y 6 neuronas cuantizada con 12 bits.

que los FFs y LUTs requeridos permanecen visiblemente constantes.

Además del uso de recursos, el factor de reutilización tiene una incidencia importante en la latencia. En este sentido, la Figura 5.8b presenta la latencia de las arquitecturas implementadas, donde se observa que un mayor factor de reutilización incrementa la latencia de la red neuronal. Al igual que en los experimentos del Capítulo 4.7, la menor latencia se obtiene con un factor de reutilización igual a 2, en lugar del caso que representa paralelismo completo. Por otro lado, incluso implementando la arquitectura con el factor de reutilización máximo considerado en este estudio, la latencia se mantiene en el orden de los nanosegundos, lo que resulta promisorio para aplicaciones de MPC con restricciones de tiempo. Para este caso de estudio, la restricción de tiempo de ejecución es de 10 [ms], que es cinco órdenes de magnitud superior a la latencia de la red con la arquitectura más lenta implementada en esta exploración (600 [ns]). Sin embargo, es importante considerar que, en un lazo de control, además del cálculo de la actuación, se debe dedicar tiempo a la comunicación con sensores y actuadores, así como a otras tareas involucradas en el algoritmo de control, como saturaciones y estimaciones.

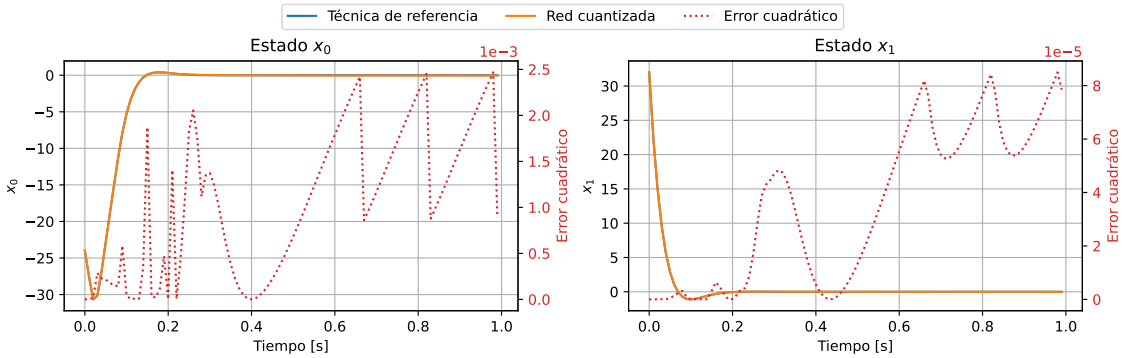
La Tabla 5.1 detalla los datos mostrados en la Figura 5.8, incluyendo además una columna de estimación del uso de DSPs basada en el número de multiplicaciones de la red (Ver Anexo B para más detalles). A través de la tabla es posible distinguir las variaciones menores con el factor de reutilización, destacando un ligero incremento en las LUTs desde $R = 2$ y una leve tendencia al incremento en los FFs, salvo en $R = 1$, donde se registra el mayor uso de FFs entre las arquitecturas evaluadas.

Con base en los datos presentados, se selecciona la arquitectura con un factor de reutilización dos ($R = 2$) para la implementación y validación en hardware, ya que esta arquitectura tiene la menor latencia dentro de la evaluación y no agota los recursos disponibles en la plataforma.

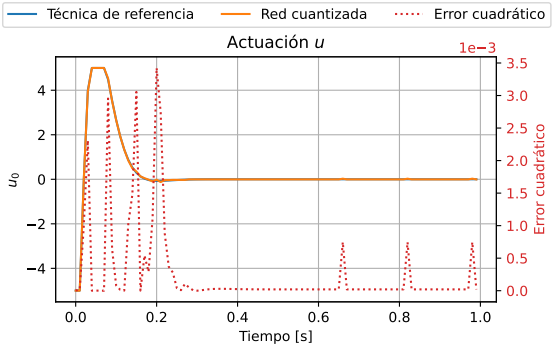
5.1.4. Validación de hardware

Para completar el diseño del controlador en FPGA, es necesario diseñar una plataforma de software que permita acceder al hardware diseñado, y así evaluar su desempeño en conjunto con una planta simulada. Este análisis permite comparar el comportamiento del hardware implementado con el controlador MPC de referencia y determinar si el desempeño de control logrado por la implementación es adecuado para la aplicación.

Para esta aplicación, se incluye además una medición preliminar del tiempo efectivo requerido para obtener una acción de control desde la plataforma de software. Se consideran dos enfoques: un software *Bare Metal* escrito en C++ y otro que utiliza las bibliotecas de PYNQ sobre Linux, escrito en Python. El software utilizado está disponible en el repositorio asociado a este documento.



(a) Comportamiento de estados.



(b) Comportamiento de actuación.

Figura 5.7: Resultados de simulación de la red neuronal cuantizada.

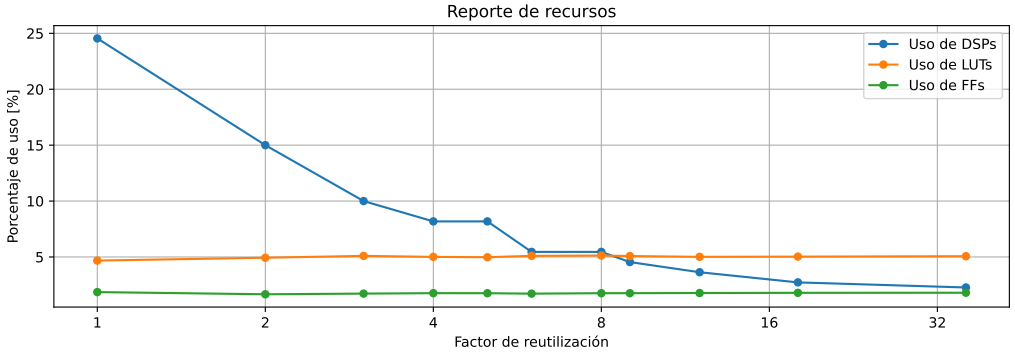
5.1.4.1. Desempeño de control

La Figura 5.9 muestra el comportamiento de la red implementada en FPGA conectada a una planta simulada, en comparación con el controlador de referencia. Al igual que en las simulaciones previas, los comportamientos de los estados y la actuación no presentan diferencias visualmente apreciables.

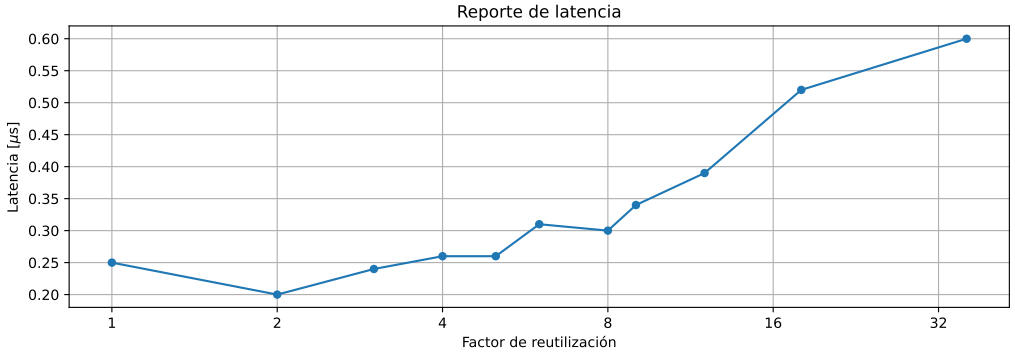
Las curvas de error presentadas en la Figura 5.9 (en rojo) tienen el mismo orden de magnitud que las obtenidas para la red cuantizada en software (Figura 5.7), aunque con máximos ligeramente menores en el caso de los estados. Además, a diferencia de la simulación de la red cuantizada en software, donde se observan oscilaciones, la Figura 5.9a muestra que en la implementación en hardware el error de los estados converge a cero. En comparación con las actuaciones de la red cuantizada en software, en la Figura 5.9b se observa que el error de actuación del hardware tiene valor máximo mayor, pero con un pico menos durante el transitorio. Además, durante el estado estacionario, la actuación no presenta los mismos picos periódicos de la red cuantizada en software.

La Tabla 5.2 presenta un sumario de indicadores cuantitativos sobre las simulaciones realizadas para las redes de punto flotante, cuantizada y la implementación en hardware. En la tabla, se reporta el MSE calculado para cada estado y la actuación a lo largo del tiempo de simulación. Se observa que, en todas las señales, el MSE de la red en hardware es menor que el de la red cuantizada. Aunque la red cuantizada en Python utiliza pesos cuantizados, las operaciones siguen realizándose en punto flotante, lo que puede ser la causa de las diferencias en el comportamiento con respecto del hardware. Además, es posible que la acumulación de errores en la realimentación del lazo haya amplificado la diferencia observada. A pesar de ello, estas diferencias son esperables y no afectan la capacidad del controlador para llevar los estados al origen, por lo que son aceptables para esta aplicación.

Además, la Tabla 5.2 incluye el tiempo de asentamiento, definido como el tiempo en que la señal se estabiliza dentro del 1.8 % de su valor inicial. Esta métrica se ha usado de manera relativa para comparar la capacidad de cada controlador para estabilizarse. La tabla, muestra que la cuantización



(a) Reportes de uso de recursos.



(b) Reportes de latencia.

Figura 5.8: Uso de recursos y latencia reportados por Vitis HLS según el factor de reutilización para la red cuantizada a 12 bits de 4 capas ocultas y 6 neuronas por capa.

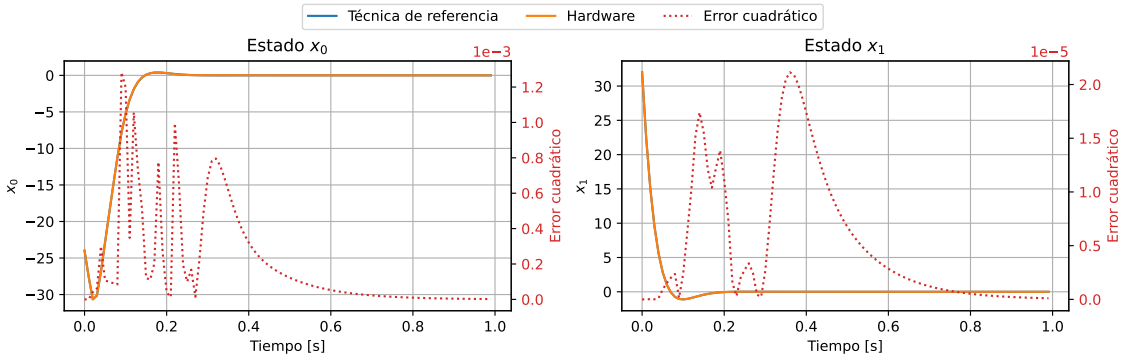
de la red neuronal incrementa el tiempo de asentamiento de la actuación, lo que se relaciona con un pico de error cuadrático en 0.22 [s], mostrado en las Figuras 5.7b y 5.9b.

En estado estacionario, se observa en la Tabla 5.2 que el MSE de los estados de la red en hardware es menor que en la red cuantizada, lo que se explica por la ausencia de la oscilación de los estados en el hardware. Por otro lado, en todas las redes, el MSE de la actuación en estado estacionario es menor que el MSE de toda la simulación, lo que indica una mayor similitud entre las redes y el controlador de referencia en estado estacionario en comparación con el transitorio.

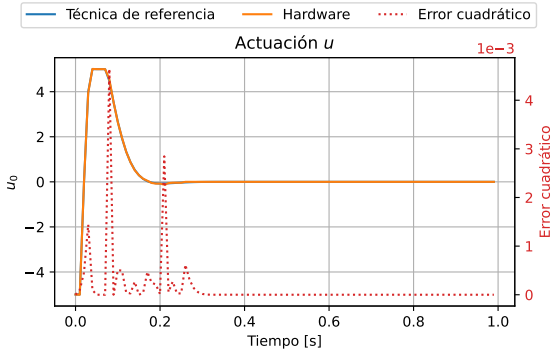
A pesar de las diferencias observadas, el costo de control promedio mostrado en la Tabla 5.2 indica que los desempeños de las redes simuladas y, en particular, de la red implementada en FPGA, son idénticos en términos de control.

Las restricciones de actuación, descritas en la expresión (5.3) no se cumplen ni en la red cuantizada ni en el hardware, como indica la Tabla 5.2. Aunque el incumplimiento de las restricciones es esperable al emplear redes para MPC [28], es importante evaluar su magnitud para determinar si se mantiene dentro de un rango aceptable y establecer estrategias de mitigación, en caso de ser necesario. En este sentido, la tabla muestra que al usar una red neuronal con representación cuantizada se producen violaciones a las restricciones de actuación. Luego, al implementarse en hardware, la red neuronal excede ambas restricciones en ± 0.01 [V]. Sin embargo, en esta aplicación específica, donde el sistema controlado es un motor DC, un desvío de 10 [mV] no es significativo para su desempeño, por lo que no se espera que este exceso afecte negativamente al sistema.

En síntesis, las métricas presentadas confirman que, a pesar de las diferencias con el controlador de referencia, la implementación de la red en FPGA logra un desempeño adecuado para la aplicación. La red consigue llevar los estados al origen y, aunque las restricciones se sobrepasan ligeramente, permanecen dentro de un rango aceptable para la aplicación. Estos resultados indican que el flujo realizado permite implementar de manera viable una red neuronal en FPGA para su uso en MPC, aplicando las directrices extraídas en el Capítulo 4. Además, las simulaciones mostraron que el



(a) Comportamiento de estados.



(b) Comportamiento de actuación.

Figura 5.9: Resultados de simulación con la red neuronal en FPGA.

desempeño de control se mantuvo consistente a lo largo del flujo, evidenciando la efectividad del enfoque para la implementación de controladores para MPC.

5.1.4.2. Análisis comparativo de tiempos de comunicación entre plataformas de software

En la práctica, la latencia del hardware no es el único factor relevante al implementar un controlador en un sistema. El tiempo de muestreo debe ser lo suficientemente amplio para considerar los tiempos de comunicación, medición y actuación. Dado que habitualmente una CPU se encarga del muestreo de los sensores y de la transmisión de la acción de control, se realiza una medición preliminar del tiempo de ejecución de la red neuronal implementada cuando se invoca desde la CPU integrada en la tarjeta de desarrollo utilizada. Para ello, se evalúan dos plataformas de software, con el fin de caracterizar el impacto del entorno de ejecución sobre los tiempos de inferencia del controlador:

- Implementación de software en *Bare Metal* usando C++.
- Implementación de software basada en las bibliotecas de PYNQ sobre Linux en Python.

Para comparar las plataformas de software, se utiliza el hardware representado en la Figura 5.10. El diagrama muestra la interconexión entre la red neuronal (`L4M6W12Q2E_xc7020_10r2`) y la CPU (`ZYNQ7 Processing System`) a través de un bus AXI, así como la señal de interrupción que la red envía a la CPU. Las mediciones se realizan con el módulo `AXI_transaction_timer`, diseñado específicamente para este propósito y disponible en este repositorio [1].

El tiempo de ejecución se mide a través del bus AXI que conecta la CPU con la red neuronal, contabilizando desde la escritura de la primera entrada en la red hasta la lectura de la salida.

[1] Disponible en: `https://github.com/JuanjoV/axi_transaction_timer`

Tabla 5.2: Métricas obtenidas durante las simulaciones realizadas para el Motor DC.

Métrica	Señal	Controlador de referencia	Red en punto flotante	Red cuantizada	Red en hardware
MSE	x_0	0	5.12e-05	1.05e-03	2.00e-04
	x_1	0	1.89e-06	3.54e-05	5.10e-06
	u_0	0	2.25e-06	2.60e-04	1.34e-04
Tiempo de asentamiento [ms]	x_0	140	140	140	140
	x_1	140	140	150	140
	u_0	170	170	220	220
MSE estacionario	x_0	0	5.75e-05	1.20e-03	1.70e-04
	x_1	0	2.18e-06	4.16e-05	5.45e-06
	u_0	0	2.17e-07	6.87e-05	1.94e-05
Costo de control promedio	-	19.105	19.106	19.105	19.105
Actuación máxima	u_0 [V]	5.0000	4.9996	5.0013	5.0098
Actuación mínima	u_0 [V]	-5.0000	-4.9998	-5.0041	-5.0098

Tabla 5.3: Resumen de evaluaciones de tiempo para el motor DC.

Métrica	Bare Metal	PYNQ
Promedio	1.90 [μs]	208.53 [μs]
Máximo	3.14 [μs]	32,512.21 [μs]
Mediana	1.91 [μs]	110.52 [μs]
Desviación estándar	0.04 [μs]	1121.00 [μs]
Percentil 99	1.99 [μs]	239.20 [μs]

Específicamente, considerando la terminología técnica de AXI [66], esto corresponde al intervalo comprendido entre:

1. Cuando en el canal de dirección de escritura se encuentra la dirección de memoria asociada a la primera entrada del IP, y las señales `VALID` y `READY` de ese canal están en alto.

2. En el canal de dirección de lectura se encuentra la dirección de memoria asociada a la salida del IP, y las señales `VALID` y `READY` de ese canal están en alto.

Es importante destacar que, para asegurar la validez de la medición, la primera dirección de memoria escrita es siempre la misma. De esta manera, el IP `AXI_transaction_timer` contabiliza con una precisión de 10 [ns], la cantidad de ciclos de reloj comprendidos entre ambos sucesos, la cual luego es accedida por la CPU.

Se recopilaron 10 mil muestras en cada plataforma de software para asegurar una evaluación representativa.

La Tabla 5.3 resume los resultados obtenidos de las mediciones. Considerando que la latencia del hardware es de 200 [ns], los tiempos medidos en ambas plataformas de software son considerablemente mayores. En particular, en *Bare Metal*, el promedio y la mediana son similares, lo que, junto con la baja desviación estándar, sugiere una baja variabilidad en los tiempos medidos. Además, el peor caso en *Bare Metal* corresponde a 3.14 [μs], lo que indica que es adecuado usar esta plataforma de software para la aplicación, ya que permite cumplir con sus restricciones de tiempo (10 [ms]) en todos los casos observados.

Por otro lado, para el caso PYNQ, el tiempo promedio es notablemente mayor en comparación con la mediana, lo que indica una alta variabilidad en los tiempos medidos. Además, el promedio es más cercano al percentil 99 % que a la mediana, lo que sugiere la presencia de *outliers* significativos, reflejados en el valor máximo registrado. Esta variabilidad es común en sistemas operativos que no están orientados a la operación en tiempo real y se explica por el nivel de abstracción en el que se ejecuta el software, que depende de un sistema operativo que asigna prioridades a las tareas en

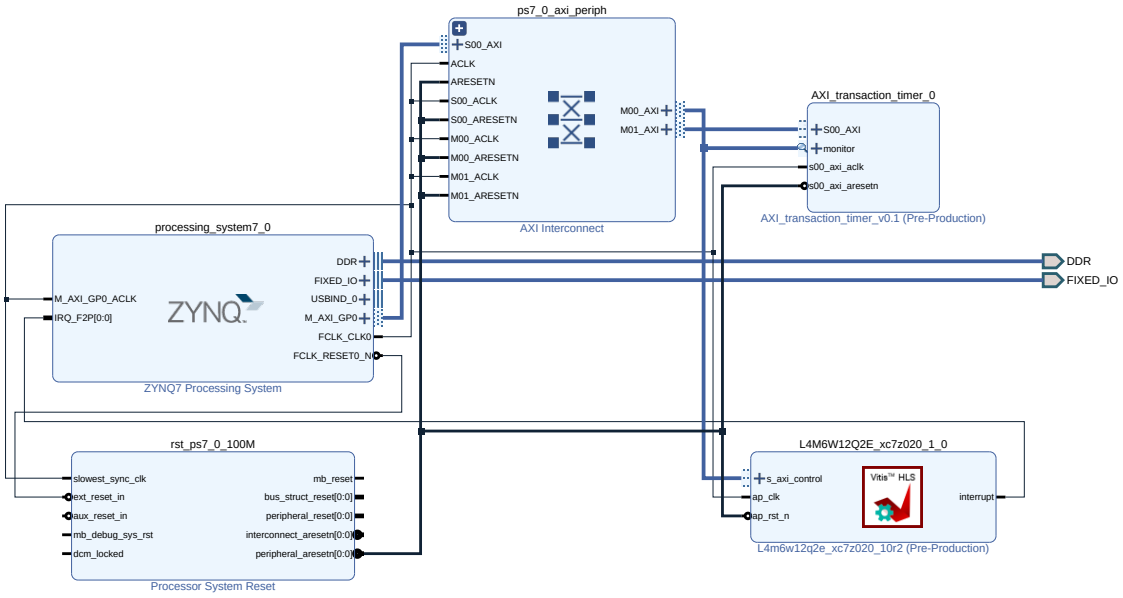


Figura 5.10: Diagrama de bloques de la plataforma de hardware implementada.

ejecución. En este contexto, aunque el tiempo promedio es considerablemente menor que requisito de la aplicación, el uso de PYNQ como plataforma de software no garantiza el cumplimiento estricto de las restricciones de tiempo, ya que el tiempo máximo medido supera el período de muestreo del motor DC.

En resumen, la variabilidad observada en PYNQ hace que esta plataforma no sea una opción viable en una implementación en tiempo real, dado que en el peor caso el controlador podría no cumplir con su función de llevar los estados del sistema a un valor deseado. Por otro lado, aunque también añade un *overhead* considerable en relación a la latencia de la red implementada, la alternativa *Bare Metal* sí es viable para ser utilizada en tiempo real. Este panorama subraya la importancia de elegir adecuadamente el software, ya que una selección inapropiada puede anular la aceleración lograda durante el diseño del hardware.

5.2 Caso de estudio: Recursos Energéticos Distribuidos

Los recursos energéticos distribuidos (DER, por sus siglas en inglés) son sistemas que combinan generación y almacenamiento de energía a nivel local, a menudo utilizando fuentes renovables como paneles solares. Estos sistemas se conectan a la red eléctrica mediante convertidores de potencia DC/AC, que transforman la energía de corriente directa (DC) en corriente alterna (AC) para su integración en la red. Los DER son fundamentales en las micro-redes, estructuras autónomas que mejoran la eficiencia, escalabilidad y resiliencia de los sistemas eléctricos. Sin embargo, su operación requiere un esquema de control preciso para regular los convertidores DC/AC y garantizar que la salida cumpla con las especificaciones de voltaje y frecuencia de la red, manteniendo condiciones de operación seguras.

5.2.1. Planteamiento MPC

El modelo del sistema DER utilizado incluye dos estados de corriente, dos estados de voltaje y dos entradas de voltaje, que corresponden a las señales aplicadas sobre el actuador. Los estados de corriente y las entradas están sujetos a restricciones no lineales, aproximadas mediante polígonos regulares [45]. Aunque un mayor número de lados en los polígonos mejora la aproximación, también incrementa el tamaño de las matrices y la cantidad de regiones en la ley de control. De acuerdo con [67], un polígono de 10 lados es suficiente para esta aplicación, por lo que se adopta esta recomendación para la implementación.

Los estados del sistema se representan como $\boldsymbol{x} = [x_0, x_1, x_2, x_3]^\intercal$, donde x_0 y x_1 son las corrientes del sistema, mientras que x_2 y x_3 corresponden a los voltajes. De manera similar, las actuaciones están dadas por $\boldsymbol{u} = [u_0, u_1]^\intercal$, las cuales representan los voltajes enviados al actuador del sistema. Además, el sistema posee dos señales de referencia $\boldsymbol{r} = [r_0, r_1]^\intercal$, utilizadas para el seguimiento de los estados x_2 y x_3. Las matrices que describen la dinámica del sistema son las siguientes:

$$\boldsymbol{A} = \begin{bmatrix} -\frac{R_f}{L_f} & W_b & -\frac{1}{L_f} & 0 \\ -W_b & -\frac{R_f}{L_f} & 0 & -\frac{1}{L_f} \\ \frac{1}{C_f} & 0 & -\frac{1}{R_L C_f} & W_b \\ 0 & \frac{1}{C_f} & -W_b & -\frac{1}{R_L C_f} \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \frac{1}{L_f} & 0 \\ 0 & \frac{1}{L_f} \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{C} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.4}$$

Los parámetros del sistema se describen a continuación:

$$\begin{aligned} R_f &= 0.065\ [\Omega], \\ L_f &= 3 \cdot 10^{-3}\ [H], \\ C_f &= 15 \cdot 10^{-6}\ [F], \\ V_{DC} &= 200\ [V], \\ W_b &= 2\pi \cdot 50\ [\text{rad}], \\ I_{nom} &= 8\ [A], \\ R_L &= 10\ [\Omega], \\ T_s &= 200 \cdot 10^{-6}\ [s], \end{aligned}$$

donde R_f, L_f, C_f son la resistencia, inductancia y capacitancia del filtro del sistema, respectivamente. Además, V_{DC} es el voltaje de la fuente DC conectada al convertidor, W_b es la frecuencia nominal, I_{nom} es la corriente nominal, R_L es la resistencia en la carga y T_s es la frecuencia de muestreo. Para más detalles sobre los parámetros y el modelo del sistema, se sugiere revisar [67].

Las restricciones en los estados se definen mediante un polígono compuesto de 10 rectas, que

aproximan una circunferencia de radio I_{nom}:

$$\begin{bmatrix} -24.6215 \\ -9.4046 \\ -7.6085 \\ -9.4046 \\ -24.6215 \end{bmatrix} \leq \begin{bmatrix} 3.0777 & 1 & 0 & 0 \\ 0.7265 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -0.7265 & 1 & 0 & 0 \\ -3.0777 & 1 & 0 & 0 \end{bmatrix} \boldsymbol{x} \leq \begin{bmatrix} 24.6215 \\ 9.4046 \\ 7.6085 \\ 9.4046 \\ 24.6215 \end{bmatrix}, \tag{5.5}$$

donde las columnas de ceros en (5.5) indican que los estados x_2 y x_3 no están sujetos a restricciones.

Las actuaciones también se restringen por un polígono compuesto de 10 rectas, pero que aproxima una circunferencia de radio $\frac{V_{DC}}{\sqrt{3}}$:

$$\begin{bmatrix} -355.3803 \\ -135.7432 \\ -109.8185 \\ -135.7432 \\ -355.3803 \end{bmatrix} \leq \begin{bmatrix} 3.0777 & 1 \\ 0.7265 & 1 \\ 0 & 1 \\ -0.7265 & 1 \\ -3.0777 & 1 \end{bmatrix} \boldsymbol{u} \leq \begin{bmatrix} 355.3803 \\ 135.7432 \\ 109.8185 \\ 135.7432 \\ 355.3803 \end{bmatrix}. \tag{5.6}$$

Basándose en el trabajo de [67], la formulación del controlador considera un horizonte de predicción $N = 2$ y las siguientes matrices de pesos:

$$\boldsymbol{Q} = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \boldsymbol{R} = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}. \tag{5.7}$$

A diferencia del caso de estudio del motor DC, en este sistema las señales de referencia varían dinámicamente, lo que es más representativo de las aplicaciones prácticas de control, donde las referencias suelen ajustarse según las condiciones operativas en lugar de permanecer fijas.

Además, en contraste con el caso de estudio anterior, donde se utilizó un controlador explícito como referencia, en este caso se emplea una formulación implícita de MPC, resuelta mediante el solver `quadprog` de Matlab. Como se mencionó en el Capítulo 3.2, el flujo de implementación es independiente de la formulación utilizada para obtener los datos de entrenamiento, por lo que se emplea el controlador implícito obtenido para este propósito.

5.2.2. Diseño de red neuronal cuantizada

Como se ilustra en la Figura 3.1, una vez obtenido el conjunto de datos de entrenamiento, es necesario diseñar una red cuantizada compatible con HLS4ML que cumpla con los requisitos de control de la aplicación en simulación. A continuación, se describe el proceso de diseño realizado para el sistema DER, el cual incluye la selección de hiperparámetros, el ajuste del nivel de cuantización y el entrenamiento de la red neuronal cuantizada.

5.2.2.1. Entrenamiento con precisión flotante

Dado que en este caso no se dispone de la ley de control de forma explícita, no es posible estimar preliminarmente una red neuronal que aproxime adecuadamente la ley de control. En consecuencia, se adopta un rango de exploración más amplio en comparación con el caso de estudio anterior, considerando entre 6 y 30 neuronas por capa ($M = [6, 30]$), y entre 1 y 8 capas ocultas ($L = [1, 8]$). El límite inferior de neuronas por capa es 6 y se establece según la cantidad de entradas de la red, que corresponde a la suma de los estados y las señales de referencia.

La Figura 5.11 muestra los resultados de la búsqueda de hiperparámetros para la red neuronal, destacando que la mayoría de las redes convergen a un MSE cercano a 10^{-5}, con excepción de aquellas con una o dos capas ocultas, cuyo error durante la exploración es mayor.

Como se discutió en el Capítulo 4.3, el número de multiplicaciones en la red es proporcional al uso de recursos en la FPGA. Para balancear el error de aproximación y el número de multiplicaciones

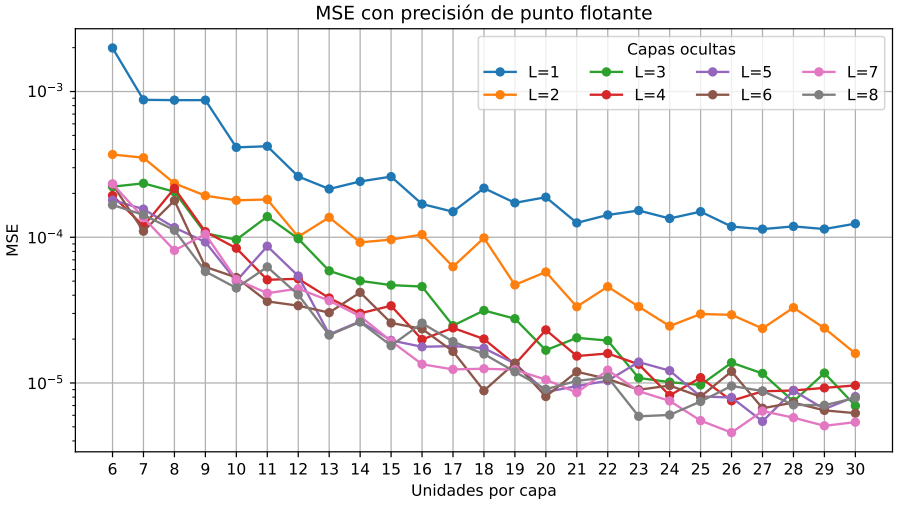


Figura 5.11: Resultados de la exploración de arquitecturas de red neuronal para el DER.

Tabla 5.4: Top-5 de combinaciones de hiperparámetros según la métrica presentada en el Anexo D.

Hiperparámetros ($L\times M$)	**Número de multiplicaciones**	**RMSE**	**MERS**
3×23	1242	$3.290 \cdot 10^{-3}$	0.9493
3×24	1344	$3.184 \cdot 10^{-3}$	0.9491
3×25	1450	$3.112 \cdot 10^{-3}$	0.9481
3×28	1792	$2.733 \cdot 10^{-3}$	0.9478
5×20	1760	$2.945 \cdot 10^{-3}$	0.9443

de la red, se utilizó una métrica denominada MERS (*Multiplications-Error Ranking Score*), diseñada específicamente para este propósito. Esta métrica asigna un puntaje entre 0 y 1, que refleja el desempeño relativo de cada red en términos de error de aproximación y número de multiplicaciones, en comparación con las demás redes evaluadas. Es importante destacar que el orden de las redes mejor evaluadas es lo relevante en esta métrica, ya que el puntaje depende del rango de exploración y de la variabilidad de los resultados en términos de error y número de multiplicaciones. Los detalles del cálculo de esta métrica se presentan en el Anexo D.

En este caso, se priorizó la precisión asignando un peso de 0.85 al error dentro del cálculo del MERS, con el objetivo de asegurar un bajo error de aproximación en el modelo escogido. En la práctica, la elección de los pesos de la métrica empleada requiere de un proceso iterativo de ajuste, cuya efectividad depende tanto de la aplicación específica como del rango de exploración seleccionado. La Tabla 5.4 muestra las 5 redes con mayor puntaje MERS dentro del conjunto de redes evaluadas en la Figura 5.11. Se observa que las redes con mejores puntajes poseen tres capas ocultas, lo que sugiere un balance adecuado entre multiplicaciones y precisión. En consecuencia, la combinación de hiperparámetros seleccionada para continuar el flujo de diseño corresponde a $L = 3$ y $M = 23$, con un MSE de $1.08 \cdot 10^{-5}$.

La Figura 5.12 muestra los resultados de la inferencia para la red seleccionada. Cada punto azul representa una inferencia en comparación con el valor esperado, obtenido mediante el controlador de referencia. La línea roja indica donde la actuación inferida es exactamente igual a la esperada. La línea segmentada gris indica la posición del origen de la actuación al revertir la normalización.

Aunque la mayoría de los puntos en la Figura 5.12 se encuentran cercanos a la línea roja, algunos valores presentan desviaciones significativas, lo que sugiere que ciertas regiones del espacio de estados podrían estar subrepresentadas en el conjunto de entrenamiento. Esto podría ser problemático en la aplicación si estas desviaciones ocurren en estados críticos del sistema. En particular, en la actuación u_1 (gráfico derecho), se observa que, a pesar de presentar un coeficiente de determinación (R^2) ligeramente más alto que u_0, algunos valores exceden el rango $[0, 1]$, lo que indica que esa acción de control podría infringir las restricciones al tomar valores fuera del límite de actuación considerado.

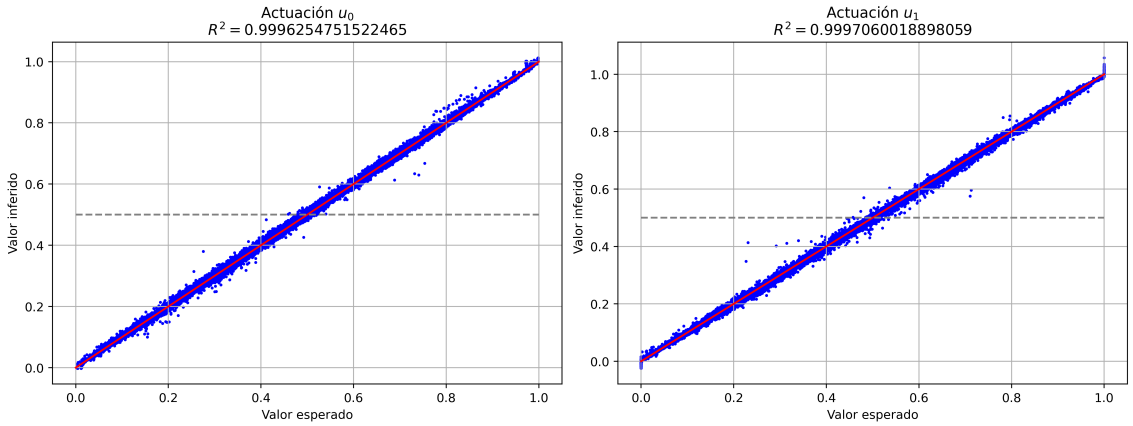


Figura 5.12: Relación entre valor de predicción y valor esperado para las dos salidas de la red de 3 capas y 23 neuronas de precisión flotante.

Estas observaciones resaltan la importancia de evaluar la red neuronal en simulación, ya que al generarse los estados de prueba de manera aleatoria, es posible que algunos puntos mostrados no sean estados alcanzables en la práctica, reduciendo la relevancia de ciertas desviaciones en la predicción.

El comportamiento simulado de la red neuronal con $L = 3$ y $M = 23$ se presenta en la Figura 5.13. La simulación se realizó utilizando cinco valores diferentes para la señal de referencia, con una duración de 10 [ms] cada uno. En particular, en los intervalos $[10, 20]$ [ms] y $[30, 40]$ [ms], se busca evaluar la respuesta del controlador en condiciones límite del sistema, donde seguir las señales de referencia implica exceder las restricciones. Además, el caso del intervalo $[10, 20]$ [ms] se encuentra fuera del espacio de muestreado por los datos de entrenamiento.

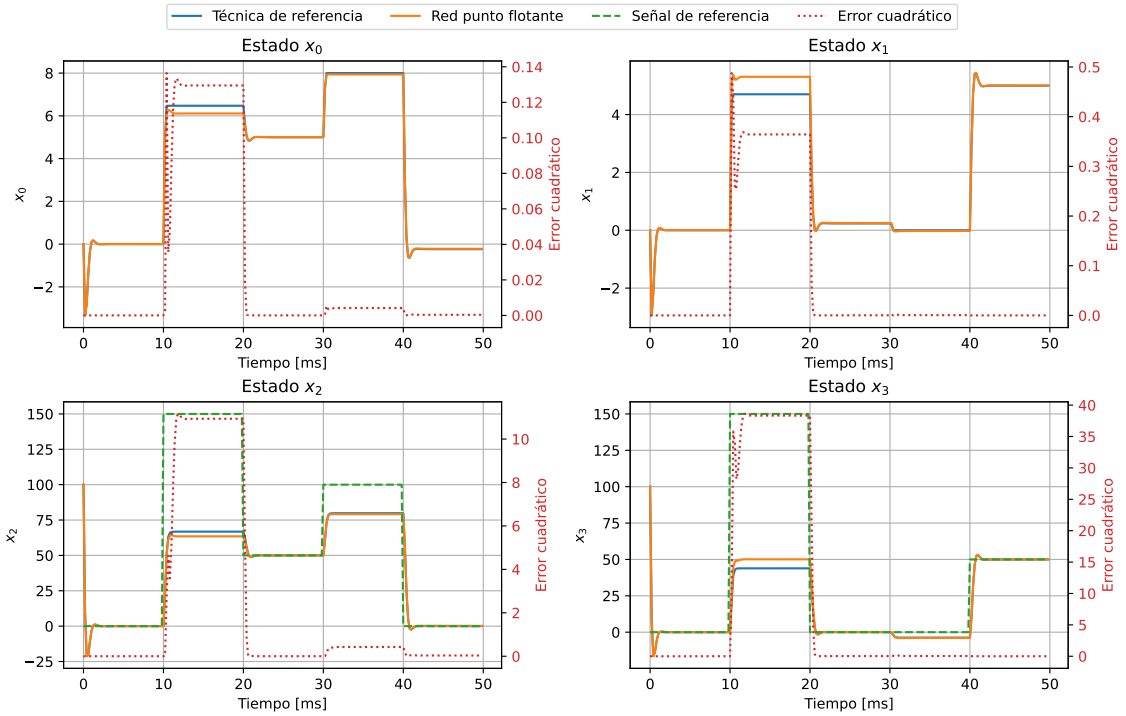
En la Figura 5.13 se observa que el MSE de los estados y la actuación es significativamente mayor en el primer caso límite ($[10, 20]$ [ms]), donde se intentó sobrepasar las restricciones. Esto se debe a que este caso no fue considerado en el conjunto de datos de entrenamiento, por lo que la red no tuvo ejemplos para manejar adecuadamente esta condición. Este problema podría abordarse ajustando la estrategia de recolección de datos, considerando un espacio de muestreo mayor en condiciones límite. Sin embargo, para el resto de las condiciones simuladas, las actuaciones y los estados presentan un comportamiento similar al del controlador de referencia.

La Figura 5.14 presenta un análisis del cumplimiento de las restricciones durante la simulación. Se observa que, en el intervalo $[10, 20]$ [ms], ambas restricciones son sobrepasadas, siendo más evidente en el voltaje de actuación, donde la restricción se infringe por un período más breve que en la restricción de corriente. En particular, existe un pico de actuación ocurrido en el instante 10 [ms], como se muestra en la Figura 5.13b. Además, la restricción de corriente, asociada a los estados x_0 y x_1, aunque es infringida durante un período de tiempo más extenso, se considera dentro de un rango tolerable para la aplicación.

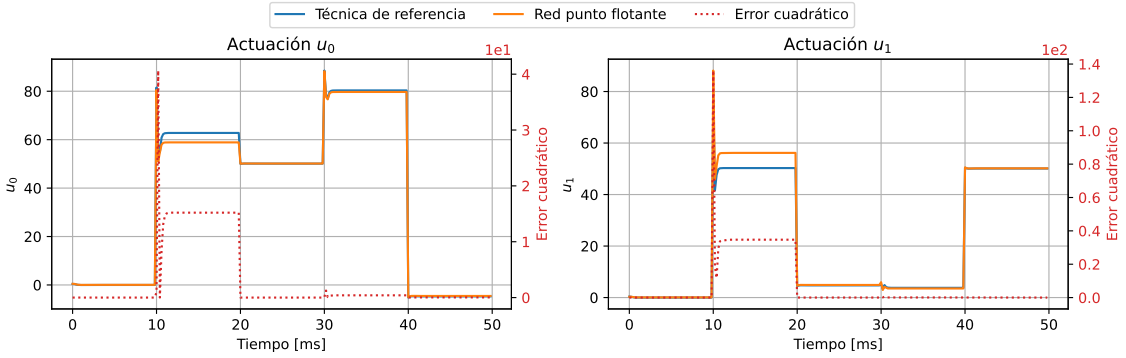
Pese a las diferencias observadas en las condiciones límite, la red cumple con los criterios de desempeño de control en los escenarios evaluados. A continuación, se determinará el nivel de cuantización adecuado para su implementación en hardware.

5.2.2.2. Entrenamiento cuantizado

Una vez seleccionada la combinación de hiperparámetros de la red, es necesario determinar un nivel de cuantización adecuado que mantenga la similitud con el controlador de referencia. Para ello, se exploran niveles de cuantización en el rango $W \in [2, 32]$ para el número total de bits y $Q \in [1, 6]$ para los bits asignados a la parte entera y signo. Estos rangos se eligen porque 32 bits es un ancho ampliamente utilizado en sistemas embebidos, mientras que 6 bits para la parte entera se consideran suficientes, dado que los datos de entrenamiento y los pesos de la red están normalizados, además de que experimentos preliminares mostraron que no hay diferencia significativa al incrementar más la cantidad de bits Q.



(a) Comportamiento de estados.



(b) Comportamiento de actuaciones.

Figura 5.13: Resultados de simulación de la red neuronal de punto flotante.

Los resultados de la exploración se presentan en la Figura 5.15, donde se incluye como referencia el MSE de la red de punto flotante (línea segmentada negra). En la figura, se observa que la cuantización con $Q = 3$ alcanza el MSE de la red en punto flotante con un total de 13 bits. Por lo tanto, se selecciona la red cuantizada con los parámetros $W = 13$ y $Q = 3$, obteniendo un MSE de $1.08 \cdot 10^{-5}$.

La Figura 5.16 muestra los resultados de la inferencia de la red cuantizada con 13 bits. En comparación con la red en punto flotante (Figura 5.12), también se observan puntos alejados de la recta roja en ambas actuaciones, así como valores fuera del rango $[0, 1]$. Dado que no se aprecian diferencias visibles respecto a la red de punto flotante, es esperable que el desempeño en simulación de la red cuantizada sea similar al de la red de punto flotante.

Tras evaluar el impacto de la cuantización en la precisión de la red, se procede a validar su desempeño en simulación. La Figura 5.17 muestra los resultados de las simulaciones de la red cuantizada y el controlador de referencia. En comparación con la simulación de la red de punto flotante, mostrada en la Figura 5.13, se observa que en el rango $[10, 20]$ [ms], que es donde se presenta el error más pronunciado en ambas simulaciones, el error de la red cuantizada es mayor

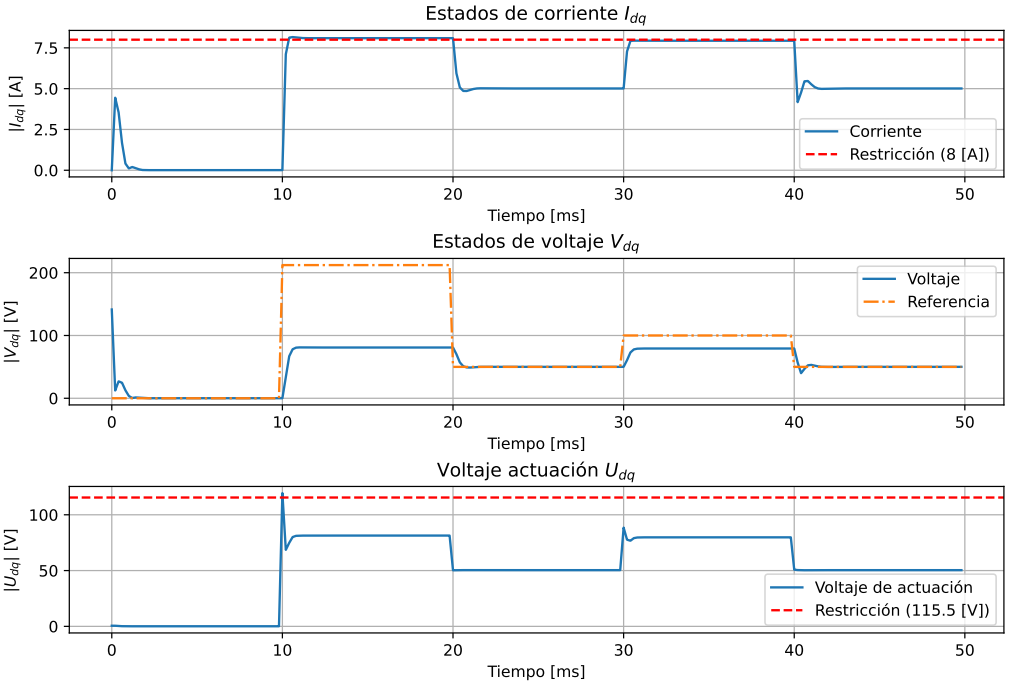


Figura 5.14: Análisis de restricciones en simulación con la red de punto flotante de 3 capas y 23 neuronas.

en magnitud para todos los estados y actuaciones. Sin embargo, el pico de error en u_1 ubicado en 10 [ms] se reduce considerablemente en la red cuantizada.

Fuera del intervalo $[10, 20]$ [ms], no se observan diferencias significativas entre la red cuantizada y el controlador de referencia, lo que indica que la red cuantizada es capaz de llevar el sistema al estado indicado por la referencia cuando las restricciones lo permiten.

La Figura 5.18 muestra un análisis de las restricciones del sistema en la simulación con la red cuantizada. Aunque no se cumplen las restricciones durante toda la simulación, los momentos en que estas se infringen son los mismos observados con la red de punto flotante, lo que indica que la cuantización no introdujo un impacto adicional en el cumplimiento de restricciones.

Las observaciones realizadas en la simulación indican que la red cuantizada mantiene el desempeño comparable al de la red de punto flotante en términos de control. Por lo tanto, la red cuantizada seleccionada cumple con los criterios de desempeño establecidos, lo que permite avanzar a la siguiente etapa de diseño de hardware.

5.2.3. Diseño de hardware

Esta sección abarca desde la conversión del modelo cuantizado entrenado en software a código compatible con HLS hasta las etapas de síntesis e implementación, las cuales se ilustran en la Figura 3.1.

La implementación de este caso de estudio se realiza en la tarjeta de desarrollo ZCU104 de AMD/Xilinx, que ya fue usada en el Capítulo 4. Para la operación del sistema, se emplea un reloj de 100 [MHz], que corresponde a la frecuencia por defecto que utilizan Vitis HLS y Vivado. La estrategia de HLS4ML utilizada es *Latency*, la cual evita el uso de BRAM en las implementaciones.

El modelo de red neuronal diseñado en la sección anterior debe realizar 1242 operaciones de multiplicación. Dado que se seleccionó una cuantización de 13 bits, como se discutió en el Capítulo 4.6, se espera un uso máximo de 1242 DSPs, un equivalente al 72 % de los DSPs disponibles, en caso de emplear un factor de reutilización $R = 1$.

La Figura 5.19a muestra el impacto del factor de reutilización sobre el uso de recursos para la red cuantizada. En particular, se observa que con $R = 1$, el uso de DSPs es aproximadamente 42 %

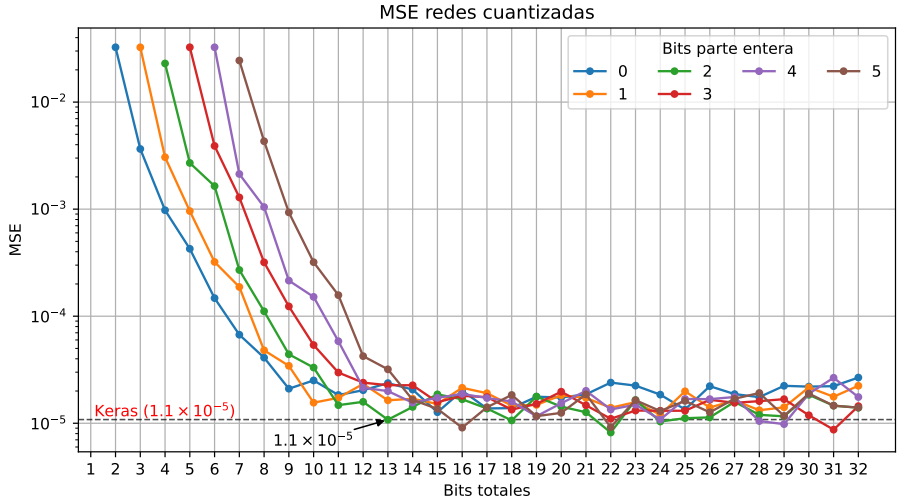


Figura 5.15: Resultados de la exploración de niveles de cuantización de la red neuronal para el DER.

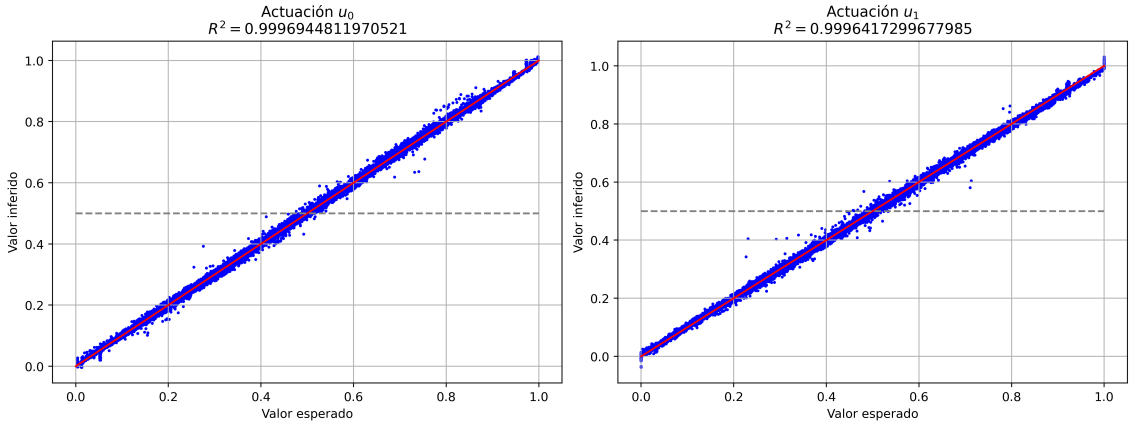


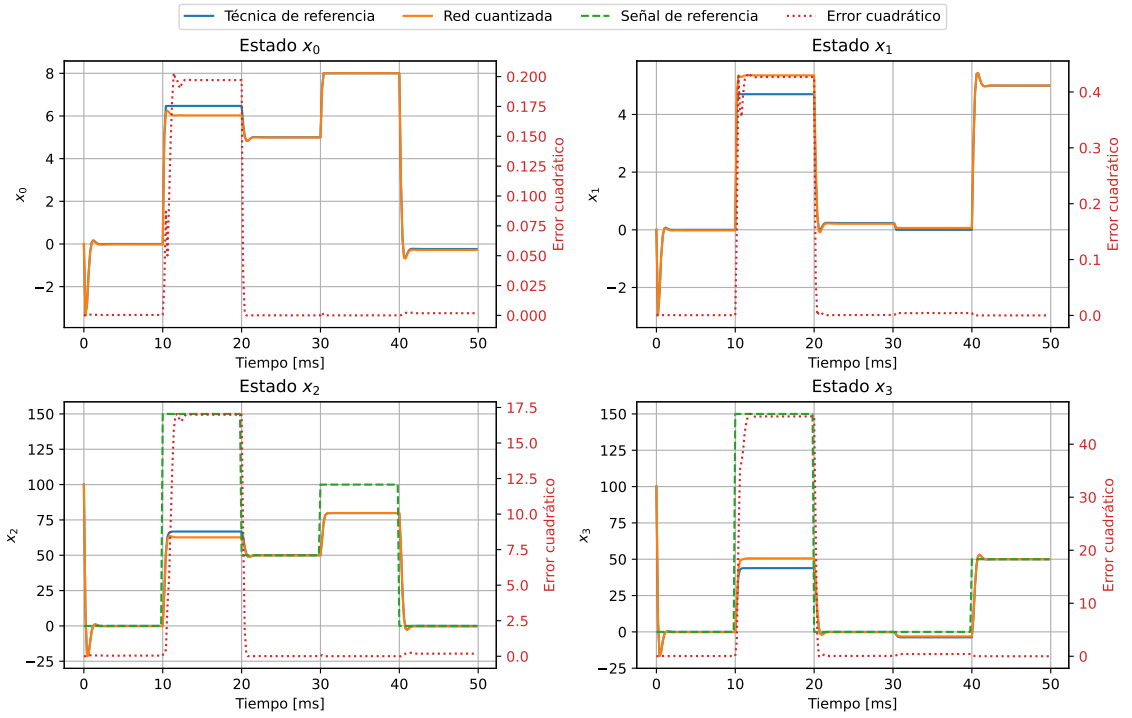
Figura 5.16: Relación entre valor de predicción y valor esperado para las dos salidas de la red de 3 capas y 23 neuronas cuantizada con 13 bits.

del total disponible en la FPGA, un valor considerablemente menor al 72 % esperado. Al igual que en el caso de estudio anterior, no se aprecia una variación significativa en el uso de FFs y LUTs en función del factor de reutilización. Este comportamiento es esperable, según lo discutido en el Capítulo 4.7, dado que no se usa el total de DSPs de la tarjeta de desarrollo, por lo que no hay un uso intensivo de LUTs para implementar multiplicadores. Además, la Figura 5.19b muestra que la latencia mínima de la red se alcanza con los factores de reutilización iguales a 2, 3 y 4.

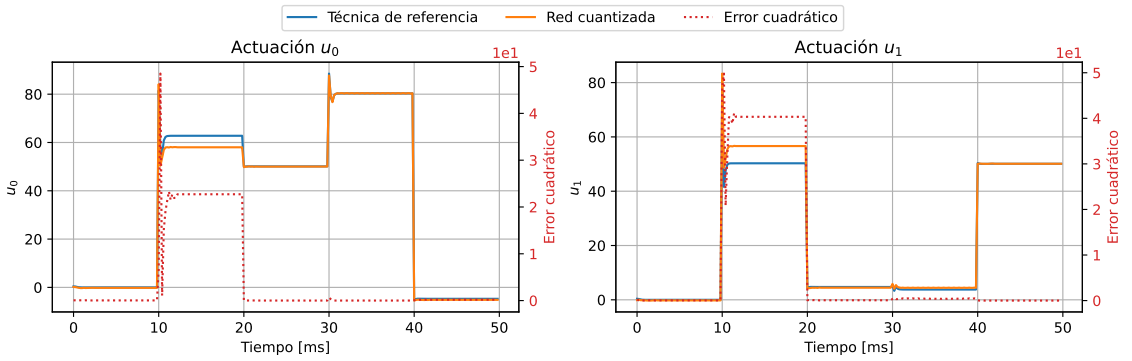
Si bien todos los factores de reutilización evaluados permiten que la red neuronal sea implementable y cumplen ampliamente con el requisito de latencia de 200 [μs] establecido por la aplicación, la opción que ofrece el mejor compromiso entre latencia y uso de recursos es $R = 4$, que ofrece una latencia de 180 [ns] y que se adopta en la implementación final.

5.2.4. Validación de hardware

Para finalizar el flujo de diseño e implementación del controlador MPC basado en redes neuronales, es necesario evaluar el desempeño en la aplicación del hardware resultante, a través de una planta simulada. De esta forma, es posible determinar si el controlador MPC en hardware cumple con los requerimientos esperados desde el punto de vista de la aplicación. En particular, para este caso de estudio los requisitos esperados corresponden a un nivel de similitud adecuado respecto del controlador de referencia, que permita lograr un adecuado seguimiento de las referencias de control



(a) Comportamiento de estados.



(b) Comportamiento de actuaciones.

Figura 5.17: Resultados de simulación de la red neuronal cuantizada.

al tiempo que cumple con las restricciones del sistema.

La Figura 5.20 muestra los comportamientos de estados y actuaciones para la simulación realizada utilizando el hardware implementado en FPGA. Como en las simulaciones previas, la mayor diferencia entre el controlador de referencia y el hardware ocurre en el intervalo $[10, 20]$ [ms], cuando las restricciones de corriente impiden que la señal de referencia sea alcanzada. Además, en las actuaciones se notan oscilaciones en el intervalo $[30, 40]$ [ms] las que afectan la estabilidad del sistema, propagándose a los estados x_1 y x_3. Sin embargo, la magnitud de estas oscilaciones no es significativa y existen métodos para manejar este problema en el controlador final. En comparación con la red neuronal cuantizada en software, el error en hardware se mantiene en el mismo orden de magnitud durante toda la simulación, indicando que la implementación en FPGA conserva la precisión del modelo.

En términos del cumplimiento de las restricciones, la Figura 5.21 muestra que, a diferencia de las simulaciones en software, el hardware se mantiene por debajo del máximo de corriente en el intervalo $[10, 20]$ [ms] y no rompe las restricciones de actuación en 10 [ms]. A pesar de ello, si se observa una mínima transgresión de las restricciones de corriente en el intervalo $[30, 40]$ [ms], que es

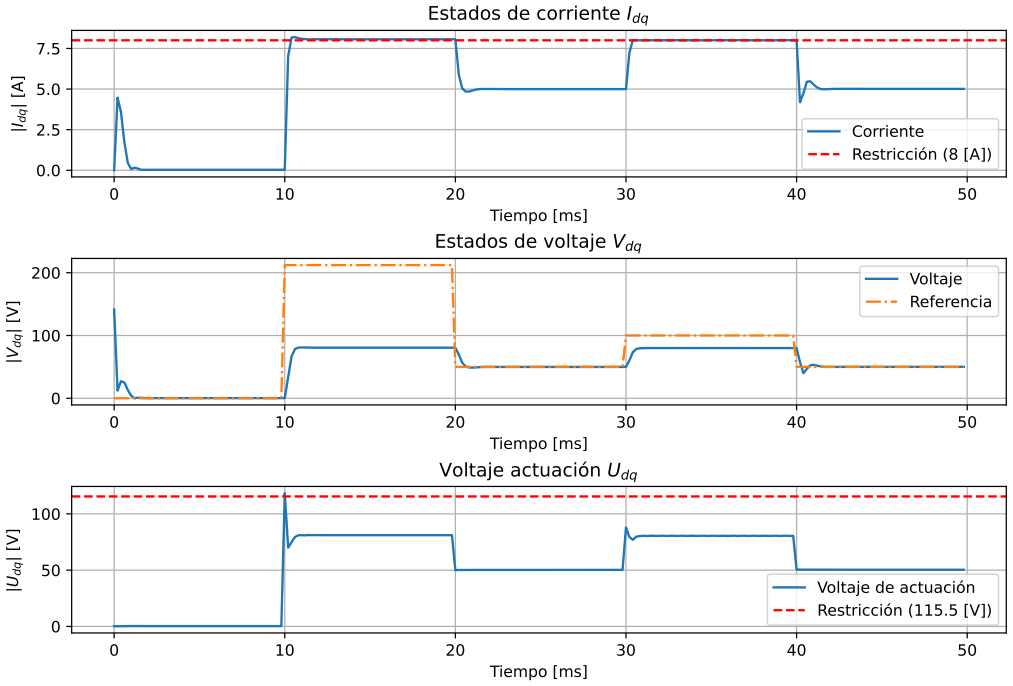


Figura 5.18: Análisis de restricciones en simulación con la red cuantizada con 13 bits.

esperable debido a que ya fue observado en las simulaciones anteriores.

La Tabla 5.5 resume las métricas obtenidas durante cada una de las simulaciones realizadas durante el flujo de diseño. La tabla muestra el MSE por separado para estados y actuaciones, donde se puede observar que, al cuantizar la red de punto flotante, aumentó el error en todas las señales. Sin embargo, para la red implementada en hardware, sólo los estados x_0, x_2 y la actuación u_0 aumentaron su error en relación a la red de punto flotante. En contraste, para los estados x_1, x_3 y la actuación u_1 el MSE medido en hardware es incluso menor que el error de la red de punto flotante.

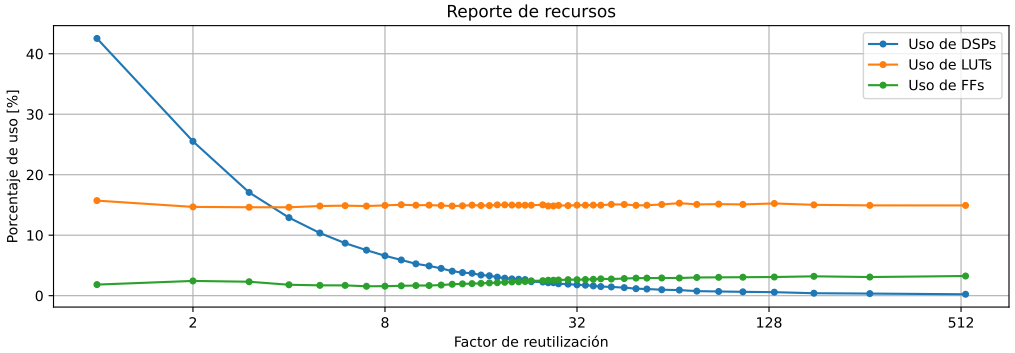
El tiempo de asentamiento reportado en la Tabla 5.5 se define como el tiempo que tarda la señal en alcanzar el 98 % de su cambio tras una variación en la referencia, lo que permite medir cuánto demora el controlador en lograr el seguimiento de la referencia. Según la Tabla 5.5, los tiempos de asentamiento de todas las señales aumentaron. El principal motivo de esto, corresponde a las oscilaciones observadas en el intervalo $[30, 40]$ [ms], lo que resulta en un asentamiento tardío de las señales, especialmente de x_1, x_3 y u_1. Aunque la oscilación de las actuaciones es un comportamiento indeseado, es importante destacar que su magnitud es leve y es posible utilizar técnicas, como el uso de un integrador, para mitigar su efecto en la aplicación.

El MSE medido en estado estacionario mostrado en la Tabla 5.5 es similar al MSE durante toda la simulación para todas las señales, lo que sugiere un equilibrio entre el aprendizaje de las regiones de la ley de control y el comportamiento de las redes en torno a los puntos de operación estudiados.

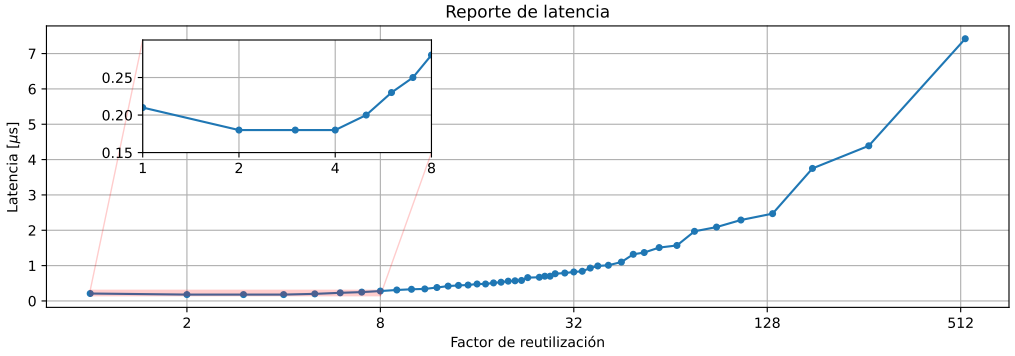
Los costos de control promedio, si bien son similares entre los modelos de las redes en software, el mayor costo de control está en la implementación en hardware. Sin embargo, el alza del costo de control del hardware es de un 3 % del costo del controlador de referencia, lo que no es considerable dado el orden de magnitud de estos valores.

Además, la Tabla 5.5 muestra los valores máximos alcanzados por las señales mostradas en la Figura 5.21. Como se observó en las Figuras 5.14 y 5.18, las restricciones de corriente se infringieron desde el inicio del flujo, con la red de punto flotante. Sin embargo, el valor de la menor transgresión a estas restricciones corresponde a la red implementada en hardware. Por otro lado, en cuanto a la restricción de voltaje de actuación, aunque las redes en software violaron las restricciones, la red en hardware las cumplió durante toda la simulación, como se muestra en la Figura 5.21.

Finalmente, la Tabla 5.6 resume los tiempos de ejecución medidos para la red implementada en hardware utilizando una plataforma de software *Bare Metal*. Los tiempos de ejecución se midieron



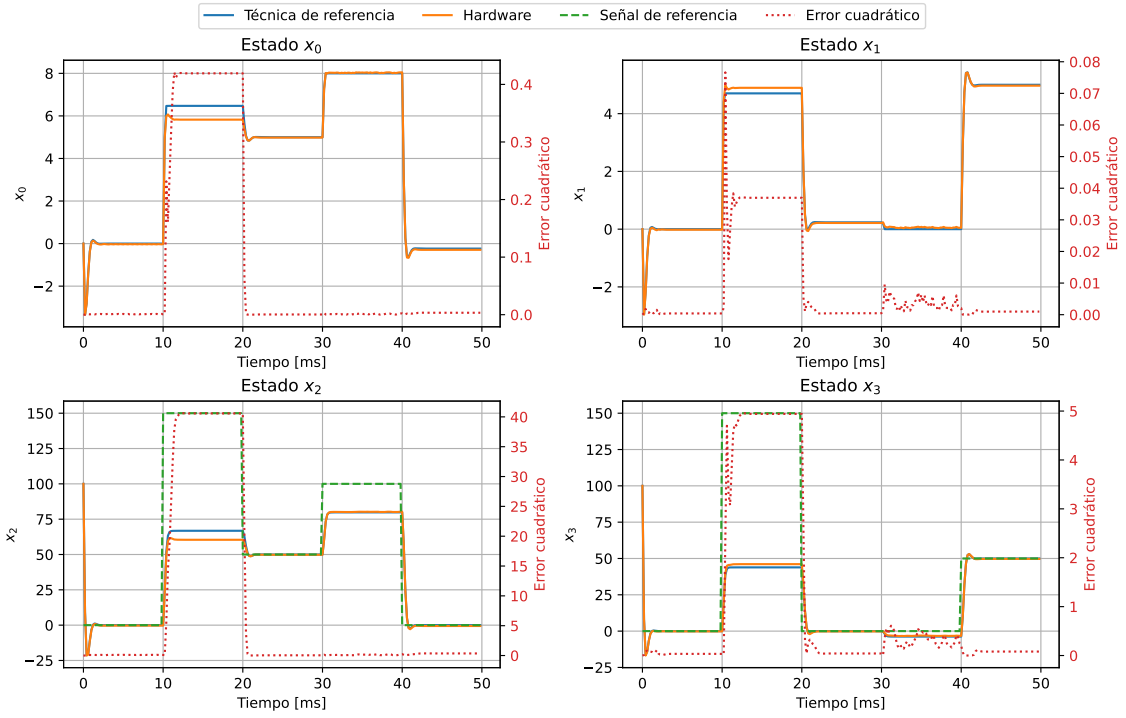
(a) Reportes de uso de recursos.



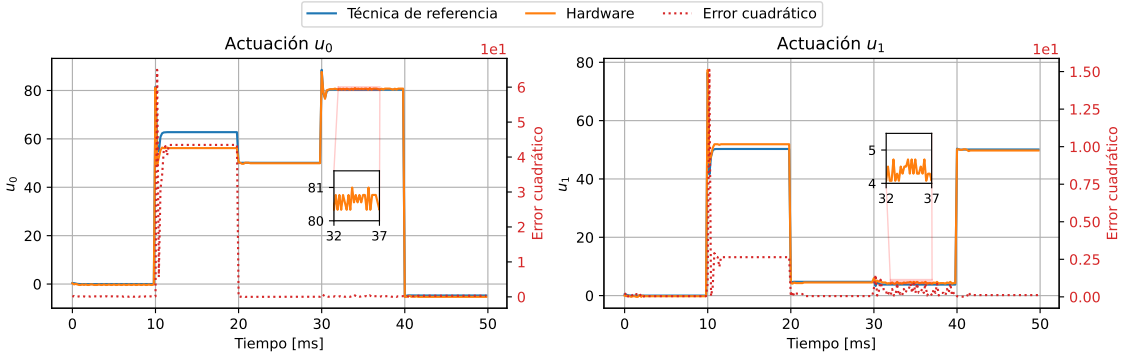
(b) Reportes de latencia.

Figura 5.19: Uso de recursos y latencia reportados por Vitis HLS según el factor de reutilización para la red cuantizada a 13 bits de 3 capas ocultas y 23 neuronas por capa.

en hardware, utilizando la misma estrategia del caso de estudio anterior, midiendo desde la escritura del primer dato de entrada desde el CPU, hasta la lectura del último dato de salida de la red. En particular, aunque la latencia del hardware implementado es de 180 [ns], el tiempo promedio medido es de 2.2 [μs], lo que resulta significativamente mayor en relación con la latencia de la red. Por otro lado, la baja variabilidad observada en la mediana, desviación estándar y el percentil 99 % indica que el tiempo de ejecución en hardware es estable y predecible. Además, el tiempo máximo de 2.81 [μs] sugiere que la red es adecuada para implementarse en un sistema de tiempo real, ya que en todos los casos cumple con el requisito y deja suficiente margen para la ejecución de la acción de control entre cada instante de muestreo, que en esta aplicación es de 200 [μs].



(a) Comportamiento de estados.



(b) Comportamiento de actuación.

Figura 5.20: Resultados de simulación con la red neuronal en FPGA.

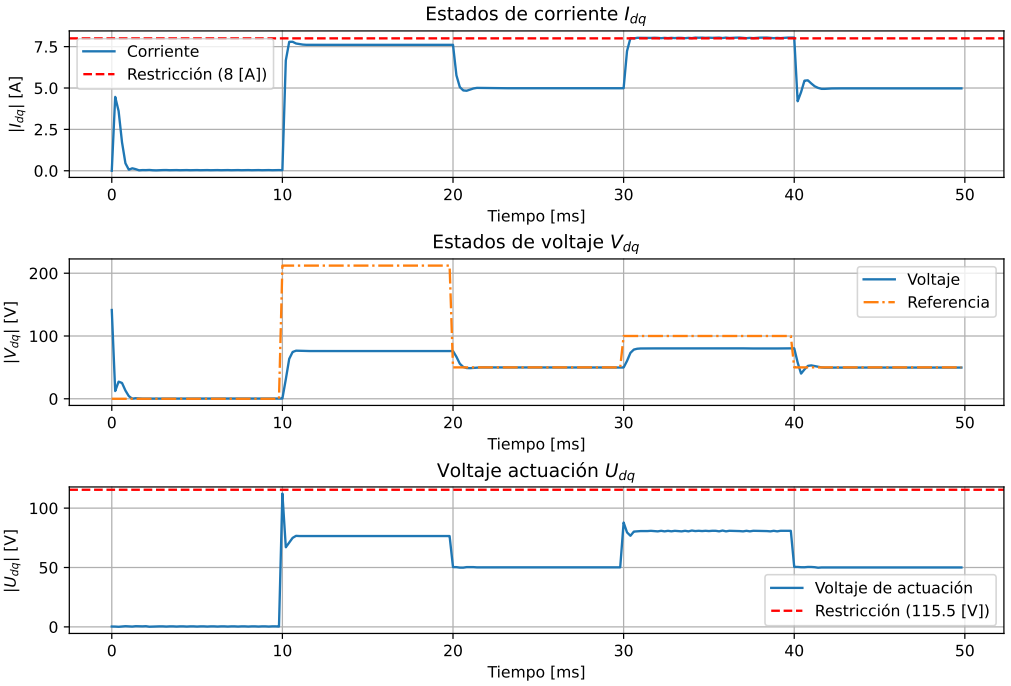


Figura 5.21: Análisis de restricciones en simulación con hardware.

Tabla 5.5: Métricas obtenidas durante las simulaciones realizadas para el DER.

Métrica	**Señal**	**Controlador de referencia**	**Red en punto flotante**	**Red cuantizada**	**Red en hardware**
MSE	x_0	0	2.55e-02	3.76e-02	8.04e-02
	x_1	0	7.30e-02	8.50e-02	8.40e-03
	x_2	0	2.13e+00	3.20e+00	7.72e+00
	x_3	0	7.54e+00	8.89e+00	1.05e+00
	u_0	0	3.08e+00	4.49e+00	8.47e+00
	u_1	0	7.18e+00	8.13e+00	6.82e-01
Tiempo de asentamiento promedio [ms]	x_0	1.04	1.16	1.16	1.28
	x_1	0.96	1.12	0.96	2.84
	x_2	1.08	1.12	1.16	1.20
	x_3	1.12	1.24	1.12	2.92
	u_0	0.72	0.68	1.00	4.56
	u_1	0.68	0.60	2.72	3.20
MSE estacionario	x_0	0	2.66e-02	3.96e-02	8.43e-02
	x_1	0	7.23e-02	8.58e-02	8.27e-03
	x_2	0	2.21e+00	3.34e+00	8.18e+00
	x_3	0	7.61e+00	9.08e+00	1.05e+00
	u_0	0	3.09e+00	4.56e+00	8.75e+00
	u_1	0	6.93e+00	8.19e+00	6.22e-01
Costo de control promedio	-	383,876	369,493	369,330	396,036
Magnitud máxima	I_{dq} [A]	8.00	8.16	8.20	8.05
	U_{dq} [V]	111.81	119.42	118.43	112.22

Tabla 5.6: Resumen de mediciones de tiempo realizadas para el DER.

Métrica	***Bare Metal***
Promedio	2.20 [μs]
Máximo	2.81 [μs]
Mediana	2.20 [μs]
Desviación estándar	0.02 [μs]
Percentil 99 %	2.23 [μs]

Capítulo 6

Conclusiones y trabajo futuro

Este trabajo propone un flujo sistemático para implementar en FPGA controladores MPC basados en redes neuronales. A partir de un controlador MPC previamente validado, se generan datos representativos para entrenar redes neuronales en precisión flotante, seguido de una exploración de niveles de cuantización para la red escogida. Mediante herramientas de alto nivel (HLS4ML/Vitis HLS), se analizan alternativas de arquitectura para la red cuantizada, que posteriormente se sintetiza e implementa en FPGA. La funcionalidad de la red se evalúa en cada etapa mediante simulaciones en lazo cerrado, y finalmente se valida en hardware a través de una plataforma de software. Además, se analiza el impacto de los hiperparámetros, niveles de cuantización y paralelismo en el uso de recursos y la latencia de la implementación, estableciendo directrices de diseño que facilitan el balance de *trade-offs*. El flujo y las directrices se evalúan en dos casos de estudio: un motor DC y un sistema DER, con el objetivo de evaluar su escalabilidad y adaptabilidad a distintas aplicaciones.

A continuación, se presentan los principales hallazgos de este trabajo y se discuten posibles direcciones para trabajos futuros.

6.1 Conclusiones

Esta tesis presenta un flujo estructurado para el diseño e implementación de aceleradores en FPGA destinados a aproximadores de MPC mediante redes neuronales, empleando técnicas y herramientas especializadas. Las directrices propuestas facilitan la integración de las distintas etapas del proceso, permitiendo equilibrar los compromisos entre latencia, uso de recursos y desempeño del lazo de control en la aplicación objetivo. Los resultados de los casos de estudio validan la efectividad del flujo, confirmando las caracterizaciones realizadas y logrando un balance adecuado en términos de uso de recursos, error de aproximación en las acciones de control y latencia del controlador.

En la práctica se evidenció la necesidad de complementar la red neuronal del controlador con técnicas adicionales que eviten *offsets* y oscilaciones indeseadas, tal como se ha discutido en la literatura [13, 28]. Por otro lado, los resultados del sistema DER mostraron que el error de aproximación aumentó significativamente cuando la red neuronal recibió señales de referencia que no estaban incluidas en el conjunto de datos de entrenamiento. Aunque la generación de estos datos no se abordó en profundidad en este trabajo, es importante destacar que, a diferencia de los estados del sistema, que dependen de su dinámica, las señales de referencia son definidas por el diseñador y, según la aplicación, pueden estar acotadas a un rango de valores. Este aspecto puede aprovecharse en una estrategia de recolección de datos que se enfoque en el rango de referencias utilizadas en la práctica, evitando el muestreo de valores irrelevantes y optimizando la cobertura del conjunto de datos de entrenamiento. De esta manera, es posible mejorar el desempeño de la red neuronal en las condiciones operativas reales del sistema.

Adicionalmente, las mediciones de tiempo revelaron en ambos casos que, aunque la latencia de la red implementada en hardware es considerablemente inferior al tiempo de muestreo, la incorporación de los tiempos de comunicación y respuesta del software incrementa significativamente el tiempo necesario total. Este hallazgo subraya la importancia de explorar y evaluar diferentes protocolos

de comunicación y plataformas de software que permitan aprovechar la aceleración lograda y garantizar el funcionamiento en tiempo real del controlador. En contraste, también se destaca que las aceleraciones logradas en comparación con la literatura proporcionan una mayor holgura temporal para la comunicación del sistema, lo que abre posibilidades la operación en tiempo real a sistemas que previamente no cumplían con estos requisitos.

En comparación con otras implementaciones reportadas en la literatura, los casos de estudio mostraron una latencia notablemente menor, incluso frente a implementaciones con redes *shallow* que requieren menos multiplicaciones para realizar inferencia. Sin embargo, esta aceleración se obtuvo a costa de un mayor uso de recursos, lo que evidencia un compromiso inherente al *framework* utilizado. En este sentido, HLS4ML resulta adecuado para implementar controladores MPC que alcancen una aceleración considerable en plataformas con suficiente holgura de recursos.

Gracias a la generalidad del flujo y las directrices de diseño, los resultados son extendibles a cualquier aplicación que utilice una red neuronal con el mismo nivel de cuantización e hiperparámetros. La selección de nivel de cuantización e hiperparámetros en la etapa previa al diseño del hardware resulta determinante para balancear el uso de recursos, especialmente de LUTs y FFs, y el error de aproximación de la ley de control. Los hallazgos indican que la relación entre número de multiplicaciones y el uso de recursos permite un ajuste preliminar de estos compromisos; no obstante, es necesaria una sistematización para identificar el balance óptimo en cada caso.

Con el fin de evaluar cuantitativamente el compromiso entre el número de multiplicaciones y el error de aproximación, se propuso la métrica MERS, aplicada en uno de los casos de estudio. Aunque el uso de esta métrica no es indispensable, su aplicación permite sistematizar la evaluación del compromiso entre error de aproximación y uso de recursos en la implementación.

Finalmente, el flujo propuesto facilita la implementación en hardware sin la necesidad de escribir código RTL, lo que reduce significativamente la barrera de entrada para usuarios sin conocimientos profundos en arquitecturas de hardware o HLS. Al mismo tiempo, el uso de herramientas de generación de código asistido, como HLS4ML, permite acelerar la exploración del espacio de diseño, favoreciendo el hallazgo de las implementaciones con las características de interés. En síntesis, se ha desarrollado y evaluado un flujo integral para el diseño e implementación en FPGA de controladores MPC basados en redes neuronales, que incorpora directrices comunes y permite consistentemente implementar aceleradores que equilibran de manera adecuada latencia, uso de recursos y precisión en la aproximación de la ley de control.

6.2 Trabajo futuro

A partir del trabajo realizado y los resultados obtenidos, se reconoce que persisten desafíos abiertos y oportunidades de mejora, los que pueden ser abordados a futuro. Algunos de los desafíos abiertos incluyen:

- **Incluir técnicas que garanticen una operación segura:** Debido a que ofrecer garantías de cumplimiento de restricciones y estabilidad es un tema de interés en el control automático, resulta relevante ampliar el flujo de diseño para considerar el uso de algoritmos que permitan ofrecer garantías adicionales al controlador. En particular, el algoritmo propuesto en [52] se presenta como una opción prometedora, ya que permite filtrar las acciones de control que podrían llevar al sistema a violar las restricciones de operación.

- **Caso de estudio con perturbaciones:** Aunque los casos de estudio no contemplaron perturbaciones, en aplicaciones de control su presencia es habitual. Por ello, sería interesante incluir un nuevo caso de estudio que evalúe su impacto en el sistema y analice cómo afectan los parámetros de diseño de las redes en el flujo.

- **Evaluación de protocolos y plataformas de software:** Las mediciones de tiempo realizadas resaltaron la importancia de caracterizar los tiempos efectivos de uso de las redes neuronales. Por este motivo, es importante realizar una exploración de opciones que permitan seleccionar el protocolo y la plataforma de software más adecuados para la operación del controlador en tiempo real.

- **Exploración de métodos de generación de conjuntos de datos:** Aunque este aspecto se abordó de manera superficial, el conjunto de datos de entrenamiento influye en el desempeño del controlador basado en la red neuronal. En particular, los resultados mostraron la importancia de incluir las señales de referencia en el conjunto de datos. Por ello, resulta interesante evaluar cómo distintos métodos de obtención de datos afectan el rendimiento del controlador en simulación, especialmente en el seguimiento de referencia.
- **Extensión de técnicas de optimización usando HLS4ML:** La exploración de estrategias avanzadas de optimización dentro de HLS4ML resulta prometedor para extender las posibilidades de balance del flujo. En particular, la asignación diferenciada de niveles de cuantización y factores de reutilización en distintas secciones de la red podría ser una alternativa para reducir el uso de recursos sin afectar significativamente el desempeño. Asimismo, el *pruning* ha demostrado en [24] reducir el uso de recursos a cambio de una penalización en el error de aproximación. Evaluar estas estrategias en conjunto permitiría ampliar el flujo de diseño, contribuyendo a implementaciones en FPGA con mejores balances en los compromisos de diseño.
- **Incorporar el consumo energético:** Imitando a otros flujos de diseño que incorporan HLS4ML, resulta interesante incorporar el consumo energético de la red como una métrica relevante para el balance de los compromisos de diseño. Dado el crecimiento de las aplicaciones con restricciones de potencia, un equilibrio inadecuado del consumo energético del controlador implementado puede ser limitante para ciertas aplicaciones.
- **Evaluación de otros frameworks para implementar redes:** Dado que existen diversos frameworks para implementar redes neuronales en FPGA, resulta interesante explorar de flujos alternativos utilizando herramientas distintas a HLS4ML. Esta exploración ampliaría las opciones de arquitecturas de hardware disponibles, brindando al diseñador la posibilidad de escoger la que mejor se adapte a las necesidades específicas de su aplicación.

Apéndice A

Máximo de multiplicaciones con unidades neuronales constantes

Este anexo presenta un análisis teórico que complementa la discusión en la Sección 4.3 de esta tesis. En particular, se examina la redistribución de una cantidad fija de neuronas que maximiza el número de multiplicaciones posibles. Aunque este caso no es representativo de situaciones comunes en la práctica, permite establecer que, en los casos analizados, el número máximo de multiplicaciones se obtiene con dos capas ocultas.

Sea la ecuación (3.1) una función dependiente de la cantidad de capas ocultas L y el número de unidades por capa M, denotada como $f(L, M)$:

$$f(L, M) = (L-1)M^2 + M(n_x + n_u), \tag{A.1}$$

$$L, M, n_x, n_u \in \mathbb{N}, \tag{A.2}$$

donde n_x es el número de entradas y n_u la cantidad de salidas de la red neuronal. Además, las capas de la red cumplen la condición $M \geq n_x$. Suponiendo que la cantidad total de unidades neuronales es una constante ρ, se tiene:

$$L \cdot M = \rho, \quad \rho \in \mathbb{N}. \tag{A.3}$$

Sustituyendo $M = \frac{\rho}{L}$ y $\alpha = n_x + n_u$ en (A.1), se obtiene:

$$f(L) = (L-1)\left(\frac{\rho}{L}\right)^2 + \frac{\rho}{L}\alpha \tag{A.4}$$

$$f(L) = \frac{(L-1)\rho^2}{L^2} + \frac{\rho\alpha}{L} \tag{A.5}$$

$$f(L) = \frac{\rho(\rho+\alpha)}{L} - \frac{\rho^2}{L^2}. \tag{A.6}$$

Para encontrar un punto crítico de $f(L)$, se iguala $f'(L) = 0$:

$$f'(L) = 0 \tag{A.7}$$

$$\frac{-\rho(\rho+\alpha)}{L^2} + \frac{2\rho^2}{L^3} = 0 \tag{A.8}$$

$$2\rho^2 - L\rho(\rho+\alpha) = 0 \tag{A.9}$$

$$L = \frac{2\rho}{\rho+\alpha}. \tag{A.10}$$

Dado que L, α y ρ son números naturales, es posible concluir que el punto crítico se encuentra en $L = 1$ ó $L = 2$. Para determinar si este punto crítico corresponde a un máximo, se evalúa la segunda derivada:

$$f''(L) < 0 \tag{A.11}$$

$$\frac{2\rho(\rho+\alpha)}{L^3} - \frac{6\rho^2}{L^4} < 0. \tag{A.12}$$

Sustituyendo $L = \frac{2\rho}{\rho+\alpha}$ en la expresión (A.12):

$$\frac{2\rho(\rho+\alpha)}{\left(\frac{2\rho}{(\rho+\alpha)}\right)^3} - \frac{6\rho^2}{\left(\frac{2\rho}{\rho+\alpha}\right)^4} < 0 \tag{A.13}$$

$$\frac{(\rho+\alpha)^4}{(2\rho)^2} - \frac{3(\rho+\alpha)^4}{2(2\rho)^2} < 0 \tag{A.14}$$

$$-\frac{(\rho+\alpha)^4}{2(2\rho)^2} < 0. \tag{A.15}$$

La desigualdad (A.15) se satisface siempre, confirmando que el punto crítico es un máximo. Finalmente, para determinar la ubicación del máximo, se analiza la diferencia entre $f(1)$ y $f(2)$:

$$f(2) - f(1) > 0 \tag{A.16}$$

$$\frac{\rho(\rho+\alpha)}{2} - \frac{\rho^2}{4} - \rho(\rho+\alpha) + \rho^2 > 0 \tag{A.17}$$

$$\rho(\rho - 2\alpha) > 0. \tag{A.18}$$

Como $\rho \in \mathbb{N}$, se cumple la desigualdad si y sólo si $\rho > 2\alpha$. Por lo tanto, si la red neuronal en sus capas ocultas tiene más neuronas que el doble de la suma de sus entradas y salidas, la configuración que maximiza las operaciones de multiplicación, y por ende el uso de recursos, es aquella en la que las neuronas están distribuidas en dos capas ocultas.

Es importante destacar que este análisis responde a un caso teórico que no suele encontrarse en aplicaciones prácticas. Sin embargo, proporciona un marco conceptual útil para interpretar los resultados presentados en la Sección 4.3 y explorar las implicaciones de la redistribución de unidades neuronales.

Apéndice B

Estimación preliminar del uso de bloques DSP para una red neuronal

Al implementar una red neuronal en una FPGA, los bloques DSP son recursos esenciales para realizar las operaciones de multiplicación y acumulación necesarias durante la inferencia. Dado que estos bloques son un recurso limitado, es importante contar con un método preliminar para estimar su uso en la etapa de diseño. Una estimación adecuada del uso de bloques DSP facilita la selección del factor de reutilización, reduciendo la cantidad de síntesis iterativas necesarias para identificar el factor que permita al diseño cumplir con sus requerimientos. Este anexo presenta una fórmula propuesta para estimar el uso de bloques DSP a partir de información de la red neuronal, como sus hiperparámetros, el nivel de cuantización y el factor de reutilización. Asimismo, se analiza el comportamiento de la estimación en comparación con los resultados obtenidos mediante la síntesis de hardware.

La estimación del uso de bloques DSP se basa en el número de multiplicaciones que realiza una red neuronal durante la inferencia. Para una red de n_x entradas, n_u salidas, L capas ocultas, M unidades por capa, el total de multiplicaciones necesarias se presenta en la ecuación (3.1). Expandiendo las multiplicaciones de la entrada y salida en la ecuación (3.1), se tiene:

$$N_{\text{mult}} = (L-1)M^2 + n_x M + n_u M. \tag{B.1}$$

Luego, considerando que cada multiplicación se realiza utilizando exactamente un bloque DSP, el efecto del factor de reutilización R puede incorporarse como:

$$N_{\text{DSP}} = (L-1)\left\lceil \frac{M^2}{R} \right\rceil + \left\lceil \frac{n_x M}{R} \right\rceil + \left\lceil \frac{n_u M}{R} \right\rceil. \tag{B.2}$$

Para determinar los bloques DSP necesarios por multiplicador, se define la función $D(W)$, donde W es el nivel de cuantización de la red neuronal. Esta función depende de las características de la FPGA y de las decisiones de optimización realizadas por la herramienta de síntesis. En general, $D(W)$ considera un parámetro T, que representa al número máximo de bits que soporta la herramienta de síntesis con un único bloque DSP.

Un ejemplo de esta función se presenta como:

$$D(W) = \begin{cases} 0 & W \leq 10 \\ \lceil \frac{W}{T} \rceil & W > 10 \end{cases}, \tag{B.3}$$

donde $D(W)$ refleja las características de las FPGAs utilizadas en este trabajo. Así, cuando se usan niveles de cuantización de 10 bits o menos, no se emplean bloques DSP, como se muestra en el Capítulo 4. En el caso de la tarjeta ZCU104 utilizada para las exploraciones de diseño descritas en el mismo capítulo, el parámetro T toma un valor de 27, que como se observó en la Sección 4.6, es el máximo nivel de cuantización antes de que la herramienta comience a utilizar dos DSP por multiplicador.

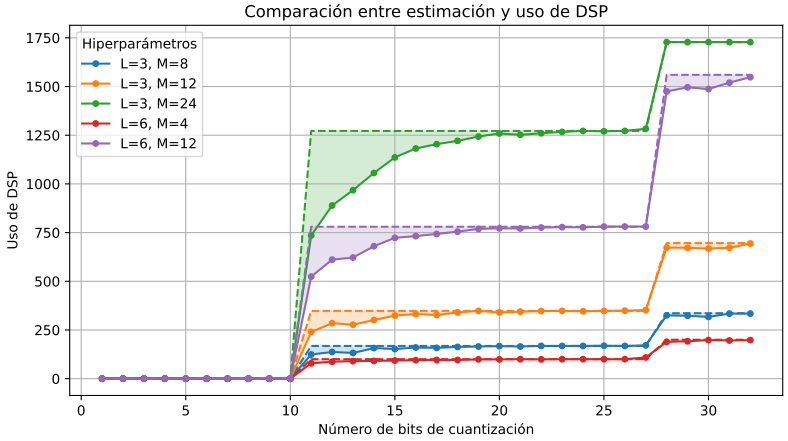


Figura B.1: Comparación entre la estimación (línea segmentada) y el uso real (línea continua) de DSP en función del nivel de cuantización W.

Incorporando el efecto de la cuantización sobre el uso de bloques DSP, la estimación queda expresada como:

$$N_{\text{DSP}} = D(W) \cdot \left((L-1) \left\lceil \frac{M^2}{R} \right\rceil + \left\lceil \frac{n_x M}{R} \right\rceil + \left\lceil \frac{n_u M}{R} \right\rceil \right). \tag{B.4}$$

Para mejorar la precisión de la estimación, se incorpora una restricción basada en el número total de bloques DSP disponibles en la FPGA, denotado como $N_{\text{AvailableDSP}}$. La ecuación ajustada queda de la siguiente forma:

$$N_{\text{DSP}} = \min(D(W) \cdot \left((L-1) \left\lceil \frac{M^2}{R} \right\rceil + \left\lceil \frac{n_x M}{R} \right\rceil + \left\lceil \frac{n_u M}{R} \right\rceil \right), N_{\text{AvailableDSP}}). \tag{B.5}$$

La precisión de la estimación se analiza en la Figura B.1, donde se observa que la estimación captura correctamente los quiebres en el uso de bloques DSP. Además, en todas las redes se identifica una mayor imprecisión en la estimación para niveles de cuantización de 11 bits, la cual disminuye conforme aumenta el nivel de cuantización. Este patrón se repite, aunque con una menor magnitud, para valores superiores a 28 bits.

Por otro lado, la Figura B.2 muestra el comportamiento de la estimación para niveles de cuantización de 16 y 32 bits, evidenciando que la mayor imprecisión ocurre cuando el factor de reutilización es igual a uno. A medida aumenta el factor de reutilización. la estimación se vuelve más precisa, lo que se observa consistentemente en ambos niveles de cuantización y en las cuatro redes estudiadas.

En base a lo observado, se concluye que la ecuación (B.5) proporciona una estimación adecuada del uso de bloques DSP. Aunque la estimación no logra capturar la totalidad de las optimizaciones realizadas por la herramienta, llevando a una sobreestimación del uso de DSPs, si es útil para guiar preliminarmente la elección del factor de reutilización, reduciendo la necesidad de realizar síntesis iterativas para conocer la factibilidad de la implementación de una red neuronal.

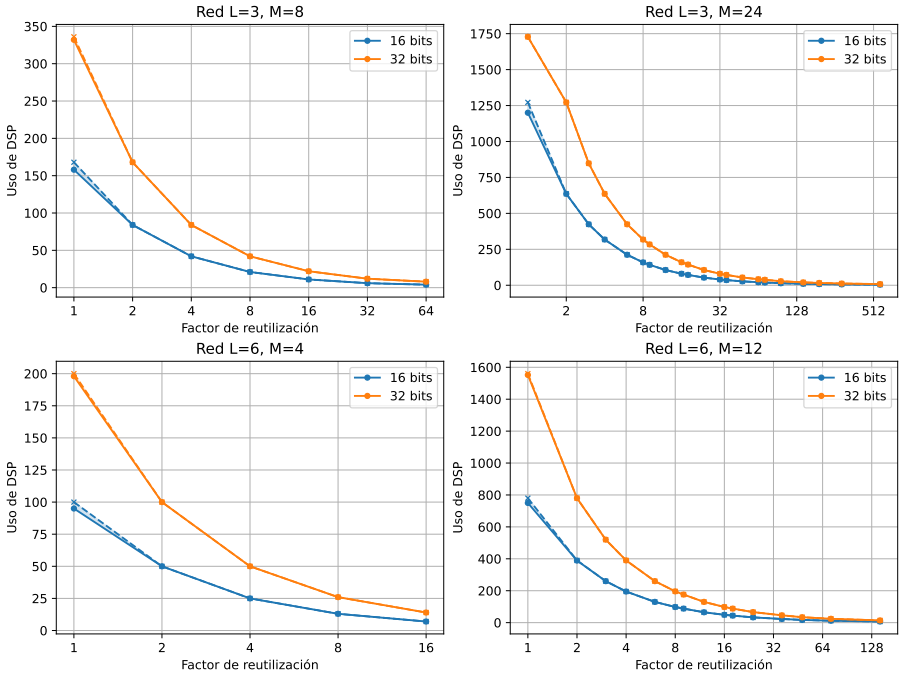


Figura B.2: Comparación entre la estimación (línea segmentada) y el uso real (línea continua) de DSP en función del factor de reutilización R.

Apéndice C

Extensión del análisis de redes con diferente arquitectura de hardware

A continuación, se presenta el análisis de dos casos adicionales relacionados con la exploración realizada en la Sección 4.7. El objetivo de este análisis es evaluar el impacto del factor de reutilización como técnica para reducir el uso de recursos y analizar su efecto en la latencia de inferencia de las redes neuronales, bajo diferentes niveles de cuantización y combinaciones de hiperparámetros.

El factor de reutilización en HLS4ML determina cuántas veces se emplea un mismo multiplicador dentro de una capa, lo que define indirectamente el nivel de paralelismo espacial de la arquitectura de hardware.

Debido a que los análisis presentados a continuación complementan los realizados en la Sección 4.7, estos utilizan los mismos parámetros definidos en la Tabla 4.7. Las redes consideradas son configuraciones más pequeñas, específicamente de 3×8 y 6×4, en comparación con las estudiadas en la Sección 4.7. Para ambas redes, se realiza la síntesis lógica explorando factores de reutilización en el rango $[1, M^2]$.

C.1 Análisis de reportes para una red neuronal de 3×8

Para la red de 3×8, la Figura C.1 ilustra el uso de recursos y la latencia a medida que aumenta el factor de reutilización, desde 1 hasta 64, que corresponde a la cantidad de operaciones de multiplicación realizadas en las capas ocultas de la red. Los datos representados en la Figura C.1 están tabulados en la Tabla C.1. A partir de los reportes de esta red, se observa lo siguiente:

- Para los bloques DSP, la Figura C.1 muestra una relación inversamente proporcional entre el factor de reutilización y el uso de bloques DSP. Al igual que como se observó en el Capítulo 4, la red con 8 bits no utilizó DSPs, mientras que con 32 bits fue necesario usar dos DSPs por multiplicación. Esto también puede corroborarse mediante la Tabla C.1. Asimismo, aunque es esperable que con un factor de reutilización de uno se empleen 168 DSPs para 16 bits y 336 con 32 bits, se registra un uso de DSPs ligeramente menor al esperado en ambos casos, probablemente debido a optimizaciones automáticas de Vitis HLS y Vivado.

- En relación a las LUTs, aunque en la Figura C.1 no se aprecia con claridad una tendencia en los casos de 16 y 32 bits, la Tabla C.1 permite observar que, en general, existe una ligera disminución en el uso de LUTs al emplear un factor de reutilización mayor que uno, aunque esta reducción no se mantiene de forma consistente con factores de reutilización superiores. Un ejemplo notable es el caso de R=32 con 32 bits, que, a diferencia de todos los demás casos, presenta un ligero aumento en el uso de LUTs en comparación con R=1. Este comportamiento podría guardar relación con la probable optimización aplicada con R=1, donde se lograron ahorros en bloques DSP a costa del uso de LUTs para implementar multiplicadores, lo que incrementó su utilización en ese caso particular de factor de reutilización.

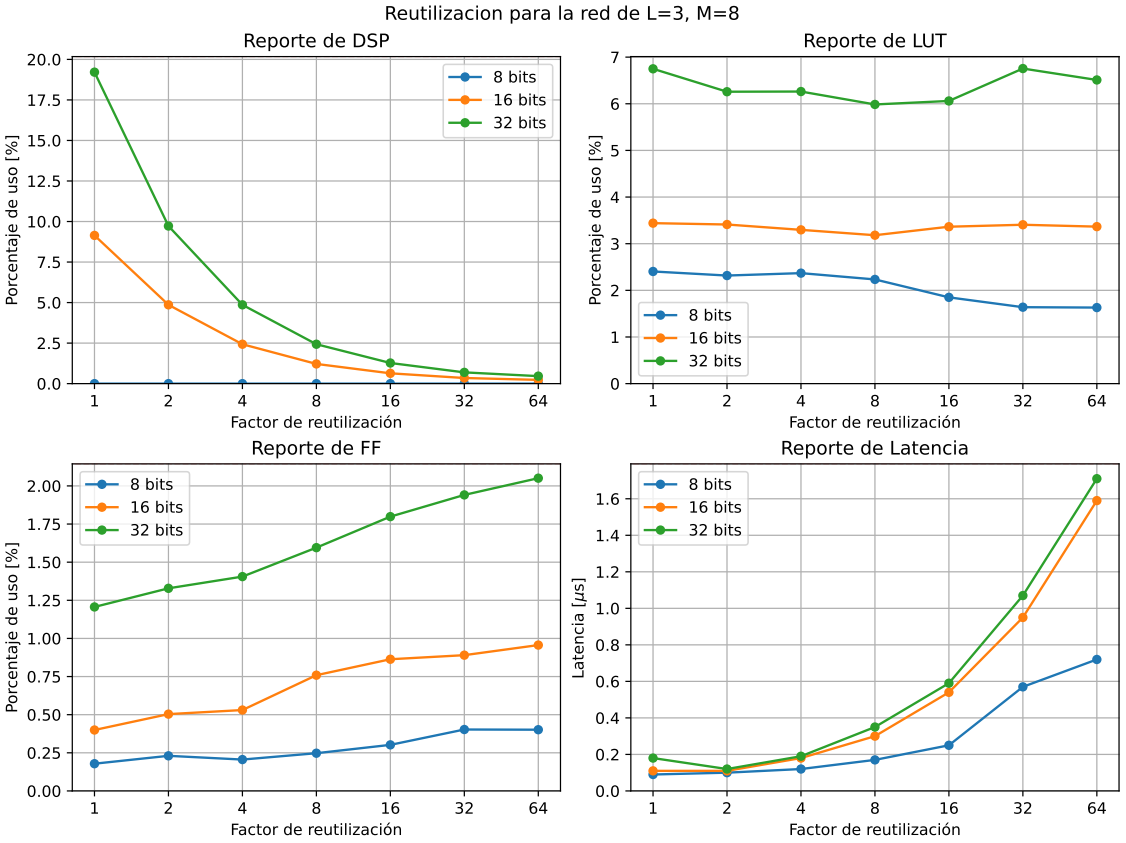


Figura C.1: Reportes de implementación de la red de 3×8 con diferentes factores de reutilización.

En el caso de 8 bits, la Figura C.1 muestra una ligera reducción en el uso de LUTs a medida que aumenta el factor de reutilización, siendo más consistente a partir de R=8. Esta reducción puede explicarse por el uso de LUTs para implementar los multiplicadores necesarios en la red neuronal, los que van reduciéndose al incrementar la reutilización. En este sentido, a partir de la Tabla C.1, se puede inferir que, en el caso de 8 bits, aproximadamente un tercio de las LUTs utilizadas con R=1 se destinaba a implementar multiplicadores, ya que, al reducir el número de multiplicadores de 168 a 4, se logró una disminución del 32.2 % en el uso de LUTs.

- Analizando los FFs, en la Figura C.1 se observa un aumento de su uso a medida que incrementa el factor de reutilización en los tres niveles de cuantización considerados. Este aumento tiende a ser más pronunciado en el caso de 32 bits de cuantización. Aunque la variación absoluta es mayor con 32 bits, la Tabla C.1 indica que los casos con 8 y 16 bits de cuantización presentan un mayor incremento porcentual en el uso de FFs con respecto al factor de reutilización. No obstante, en comparación con los demás recursos, el uso de FFs resulta secundario, ya que no presenta un crecimiento tan significativo como el de las LUTs o los DSPs.

- Finalmente, revisando la latencia, la Figura C.1 muestra que, a partir de un factor de reutilización igual a 4, en los casos de 16 y 32 bits, existe una relación aproximadamente proporcional entre la latencia y el factor de reutilización. En el caso de 8 bits, aunque se observa un crecimiento conjunto de ambas variables, el aumento en la latencia es menos pronunciado, incluso rompiendo la tendencia con el factor de reutilización más alto considerado. La Tabla C.1 permite analizar con mayor precisión lo ocurrido al aumentar el factor de reutilización de uno a dos: en el caso de 32 bits, la latencia disminuyó 6 ciclos; con 16 bits, se mantuvo constante; y con 8 bits, sólo se incrementó en un ciclo de reloj. Como se discutió en la Sección 4.7, este comportamiento se debe a la planificación de las operaciones realizada por Vitis HLS.

Tabla C.1: Reportes de implementación para la red de 3×8 bajo diferentes factores de reutilización y niveles de cuantización (8, 16 y 32 bits), mostrando el impacto en el uso de DSPs, LUTs, FFs y latencia. Las columnas coloreadas indican las variaciones porcentuales de cada caso de reutilización, donde el color verde indica una reducción y el rojo un aumento respecto al caso con factor de reutilización $R = 1$.

W bits	***R***	**DSPs**	**Δ DSPs** % de R=1	**LUTs**	**Δ LUTs** % de R=1	**FFs**	**Δ FFs** % de R=1	**Latencia** [ns]	**Δ Latencia** % de R=1
8	1	0 (0.0 %)	—	5540 (2.4 %)	—	823 (0.2 %)	—	90	—
8	2	0 (0.0 %)	0.0 %	5339 (2.3 %)	-3.6 %	1061 (0.2 %)	28.9 %	100	11.1 %
8	4	0 (0.0 %)	0.0 %	5458 (2.4 %)	-1.5 %	948 (0.2 %)	15.2 %	120	33.3 %
8	8	0 (0.0 %)	0.0 %	5147 (2.2 %)	-7.1 %	1140 (0.2 %)	38.5 %	170	88.9 %
8	16	0 (0.0 %)	0.0 %	4262 (1.8 %)	-23.1 %	1392 (0.3 %)	69.1 %	250	177.8 %
8	32	0 (0.0 %)	0.0 %	3776 (1.6 %)	-31.8 %	1856 (0.4 %)	125.5 %	570	533.3 %
8	64	0 (0.0 %)	0.0 %	3757 (1.6 %)	-32.2 %	1850 (0.4 %)	124.8 %	720	700.0 %
16	1	158 (9.1 %)	—	7927 (3.4 %)	—	1841 (0.4 %)	—	110	—
16	2	84 (4.9 %)	-46.8 %	7860 (3.4 %)	-0.8 %	2321 (0.5 %)	26.1 %	110	0.0 %
16	4	42 (2.4 %)	-73.4 %	7596 (3.3 %)	-4.2 %	2444 (0.5 %)	32.8 %	180	63.6 %
16	8	21 (1.2 %)	-86.7 %	7331 (3.2 %)	-7.5 %	3497 (0.8 %)	90.0 %	300	172.7 %
16	16	11 (0.6 %)	-93.0 %	7751 (3.4 %)	-2.2 %	3979 (0.9 %)	116.1 %	540	390.9 %
16	32	6 (0.3 %)	-96.2 %	7848 (3.4 %)	-1.0 %	4102 (0.9 %)	122.8 %	950	763.6 %
16	64	4 (0.2 %)	-97.5 %	7755 (3.4 %)	-2.2 %	4407 (1.0 %)	139.4 %	1590	1345.5 %
32	1	332 (19.2 %)	—	15551 (6.7 %)	—	5559 (1.2 %)	—	180	—
32	2	168 (9.7 %)	-49.4 %	14419 (6.3 %)	-7.3 %	6122 (1.3 %)	10.1 %	120	-33.3 %
32	4	84 (4.9 %)	-74.7 %	14428 (6.3 %)	-7.2 %	6476 (1.4 %)	16.5 %	190	5.6 %
32	8	42 (2.4 %)	-87.3 %	13790 (6.0 %)	-11.3 %	7351 (1.6 %)	32.2 %	350	94.4 %
32	16	22 (1.3 %)	-93.4 %	13963 (6.1 %)	-10.2 %	8288 (1.8 %)	49.1 %	590	227.8 %
32	32	12 (0.7 %)	-96.4 %	15564 (6.8 %)	0.1 %	8944 (1.9 %)	60.9 %	1070	494.4 %
32	64	8 (0.5 %)	-97.6 %	15000 (6.5 %)	-3.5 %	9450 (2.1 %)	70.0 %	1710	850.0 %

C.2 Análisis de reportes para una red neuronal de 6×4

Para la red neuronal de 6×4, los reportes de síntesis lógica están graficados en la Fig. C.2. Dentro de las redes analizadas, esta red presenta el rango más limitado de factores de reutilización, que abarca desde 1 hasta 16. La Tabla C.2 presenta las variaciones porcentuales en el uso de recursos y la latencia para cada factor de reutilización, tomando como referencia el caso con factor de reutilización igual a uno. Del análisis conjunto de la Figura C.2 y la Tabla C.2, se destacan las siguientes observaciones, organizadas según el tipo de recurso y latencia:

- Para el uso de DSPs, la Figura C.2 muestra una relación inversamente proporcional entre el factor de reutilización y el uso de DSPs. Considerando un factor de reutilización igual a uno, se espera preliminarmente que la red, que debe realizar 100 multiplicaciones, requiera 100 DSPs con 16 bits y 200 con 32 bits. Sin embargo, como ya se observó con la red anterior, según la Tabla C.2 el uso de DSPs es inferior al esperado para 16 y 32 bits. Además, a diferencia de la red con tres capas ocultas analizada previamente, esta red neuronal requiere al menos siete multiplicadores: seis para las capas ocultas y uno para la capa de salida. Esto se traduce en un mínimo de siete bloques DSP para 16 bits y 14 para 32 bits, reflejando las restricciones impuestas por la precisión numérica y la estructura de la red.
- En cuanto al uso de LUTs, la Figura C.2 no muestra una tendencia clara para los casos de 16 y 32 bits, lo que revela que el impacto del factor de reutilización sobre las LUTs en estos niveles de cuantización es mínimo. Por otro lado, en el caso de 8 bits, se observa una ligera reducción en el uso de LUTs a partir de un factor de reutilización igual a ocho. Desde la Tabla C.2, se nota un pequeño aumento en el uso de LUTs con un factor de reutilización igual a dos en el caso de 8 bits, aunque su magnitud es despreciable.
- Al analizar el uso de FFs, la Figura C.2 muestra una tendencia al alza a medida que aumenta el factor de reutilización. No obstante, incluso en el caso de mayor uso para esta red, este representa apenas el 1.4 % del total disponible, como se confirma en la Tabla C.2, lo que sugiere que este recurso no es crítico para la implementación de esta red.

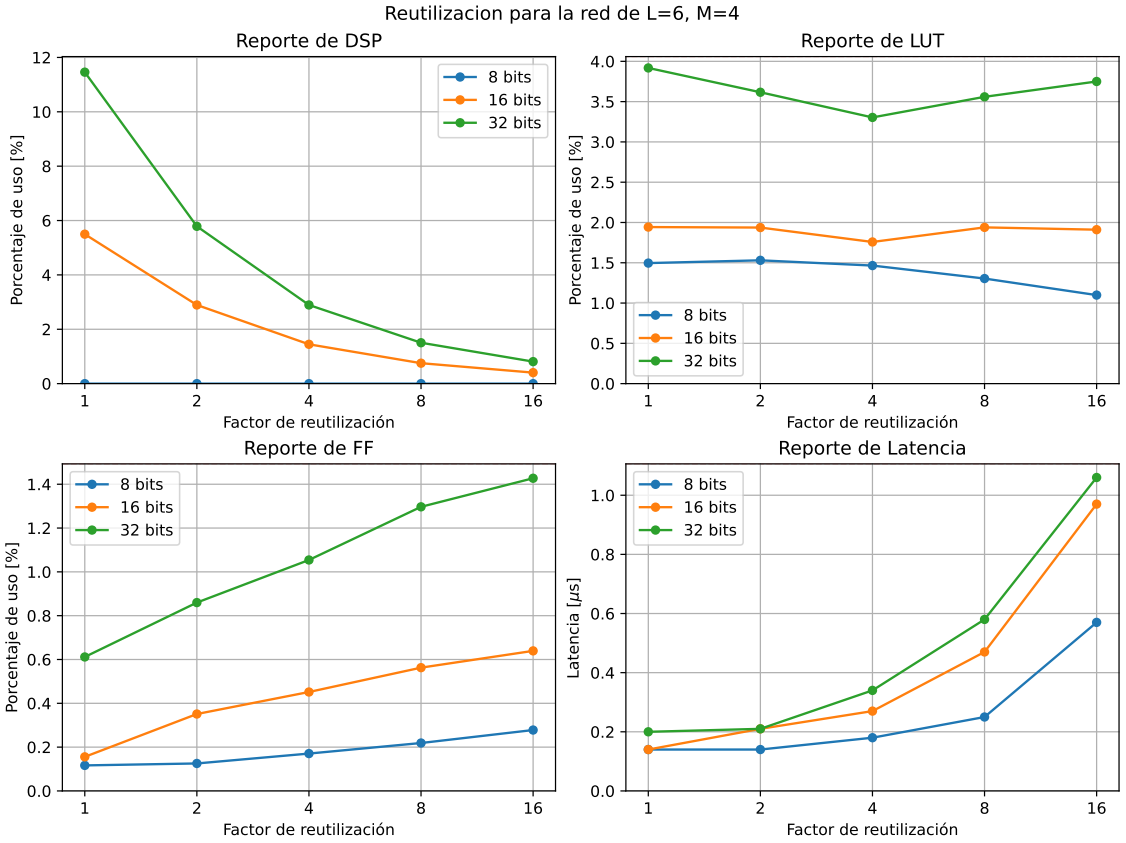


Figura C.2: Reportes de implementación de la red de 6×4 con diferentes factores de reutilización.

- Por último, en relación con la latencia, la Figura C.2 confirma la correlación entre esta y el factor de reutilización. El mayor valor de latencia, 1.06 [μs], se observa con R=16 en 32 bits. Además, tanto la Figura C.2 como la Tabla C.2 muestran que la latencia mínima es idéntica para los casos de 8 y 16 bits, lo que concuerda con los hallazgos de la Sección 4.6, donde se identificó que la latencia de esta red permanece constante entre 8 y 16 bits cuando el factor de reutilización es igual a uno.

C.3 Observaciones generales

Los casos adicionales presentados en este anexo refuerzan algunas ideas planteadas en el Capítulo 4, que se resumen a continuación:

- Las herramientas de síntesis utilizadas tienden a reemplazar DSPs usando LUTs con mayor frecuencia cuando la reutilización es igual a uno.
- El factor de reutilización establece un *trade-off* principalmente entre el uso de DSPs y la latencia.
- Aunque el uso FFs también se ve afectado por el factor de reutilización, en los casos estudiados este incremento no es significativo para la implementación.
- El uso de LUTs se ve influenciado por la reutilización en la medida en que estas se utilizan para implementar multiplicadores, como en los casos de 8 bits. Sin embargo, incluso cuando la totalidad de los multiplicadores de la red se implementan con LUTs, la efectividad del factor de reutilización para reducir su uso es menor en relación con los casos en que se emplean DSPs.

Tabla C.2: Reportes de implementación para la red de 6×4 bajo diferentes factores de reutilización y niveles de cuantización (8, 16 y 32 bits), mostrando el impacto en el uso de DSPs, LUTs, FFs y latencia. Las columnas coloreadas representan las variaciones porcentuales respecto al caso con factor de reutilización $R = 1$. En todos los casos, el color verde indica una reducción y el rojo un aumento.

W bits	***R***	**DSPs**	**Δ DSPs** % de R=1	**LUTs**	**Δ LUTs** % de R=1	**FFs**	**Δ FFs** % de R=1	**Latencia** [ns]	**Δ Latencia** % de R=1
8	1	0 (0.0 %)	—	3447 (1.5 %)	—	537 (0.1 %)	—	140	—
8	2	0 (0.0 %)	0.0 %	3526 (1.5 %)	2.3 %	578 (0.1 %)	7.6 %	140	0.0 %
8	4	0 (0.0 %)	0.0 %	3376 (1.5 %)	-2.1 %	786 (0.2 %)	46.4 %	180	28.6 %
8	8	0 (0.0 %)	0.0 %	3005 (1.3 %)	-12.8 %	1008 (0.2 %)	87.7 %	250	78.6 %
8	16	0 (0.0 %)	0.0 %	2532 (1.1 %)	-26.5 %	1281 (0.3 %)	138.5 %	570	307.1 %
16	1	95 (5.5 %)	—	4477 (1.9 %)	—	717 (0.2 %)	—	140	—
16	2	50 (2.9 %)	-47.4 %	4463 (1.9 %)	-0.3 %	1618 (0.4 %)	125.7 %	210	50.0 %
16	4	25 (1.4 %)	-73.7 %	4051 (1.8 %)	-9.5 %	2081 (0.5 %)	190.2 %	270	92.9 %
16	8	13 (0.8 %)	-86.3 %	4468 (1.9 %)	-0.2 %	2593 (0.6 %)	261.6 %	470	235.7 %
16	16	7 (0.4 %)	-92.6 %	4402 (1.9 %)	-1.7 %	2946 (0.6 %)	310.9 %	970	592.9 %
32	1	198 (11.5 %)	—	9027 (3.9 %)	—	2818 (0.6 %)	—	200	—
32	2	100 (5.8 %)	-49.5 %	8332 (3.6 %)	-7.7 %	3962 (0.9 %)	40.6 %	210	5.0 %
32	4	50 (2.9 %)	-74.7 %	7612 (3.3 %)	-15.7 %	4858 (1.1 %)	72.4 %	340	70.0 %
32	8	26 (1.5 %)	-86.9 %	8201 (3.6 %)	-9.2 %	5976 (1.3 %)	112.1 %	580	190.0 %
32	16	14 (0.8 %)	-92.9 %	8640 (3.8 %)	-4.3 %	6577 (1.4 %)	133.4 %	1060	430.0 %

- Varios de los casos presentados mostraron que un factor de reutilización de 2 ofrece un mejor equilibrio entre latencia y uso de recursos, superando al caso de paralelismo completo (R=1) al mantener o incluso reducir la latencia.

Apéndice D

Multiplications-Error Ranking Score (MERS)

A continuación, se presenta la métrica MERS (*Multiplications-Error Ranking Score*), diseñada para ordenar configuraciones de hiperparámetros de redes neuronales según su balance entre error de inferencia y número de multiplicaciones. El error de inferencia, medido como la raíz del error cuadrático medio (RMSE), es calculado sobre los datos de prueba al finalizar el entrenamiento e indica la precisión alcanzada por la red. Por otro lado, el número de multiplicaciones se relaciona directamente con el uso de recursos al implementar la red en FPGA, lo que se discute en la Sección 4.3. Balancear ambos criterios permite seleccionar configuraciones de redes adecuadas, en base a los datos de la exploración realizada y las necesidades de la aplicación, facilitando la etapa inicial del flujo descrito en el Capítulo 3.

Sea $\mathcal{A}$ el conjunto de redes neuronales evaluadas, definido como $\mathcal{A} = \{\mathcal{N}_1, \ldots, \mathcal{N}_n\}$, donde n es el número total de redes evaluadas y cada $\mathcal{N}_i$ representa una red con una configuración específica de hiperparámetros dentro del espacio de diseño explorado. Para cada red $\mathcal{N}_i$, se define ρ_i como su número total de multiplicaciones y ε_i como el RMSE obtenido al evaluarla. Para que ρ y ε sean comparables es necesario normalizarlos, siendo $\bar{\rho_i}$ y $\bar{\varepsilon_i}$ los valores normalizados para cada red, calculados de la siguiente forma:

$$\bar{\rho_i} = \frac{\rho_i - \min_{j\in\{1,\ldots,n\}}(\rho_j)}{\max_{j\in\{1,\ldots,n\}}(\rho_j) - \min_{j\in\{1,\ldots,n\}}(\rho_j)}, \tag{D.1}$$

$$\bar{\varepsilon_i} = \frac{\varepsilon_i - \min_{j\in\{1,\ldots,n\}}(\varepsilon_j)}{\max_{j\in\{1,\ldots,n\}}(\varepsilon_j) - \min_{j\in\{1,\ldots,n\}}(\varepsilon_j)}. \tag{D.2}$$

Luego, el puntaje asociado a la red $\mathcal{N}_i$ dentro del conjunto $\mathcal{A}$ se obtiene calculando:

$$\text{MERS}_i = 1 - (\omega \cdot \bar{\varepsilon_i} + (1-\omega) \cdot \bar{\rho_i}). \tag{D.3}$$

donde $\omega \in [0, 1]$ es un peso asignado por el usuario que determina la prioridad entre error y multiplicaciones. Así, un valor de ω más cercano a 0 implica una mayor prioridad por un el número de multiplicaciones, aumentando el puntaje de las redes con menos multiplicaciones, mientras que un valor cercano a 1 le da más importancia al error de la red. De este modo, la métrica MERS asigna un puntaje entre 0 y 1 a cada red evaluada, ponderando su cantidad relativa de multiplicaciones y el error relativo alcanzado. Por lo tanto, el puntaje MERS depende tanto del espacio de búsqueda empleado, como del peso ω, permitiendo priorizar configuraciones de hiperparámetros que se ajusten mejor a las necesidades de la aplicación.

Bibliografía

[1] A. Alessio and A. Bemporad, *A Survey on Explicit Model Predictive Control*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 345–369. [En línea]. Disponible en: https://doi.org/10.1007/978-3-642-01094-1_29

[2] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[3] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S0005109801001741

[4] A. Ravera, A. Oliveri, M. Lodi, A. Bemporad, W. P. M. H. Heemels *et al.*, "Co-Design of a Controller and Its Digital Implementation: The MOBY-DIC2 Toolbox for Embedded Model Predictive Control," *IEEE Transactions on Control Systems Technology*, vol. 31, no. 6, pp. 2871–2878, 2023.

[5] M. Jeong, S. Fuchs, and J. Biela, "When FPGAs meet regionless explicit MPC: An implementation of long-horizon linear MPC for power electronic systems," in *IECON 2020 the 46th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2020, pp. 3085–3092.

[6] M. Mönnigmann and M. Kastsian, "Fast explicit MPC with multiway trees," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 1356–1361, 2011, 18th IFAC World Congress. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S1474667016437985

[7] A. Oliveri and M. Storace, "Hardware-in-the-loop simulations of circuit architectures for the computation of exact and approximate explicit MPC control functions," in *2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*. IEEE, 2012, pp. 380–383.

[8] J. Holaza, B. Takács, and M. Kvasnica, "Synthesis of simple explicit MPC optimizers by function approximation," in *2013 International Conference on Process Control (PC)*, 2013, pp. 377–382.

[9] D. Ingole, M. Kvasnica, H. De Silva, and J. Gustafson, "Reducing Memory Footprints in Explicit Model Predictive Control using Universal Numbers," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 11 595–11 600, 2017, 20th IFAC World Congress. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S2405896317321018

[10] A. Norouzi, H. Heidarifar, H. Borhan, M. Shahbakhti, and C. R. Koch, "Integrating Machine Learning and Model Predictive Control for automotive applications: A review and future directions," *Engineering Applications of Artificial Intelligence*, vol. 120, p. 105878, 2023. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S0952197623000623

[11] A. Mesbah, K. P. Wabersich, A. P. Schoellig, M. N. Zeilinger, S. Lucia *et al.*, "Fusion of Machine Learning and MPC under Uncertainty: What Advances Are on the Horizon?" in *2022 American Control Conference (ACC)*, 2022, pp. 342–357.

[12] B. Karg and S. Lucia, "Efficient Representation and Approximation of Model Predictive Control Laws via Deep Learning," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, sep 2020. [En línea]. Disponible en: https://doi.org/10.1109%2Ftcyb.2020.2999556

[13] ——, "Stability and feasibility of neural network-based controllers via output range analysis," in *2020 59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 4947–4954.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [En línea]. Disponible en: https://www.deeplearningbook.org

[15] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[16] G. Verma, Y. Gupta, A. M. Malik, and B. Chapman, "Performance Evaluation of Deep Learning Compilers for Edge Inference," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, pp. 858–865.

[17] I. McInerney, G. A. Constantinides, and E. C. Kerrigan, "A Survey of the Implementation of Linear Model Predictive Control on FPGAs," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 381–387, 2018, 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S2405896318327216

[18] C. Jugade, D. Ingole, D. N. Sonawane, M. Kvasnica, and J. Gustafson, "A Memory Efficient FPGA Implementation of Offset-Free Explicit Model Predictive Controller," *IEEE Transactions on Control Systems Technology*, vol. 30, no. 6, pp. 2646–2657, 2022.

[19] S. Lucia, D. Navarro, B. Karg, H. Sarnago, and O. Lucía, "Deep Learning-Based Model Predictive Control for Resonant Power Converters," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 1, pp. 409–420, 2021.

[20] K. J. Chan, J. A. Paulson, and A. Mesbah, "Deep Learning-based Approximate Nonlinear Model Predictive Control with Offset-free Tracking for Embedded Applications," in *2021 American Control Conference (ACC)*, 2021, pp. 3475–3481.

[21] F. Dong, X. Li, K. You, and S. Song, "Standoff Tracking Using DNN-Based MPC With Implementation on FPGA," *IEEE Transactions on Control Systems Technology*, vol. 31, no. 5, pp. 1998–2010, 2023.

[22] N. R. Mohanty, C. Jugade, V. Patne, D. Ingole, and D. Sonawane, "FPGA Implementation of Low Complexity Nonlinear Model Predictive Control Using Deep Learning Approach," in *2022 Eighth Indian Control Conference (ICC)*, 2022, pp. 325–330.

[23] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[DL] A Survey of FPGA-Based Neural Network Inference Accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, mar 2019. [En línea]. Disponible en: https://doi.org/10.1145/3289185

[24] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," *JINST*, vol. 13, no. 07, p. P07027, 2018.

[25] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien *et al.*, "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, dec 2018. [En línea]. Disponible en: https://doi.org/10.1145/3242897

[26] The MathWorks Inc., "Deep Learning HDL Toolbox Version 23.2 (R2023b)," Natick, Massachusetts, United States, 2023. [En línea]. Disponible en: https://www.mathworks.com/help/deeplearning/index.html

[27] A. Cortés, "Implementación de aceleradores de cómputo en lógica reconfigurable para aplicaciones de control predictivo por modelo utilizando síntesis de alto nivel," Master's Thesis, Universidad Técnica Federico Santa María, Valparaíso, Chile, Marzo 2024.

[28] C. Gonzalez, H. Asadi, L. Kooijman, and C. P. Lim, "Neural Networks for Fast Optimisation in Model Predictive Control: A Review," 2023.

[29] H. Ferreau, S. Almér, R. Verschueren, M. Diehl, D. Frick *et al.*, "Embedded Optimization Methods for Industrial Automatic Control," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 13 194–13 209, 2017, 20th IFAC World Congress. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S2405896317325764

[30] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[31] Y. Li, S. E. Li, X. Jia, S. Zeng, and Y. Wang, "FPGA accelerated model predictive control for autonomous driving," *Journal of Intelligent and Connected Vehicles*, vol. 5, no. 2, pp. 63–71, 2022.

[32] F. Chollet, *Deep learning with Python*. Manning, 2018.

[33] C. N. Coelho Jr, A. Kuusela, S. Li, H. Zhuang, J. Ngadiuba *et al.*, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature Machine Intelligence*, vol. 3, no. 8, pp. 675–686, 2021.

[34] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for Convolutional Networks," in *2009 International Conference on Field Programmable Logic and Applications*, 2009, pp. 32–37.

[35] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 161–170. [En línea]. Disponible en: https://doi.org/10.1145/2684746.2689060

[36] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, p. 1–12, jun 2017. [En línea]. Disponible en: https://doi.org/10.1145/3140659.3080246

[37] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Research Whitepaper*, vol. 2, no. 11, pp. 1–4, 2015.

[38] C. Baskin, N. Liss, E. Zheltonozhskii, A. M. Bronstein, and A. Mendelson, "Streaming Architecture for Large-Scale Quantized Neural Networks on an FPGA-Based Dataflow Platform," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, pp. 162–169.

[39] M. Ghiglione, V. Serra, A. Raoofy, G. Dax, C. Trinitis *et al.*, "Survey of frameworks for inference of neural networks in space data systems," *Data Systems in Aerospace (DASIA). Eurospace*, 2022.

[40] Xilinx, "Vitis-AI," 2024, Último acceso: 2024-10-26. [En línea]. Disponible en: https://github.com/Xilinx/Vitis-AI

[41] Microchip-Vectorblox, "VectorBlox-SDK," 2024, accessed: 2024-10-26. [En línea]. Disponible en: https://github.com/Microchip-Vectorblox/VectorBlox-SDK

[42] H. S. Khan and K. Kauhaniemi, "Design and FPGA-in-loop based validation of predictive hierarchical control for islanded AC microgrid," *Engineering Science and Technology, an International Journal*, vol. 48, p. 101557, 2023. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S2215098623002355

[43] AMD, "Zynq 7000 SoCs," accessed 29-08-2024. [En línea]. Disponible en: https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-7000.html

[44] S. Lucia, D. Navarro, O. Lucía, P. Zometa, and R. Findeisen, "Optimized FPGA Implementation of Model Predictive Control for Embedded Systems Using High-Level Synthesis Tool," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 137–145, 2018.

[45] J. Escárate, R. López, A. L. Cedeño, J. C. Agüero, C. Silva, and G. Carvajal, "FPGA Implementation of ADMM for Model Predictive Control in a DC/AC Converter," in *2022 IEEE International Conference on Automation/XXV Congress of the Chilean Association of Automatic Control (ICA-ACCA)*, 2022, pp. 1–6.

[46] M. Jeong, M. Schoen, and J. Biela, "When FPGAs Meet ADMM with High-level Synthesis (HLS): A Real-time Implementation of Long-horizon MPC for Power Electronic Systems," in *2023 11th International Conference on Power Electronics and ECCE Asia (ICPE 2023 - ECCE Asia)*, 2023, pp. 1704–1711.

[47] A. Cortes, J. C. Aguero, C. Silva, and G. Carvajal, "Leveraging High Level Synthesis for the Design of Hardware Accelerators for Model Predictive Control," in *2024 Argentine Conference on Electronics (CAE)*, 2024, pp. 16–21.

[48] J. Nubert, J. Köhler, V. Berenz, F. Allgöwer, and S. Trimpe, "Safe and Fast Tracking on a Robot Manipulator: Robust MPC and Neural Network Control," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3050–3057, 2020.

[49] R. Winqvist, A. Venkitaraman, and B. Wahlberg, "On Training and Evaluation of Neural Network Approaches for Model Predictive Control," 2020.

[50] M. Abu-Ali, F. Berkel, M. Manderla, S. Reimann, R. Kennel, and M. Abdelrahem, "Deep Learning-Based Long-Horizon MPC: Robust, High Performing, and Computationally Efficient Control for PMSM Drives," *IEEE Transactions on Power Electronics*, vol. 37, no. 10, pp. 12 486–12 501, 2022.

[51] M. Li, H. Xu, Y. Wei, and A. Deng, "Deep Neural Network-Based Linear Quadratic Programming for Vehicle Path Tracking," in *2023 5th International Conference on Robotics, Intelligent Control and Artificial Intelligence (RICAI)*, 2023, pp. 446–451.

[52] H. Hose, J. Köhler, M. N. Zeilinger, and S. Trimpe, "Approximate non-linear model predictive control with safety-augmented neural networks," 2023.

[53] D. Wang, Z. J. Shen, X. Yin, S. Tang, X. Liu *et al.*, "Model predictive control using artificial neural network for power converters," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 4, pp. 3689–3699, 2022.

[54] E. Maddalena, C. da S. Moraes, G. Waltrich, and C. Jones, "A Neural Network Architecture to Learn Explicit MPC Controllers from Data," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 11 362–11 367, 2020, 21st IFAC World Congress. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S2405896320308442

[55] I. S. Mohamed, S. Rovetta, T. D. Do, T. Dragicević, and A. A. Z. Diab, "A Neural-Network-Based Model Predictive Control of Three-Phase Inverter With an Output *LC* Filter," *IEEE Access*, vol. 7, pp. 124 737–124 749, 2019.

[56] F. Simonetti, A. D'Innocenzo, and C. Cecati, "Neural Network Model-Predictive Control for CHB Converters With FPGA Implementation," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 9, pp. 9691–9702, 2023.

[57] F. Chollet *et al.*, "Keras," 2015. [En línea]. Disponible en: https://keras.io

[58] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5784–5789, 2018.

[59] The MathWorks, Inc., *MATLAB Python Environment Documentation*, 2024, accessed: 2024-11-12. [En línea]. Disponible en: https://www.mathworks.com/help/matlab/ref/matlab.pyclient.pythonenvironment.html

[60] FastML Team, "fastmachinelearning/hls4ml," 2023. [En línea]. Disponible en: https://github.com/fastmachinelearning/hls4ml

[61] AMD, *Vivado Design Suite User Guide: High-Level Synthesis*, 2017, accessed: 2025-01-16. [En línea]. Disponible en: https://docs.amd.com/v/u/2017.4-English/ug902-vivado-high-level-synthesis

[62] ——, *Vitis High-Level Synthesis User Guide (UG1399)*, 2022, Último acceso: 2024-11-02. [En línea]. Disponible en: https://docs.amd.com/r/2022.2-English/ug1399-vitis-hls

[63] Xilinx, Inc., *UltraScale Architecture DSP Slice User Guide*, Xilinx, 2022, accessed: 2024-12-10. [En línea]. Disponible en: https://0x04.net/~mwk/xidocs/ug/ug579-ultrascale-dsp.pdf

[64] J. B. Mare and J. A. De Doná, "Solution of the input-constrained LQR problem using dynamic programming," *Systems & Control Letters*, vol. 56, no. 5, pp. 342–348, 2007. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S016769110600185X

[65] A. Bemporad, "Hybrid Toolbox - User's Guide," 2004. [En línea]. Disponible en: http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox

[66] ARM, *AMBA AXI and ACE Protocol Specification*, 2013, Último acceso: 2025-01-16. [En línea]. Disponible en: https://developer.arm.com/documentation/ihi0022/e

[67] J. D. Escárate, "Implementación en FPGA de control predictivo basado en modelos en un convertidor DC/AC conectado a un filtro LC en DERs," Master's Thesis, Universidad Técnica Federico Santa María, Valparaíso, Chile, Abril 2023.