

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Diseño e implementación de algoritmos de control predictivo por modelo en FPGA utilizando representación de punto fijo

Tesis de Grado presentada por

Francisca Nazarena Donoso Bastías

como requisito parcial para optar al título de

Ingeniería Civil Electrónica

y al grado de

Magíster en Ciencias de la Ingeniería Electrónica

Supervisor

Dr. Gonzalo Carvajal

Comisión

Dr. César Silva

Dr. Miguel Figueroa

Septiembre de 2026

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: Diseño e implementación de algoritmos de control predictivo por modelo en FPGA utilizando representación de punto fijo

Nombre del candidato(a): Francisca Nazarena Donoso Bastías

Carrera / Grado: Magíster en Ciencias de la Ingeniería Electrónica

Campus: Casa Central **Departamento:** Departamento de Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Gonzalo Carvajal, 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 (marcar una opción)

☒ 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 **(marcar una opción)**:

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

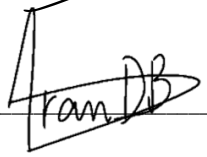
Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

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

Fecha: 27/10/2026 **Firma:** 

Estudiante o Candidato(a):

Fecha: 27/10/2026 **Firma:** 

AGRADECIMIENTOS

Quiero comenzar agradeciendo a mi familia por su apoyo incondicional y por acompañarme durante todos estos años. También a mis amigos, tanto a quienes estuvieron desde el inicio de mi proceso académico como a quienes se fueron sumando en el camino, incluidos aquellos que conocí a través de las distintas actividades extracurriculares en las que participé. Cada uno, de distintas formas y en diferentes etapas, hizo que este proceso fuera mucho más llevadero y significativo.

Agradezco especialmente al profesor Gonzalo Carvajal por su paciencia, dedicación y apoyo durante el desarrollo de este trabajo, así como a los antiguos y actuales miembros del Laboratorio Turing por su disposición, ayuda y compañía durante estos años.

Finalmente, quiero agradecer a mi pareja por acompañarme y darme ánimo a lo largo de todo este proceso.

A todos quienes fueron parte de este camino, muchas gracias.

Francisca Nazarena Donoso Bastías

El desarrollo de este trabajo contó con el apoyo de los siguientes proyectos ANID:

- FONDECYT 1261452
- AC3E CIA250006

RESUMEN

La implementación de Control Predictivo por Modelo (Model Predictive Control, MPC) en Field-Programmable Gate Arrays (FPGAs) permite aprovechar arquitecturas especializadas para reducir el tiempo de cálculo requerido por la resolución en línea del problema de optimización. El uso de herramientas de Síntesis de Alto Nivel (High Level Synthesis, HLS) facilita el desarrollo de estas arquitecturas a partir de descripciones algorítmicas en C/C++, mientras que la aritmética de punto fijo permite adaptar la representación numérica a los requerimientos de la aplicación y reducir el costo de implementación. Sin embargo, seleccionar una representación en punto fijo adecuada y trasladarla entre entornos de diseño no es un proceso directo. No existe un estándar ampliamente adoptado que unifique el comportamiento entre herramientas, por lo que un mismo formato numérico puede producir resultados diferentes si no se consideran las configuraciones asociadas a las operaciones.

Este trabajo presenta un flujo para el diseño e implementación en FPGA de controladores MPC lineales con formulación implícita utilizando representaciones de punto fijo. El flujo parte de un modelo funcional previamente validado en punto flotante y permite explorar representaciones numéricas en MATLAB antes de avanzar hacia su implementación en C/C++ y Vitis HLS. A lo largo del desarrollo se comparan las distintas implementaciones para mantener trazabilidad sobre las diferencias numéricas introducidas y evaluar si la representación seleccionada satisface los criterios definidos para la aplicación. El flujo se estudia inicialmente mediante una operación de producto punto y posteriormente se aplica a dos casos de control cuyos problemas de optimización se resuelven mediante ADMM: un motor de corriente continua y un sistema de recursos energéticos distribuidos (DER).

Los resultados muestran que las representaciones seleccionadas mantuvieron el comportamiento en lazo cerrado dentro de los criterios definidos y permitieron reducir el uso de recursos y la latencia respecto de las implementaciones equivalentes en punto flotante. El análisis del traslado entre herramientas mostró además que la estructura de cálculo y la configuración de las operaciones influyen en la correspondencia numérica entre implementaciones, por lo que estos aspectos deben considerarse además del formato de punto fijo. En conjunto, los resultados respaldan la selección de la representación numérica durante etapas tempranas del diseño, considerando simultáneamente la precisión numérica, el desempeño en lazo cerrado y el costo de implementación.

ABSTRACT

The implementation of Model Predictive Control (MPC) on Field-Programmable Gate Arrays (FPGAs) enables the use of specialized architectures to reduce the computation time required to solve the optimization problem online. High-Level Synthesis (HLS) tools facilitate the development of these architectures from algorithmic descriptions in C/C++, while fixed-point arithmetic allows the numerical representation to be tailored to the requirements of the application and can reduce implementation cost. However, selecting an appropriate fixed-point representation and transferring it between design environments is not straightforward. No widely adopted standard defines consistent fixed-point behavior across tools, and therefore the same numerical format may produce different results if the arithmetic configuration of the operations is not taken into account.

This thesis presents a design and implementation flow for linear MPC controllers with an implicit formulation using fixed-point arithmetic. The flow starts from a functional model previously validated in floating-point and allows different numerical representations to be explored in MATLAB before being transferred to C/C++ and Vitis HLS. The implementations are compared throughout the development process to track the numerical differences introduced at each stage and to determine whether the selected representation satisfies the criteria defined for the application. The flow is first evaluated using a dot-product operation and is then applied to two control case studies whose optimization problems are solved using ADMM: a direct current (DC) motor and a distributed energy resources (DER) system.

The results show that the selected fixed-point representations preserved the closed-loop behavior within the defined acceptance criteria while reducing resource usage and latency compared with the corresponding floating-point implementations. The analysis of the transfer between tools also showed that both the computational structure and the arithmetic configuration of the operations affect the numerical agreement between implementations. These aspects must therefore be considered in addition to the fixed-point format itself. Overall, the results support selecting the numerical representation during the early stages of the design process while considering numerical accuracy, closed-loop performance, and implementation cost together.

ÍNDICE

AGRADECIMIENTOS **I**

RESUMEN **III**

ABSTRACT **IV**

ÍNDICE DE FIGURAS **VIII**

ÍNDICE DE TABLAS **XI**

ACRÓNIMOS **XII**

1. INTRODUCCIÓN **1**

1.1. Motivación y contexto 1
1.2. Formulación del problema 3
1.3. Hipótesis 4
1.4. Alcances y contribuciones 4
1.5. Estructura de la tesis 5

2. ANTECEDENTES **6**

2.1. Control predictivo por modelo 6
2.1.1. Método de multiplicadores de dirección alternante 8
2.2. Implementación de MPC en FPGA 8
2.3. Diseño basado en modelos 9
2.4. Precisión numérica en el diseño de controladores MPC 11
2.4.1. Selección de formatos de punto fijo 11
2.4.2. Implementaciones de MPC en punto fijo mediante herramientas HLS . . . 11
2.4.3. Verificación numérica y evaluación en lazo cerrado 12
2.5. Discusión 13

3. FUNDAMENTOS DE REPRESENTACIÓN EN PUNTO FIJO **14**

3.1. Notación para representación en punto fijo 14
3.2. Rango, precisión y representación binaria 15
3.3. Cuantización y desbordamiento 18
3.3.1. Cuantización 18

3.3.1.1. Truncamiento . . . 19
3.3.1.2. Redondeo . . . 19
3.3.1.3. Efectos de truncamiento y redondeo . . . 20
3.3.2. Desbordamiento . . . 21
3.3.2.1. Envolvimiento (wrap o módulo) . . . 21
3.3.2.2. Saturación . . . 22
3.4. Aritmética en punto fijo . . . 22
3.4.1. Suma y resta en punto fijo . . . 24
3.4.2. Multiplicación . . . 28
3.4.3. División . . . 30
3.5. Discusión . . . 32

4. IMPLEMENTACIÓN DE OPERACIONES EN PUNTO FIJO EN FPGA 33
4.1. Flujo propuesto para diseño e implementación de algoritmos en FPGA . . . 33
4.1.1. Entorno MATLAB . . . 35
4.1.2. Entorno AMD . . . 35
4.1.3. Entorno MATLAB + AMD . . . 36
4.2. Exploración de representación numérica en MATLAB . . . 37
4.2.1. Representación de punto fijo mediante objetos `fi` . . . 37
4.2.2. Consideraciones sobre la estructura algorítmica . . . 40
4.2.3. Fixed-Point Converter y análisis de rangos . . . 43
4.3. Implementación en punto fijo mediante Vitis HLS . . . 46
4.3.1. Tipos `ap_fixed` y equivalencia con la notación Q . . . 46
4.4. Caso de estudio: Implementación de operación de producto punto . . . 48
4.4.1. Archivos de entrada para la conversión . . . 49
4.4.2. Exploración de formatos de punto fijo en MATLAB . . . 50
4.4.3. Traslado de la representación numérica a Vitis HLS . . . 54
4.4.4. Validación en hardware mediante FPGA-in-the-Loop . . . 57
4.4.5. Restricciones prácticas para equivalencia entre entornos . . . 58
4.5. Observaciones y discusión del capítulo . . . 60

5. IMPLEMENTACIÓN Y VERIFICACIÓN DE LAZOS MPC 62
5.1. Metodología de evaluación . . . 62
5.1.1. Modelo y formulación común de los lazos de control . . . 62
5.1.2. Preparación del entorno y etapas de evaluación . . . 63
5.1.3. Representación numérica común . . . 64
5.1.4. Métricas y criterios de comparación . . . 64
5.2. Motor de corriente continua . . . 65
5.2.1. Modelo, controlador y criterios de evaluación . . . 66
5.2.2. Exploración de formatos de punto fijo en MATLAB . . . 67
5.2.3. Implementación y validación en Vitis HLS . . . 69
5.2.4. Validación en hardware mediante FPGA-in-the-Loop . . . 76
5.3. Sistema de recursos energéticos distribuidos . . . 76
5.3.1. Modelo, controlador y escenario de evaluación . . . 76

5.3.2. Exploración de formatos de punto fijo en MATLAB 78
5.3.3. Traslado de la representación numérica a Vitis HLS 80

6. CONCLUSIONES Y TRABAJO FUTURO 85
6.1. Conclusiones 85
6.2. Trabajo futuro 86

BIBLIOGRAFÍA 88

A. APLICACIÓN COMPLEMENTARIA DEL FLUJO DE DISEÑO A UN CONTROLADOR PI 91
A.1. Sistema de referencia 91
A.2. Exploración de la representación en punto fijo 92
A.3. Traslado y verificación en HLS 92
A.4. Validación experimental en FPGA 94
A.5. Extensión hacia una implementación ASIC 94
A.6. Relación con el flujo utilizado en esta tesis 95

B. PUBLICACIONES GENERADAS 96

Índice de figuras

3.1. Generalización de la Notación $Q_{m,n}$ para el formato de punto fijo. 15

3.2. Grilla de cuantización para un formato de punto fijo $Q_{1,2}$. 16

3.3. Grilla de cuantización para el formato $Q_{1,2}$. En la parte superior (azul) se ilustran los índices enteros correspondientes a cada nivel. La variable x mapea directamente sobre el índice 1, correspondiente al valor real 0,25, mientras que el ejemplo y se ilustra sobre el índice -3, correspondiente a $-0{,}75$. 17

3.4. Mapeo de la grilla de cuantización para el formato $Q_{4,4}$. En la parte superior (azul) se ilustran los índices en el plano entero, delimitados por el rango de representación $[-128; 127]$ y en la parte inferior se muestran los valores de la grilla de cuantización. El valor $x = 3{,}27$ cae entre los índices 52 y 53, requiriendo una aproximación. 19

3.5. Representación del desbordamiento para $x = 9{,}5$ en formato $Q_{4,4}$. El valor escalado (152) excede el límite superior de la grilla (127). La saturación lo ancla al máximo permitido (127), mientras que el envolvimiento proyecta el valor hacia el espectro negativo del rango (debido a la interpretación del bit de signo en complemento a dos). 22

4.1. Flujo general de herramientas propuesto para la implementación y validación de operaciones en punto fijo. 34

4.2. Representación del histograma de rango dinámico para los valores registrados de la variable X. La figura reproduce de forma ordenada el tipo de visualización entregada por la herramienta, con el propósito de describir cómo se organiza la información en intervalos de magnitud asociados a potencias de 2. 44

4.3. Error absoluto entre el producto punto en punto flotante y la versión en punto fijo obtenida con las opciones por defecto de *Fixed-Point Converter*. 51

4.4. Histogramas utilizados para seleccionar formatos de punto fijo en el producto punto. 52

4.5. Error absoluto entre el producto punto en punto flotante y la versión en punto fijo con formatos ajustados a partir del análisis de rangos. 52

4.6. Histograma de salida para el formato $U_{5,7}$, obtenido al reducir los bits fraccionales respecto del formato $U_{5,10}$. 53

4.7. Error absoluto entre el producto punto en punto flotante y la versión en punto fijo con formatos ajustados a partir del análisis de rangos, con menos bits para acumulador. 53

4.8. Error absoluto entre el producto punto en punto flotante y la versión en punto fijo con formatos, métodos de cuantización y desbordamiento explícitos. 54

4.9. Comparación de errores entre la referencia en punto flotante, la implementación en punto fijo de MATLAB y la implementación en punto fijo de Vitis HLS. 56

4.10. Esquema de validación mediante FPGA-in-the-Loop para el producto punto. 57

4.11. Comparación entre la *golden reference* y la salida obtenida mediante FPGA-in-the-Loop. 58

4.12. Comparación entre la referencia en punto flotante y la implementación en punto fijo obtenida a partir de la descripción alternativa del producto punto. 59
4.13. Errores relativos obtenidos al comparar la descripción alternativa del producto punto en MATLAB y Vitis HLS respecto de la referencia en flotante. 60

5.1. Esquema general de simulación en lazo cerrado utilizado en los casos de estudio. El vector de perturbaciones medibles $\mathbf{d}_k$ se utiliza únicamente en el sistema DER. . . 63
5.2. Simulación del controlador MPC en lazo cerrado utilizando representación en punto flotante. 68
5.3. Comparación de la respuesta del motor en punto flotante y punto fijo con formato $Q_{4,7}$ para el primer escenario de simulación. 69
5.4. Comparación de la respuesta del motor con formato $Q_{4,7}$ para el segundo escenario de simulación. 70
5.5. Comparación de la respuesta del motor con formato $Q_{6,8}$ para el segundo escenario de simulación. 71
5.6. Comparación en lazo cerrado entre las implementaciones en punto flotante y punto fijo de MATLAB y la simulación en C de Vitis HLS para el formato $Q_{6,8}$. 72
5.7. Uso de recursos y latencia del controlador con directivas de optimización para distintas longitudes de palabra, manteniendo $m = 6$. 73
5.8. Error absoluto de la actuación respecto de la referencia en punto flotante para distintas longitudes de palabra, manteniendo $m = 6$. 74
5.9. Uso de recursos y latencia para distintas distribuciones de bits enteros y fraccionales, manteniendo fija la longitud total en $W = 32$ bits. 75
5.10. Variación de la resistencia de carga aplicada durante la simulación del sistema DER. Los cambios introducidos en $t = 0,1$ s, $t = 0,3$ s y $t = 0,4$ s permiten evaluar la respuesta del controlador en distintos puntos de operación. 79
5.11. Respuesta temporal del sistema DER en lazo cerrado utilizando el controlador MPC en punto flotante. Se muestran los voltajes del condensador y sus referencias, las corrientes del inductor del filtro y las actuaciones calculadas por el controlador. 80
5.12. Trayectorias de las actuaciones y de las corrientes del filtro en los planos dq para la simulación en punto flotante. El gráfico izquierdo presenta la trayectoria de actuación, el límite circular y su aproximación politópica, mientras que el gráfico derecho muestra la corriente del filtro respecto del límite nominal. 81
5.13. Comparación de la respuesta temporal del sistema DER en punto flotante y punto fijo con formato $Q_{16,11}$. Se muestran los voltajes del condensador y sus referencias, las corrientes del inductor del filtro y las actuaciones calculadas por el controlador. . . 82
5.14. Comparación de las trayectorias de actuación y corriente para las implementaciones en punto flotante y punto fijo con formato $Q_{16,11}$. El gráfico izquierdo muestra el límite de actuación y su aproximación politópica, mientras que el gráfico derecho presenta la trayectoria de corriente respecto del límite nominal. 83
5.15. Comparación de las actuaciones u_d y u_q en punto flotante y punto fijo con formato $Q_{16,11}$, junto con sus errores absolutos. En cada caso se indican el error absoluto máximo y el error absoluto medio respecto de la referencia en punto flotante. 83
5.16. Comparación de las actuaciones u_d y u_q obtenidas con el modelo de referencia en punto flotante, la implementación en punto fijo de MATLAB y la implementación en punto fijo de Vitis HLS para el formato $Q_{16,11}$. Los gráficos inferiores presentan el error absoluto de Vitis HLS respecto de la referencia en punto flotante y respecto de la implementación en punto fijo de MATLAB. 84

A.1. Arquitectura general del sistema de control utilizado para el convertidor 3L-FCC. . 92

A.2. Respuesta simulada en lazo cerrado del convertidor con los controladores en punto flotante y $Q_{8,14}$ 93
A.3. Diagrama de regularización necesaria para implementar el controlador PI en Bambu HLS. 93
A.4. Montaje utilizado para la validación experimental. 94
A.5. Voltaje de salida medido para escalones de referencia de 0,6 V. 94

Índice de tablas

3.1. Comparación de errores generados por distintos métodos de truncamiento y redondeo al almacenar valores en un formato $Q_{4,4}$. 21

4.1. Modos de cuantización de `fimath`. 38
4.2. Distribución de la variable X en intervalos de magnitud definidos por potencias de dos. 45
4.3. Modos de cuantización seleccionados de `ap_fixed` en Vitis HLS. 47
4.4. Modos de desbordamiento seleccionados de `ap_fixed` en Vitis HLS. 47
4.5. Correspondencias prácticas entre configuraciones de MATLAB y Vitis HLS. . . . 48
4.6. Configuraciones consideradas para la conversión del producto punto en MATLAB. . 54
4.7. Uso de recursos para los tres casos evaluados en la implementación del producto punto. 56

5.1. Resultados posteriores a la implementación en Vitis HLS para distintas representaciones numéricas del controlador MPC, sin directivas manuales de optimización. 71
5.2. Resultados de las implementaciones del controlador MPC con directivas de optimización para distintas representaciones numéricas y dispositivos objetivo. 75
5.3. Errores absolutos de las actuaciones u_d y u_q para las implementaciones evaluadas. . 81
5.4. Uso de recursos y desempeño de las implementaciones del controlador DER. Las representaciones en punto fijo corresponden a resultados posteriores a la implementación, mientras que los valores en punto flotante corresponden a la síntesis. Entre paréntesis se indica el porcentaje respecto de los recursos disponibles en la FPGA Artix-7 100T. 84

ACRÓNIMOS

ADMM	: Alternating Direction Method of Multipliers
ASIC	: Application-Specific Integrated Circuit
BRAM	: Block Random Access Memory
CPU	: Central Processing Unit
DC	: Direct Current
DER	: Distributed Energy Resources
DSP	: Digital Signal Processor
FCS-MPC	: Finite Control Set Model Predictive Control
FF	: Flip-Flop
FGM	: Fast Gradient Method
FIL	: FPGA-in-the-loop
FPGA	: Field Programmable Gate Array
HDL	: Hardware Description Language
HIL	: Hardware-in-the-loop
HLS	: High Level Synthesis
LUT	: Look-Up Table
MBD	: Model-Based Design
MIL	: Model-in-the-loop
MPC	: Model Predictive Control
PI	: Proporcional-Integral
PIL	: Processor-in-the-loop
QP	: Quadratic Programming
RTL	: Register Transfer Level
SIL	: Software-in-the-loop
WNS	: Worst Negative Slack
ZOH	: Zero-Order Hold

Capítulo 1

INTRODUCCIÓN

En este capítulo se presenta el contexto y la motivación que sustentan el desarrollo de la tesis, con énfasis en los desafíos asociados a la implementación de controladores MPC en FPGA mediante representaciones de punto fijo y herramientas de alto nivel. Además, se presenta la formulación del problema de investigación, junto con la hipótesis, los alcances y las contribuciones del trabajo. Finalmente, se describe la organización general de la tesis.

1.1. Motivación y contexto

El Control Predictivo por Modelo (*Model Predictive Control*, MPC) es una estrategia de control multivariable que utiliza un modelo matemático de la planta para anticipar el comportamiento futuro del sistema y determinar una secuencia de acciones de control de acuerdo con un criterio de desempeño predefinido. De manera intuitiva, en cada instante de muestreo el controlador mide o estima el estado del sistema, proyecta su evolución sobre un horizonte de predicción de muestras futuras, resuelve un problema de optimización considerando la información disponible, aplica la primera acción de control obtenida y repite el procedimiento en el siguiente instante con una nueva medición o estimación del estado. Una de las principales ventajas de MPC es que permite incorporar explícitamente restricciones sobre estados, entradas y salidas, lo que lo hace atractivo para sistemas sujetos a límites físicos u operacionales [1, 2].

Cuando se emplea un modelo lineal de la planta, una función de costo cuadrática y restricciones lineales, el problema de optimización asociado al cálculo de la acción de control suele formularse como un problema de Programación Cuadrática (*Quadratic Programming*, QP). En implementaciones implícitas de MPC lineal, un nuevo problema QP debe formularse y resolverse en línea en cada período de muestreo. La carga computacional asociada a la resolución en línea del problema QP históricamente limitó las primeras aplicaciones industriales del MPC a sistemas con dinámicas relativamente lentas, como los procesos químicos y petroquímicos, donde los períodos de muestreo del orden de decenas de segundos o minutos entregaban tiempo suficiente para completar el cálculo antes de aplicar la siguiente actuación [3–5].

Durante los últimos años, el desarrollo de algoritmos de optimización para problemas QP, junto con los avances en las capacidades de cómputo de plataformas embebidas, ha favorecido la implementación de MPC en sistemas con requerimientos de tiempo cada vez más exigentes [6, 7]. Distintas familias de *solvers* emplean de forma repetitiva operaciones de álgebra lineal que admiten distintos grados de ejecución paralela, lo que resulta particularmente adecuado para arquitecturas especializadas implementadas en *Field-Programmable Gate Arrays* (FPGAs) [6, 8, 9]. En particular, el Método de Multiplicadores de Dirección Alternante (*Alternating Direction Method of Multipliers*, ADMM) emplea multiplicaciones matriz-vector, operaciones entre vectores y comparaciones elemento a elemento, dentro

de las cuales existen cálculos que pueden ejecutarse de manera independiente [6,10]. El aprovechamiento de este paralelismo ha permitido obtener implementaciones de MPC en FPGA con tiempos de resolución del orden de microsegundos y, en algunos casos, tasas de control superiores a 1 MHz [6, 7].

Tradicionalmente, el desarrollo de sistemas basados en FPGA se ha realizado mediante lenguajes de descripción de hardware (*Hardware Description Language*, HDL) a nivel de transferencia de registros (*Register Transfer Level*, RTL), donde el diseñador define de forma explícita las unidades funcionales, los registros, el movimiento de datos y la coordinación de las operaciones a lo largo de los ciclos de reloj. Este nivel de descripción proporciona un control detallado sobre la arquitectura y permite definir el grado de paralelización espacial y temporal, además de los anchos de datos y operadores. En implementaciones de métodos de optimización para MPC, el grado de paralelización y la elección de la representación numérica se han utilizado para establecer compromisos entre latencia, utilización de recursos y precisión de los resultados [6, 11]. Sin embargo, la descripción directa a nivel RTL suele demandar tiempos de desarrollo elevados, especialmente cuando se requiere explorar distintas alternativas de implementación [12,13]. Además, la especialización de la arquitectura para las dimensiones y características de una aplicación puede dificultar su reutilización o adaptación a otros problemas.

Más recientemente, el desarrollo y la progresiva madurez de las herramientas de Síntesis de Alto Nivel (*High Level Synthesis*, HLS) han favorecido su uso como una alternativa al diseño directo en RTL para implementar hardware especializado desde un nivel de abstracción mayor [12]. El diseño mediante HLS parte de una descripción algorítmica en lenguajes como C/C++, sobre la cual pueden especificarse directivas de síntesis para orientar la arquitectura generada. Estas directivas permiten modificar el grado y la forma de paralelización mediante técnicas como el uso de *pipeline* y el desenrollado de lazos, además de controlar la reutilización de unidades funcionales y la organización de memoria, entre otras propiedades. A partir de la descripción algorítmica y de las directivas de síntesis especificadas, la herramienta de HLS genera una descripción RTL sintetizable [12, 14]. La posibilidad de evaluar distintas alternativas de diseño a partir de una misma descripción de alto nivel ha sido aprovechada en implementaciones en FPGA de *solvers* para MPC. Distintos trabajos han estudiado cómo la organización de las operaciones y las directivas de síntesis afectan la latencia y la utilización de recursos de la arquitectura resultante [13,15]. No obstante, en estas implementaciones la representación numérica se mantiene en punto flotante durante la exploración, por lo que el análisis se concentra en las decisiones sobre la arquitectura del hardware sin incorporar simultáneamente las decisiones asociadas a una implementación en punto fijo, lo cual puede modificar el comportamiento numérico del algoritmo.

Las herramientas HLS suelen ofrecer tipos de datos en punto fijo con anchos configurables, lo que permite adaptar la representación numérica a los requerimientos de la aplicación. El uso de punto fijo con anchos de palabra reducidos puede disminuir la utilización de recursos y la latencia de las operaciones respecto de implementaciones equivalentes en punto flotante [6, 8, 16, 17]. Sin embargo, la conversión desde una descripción en punto flotante hacia punto fijo no constituye un paso directo, ya que requiere seleccionar los anchos de palabra y la resolución de las variables junto con el comportamiento de las operaciones ante la cuantización, el desbordamiento y la precisión utilizada en los resultados intermedios [6, 16]. Además, estas decisiones no se encuentran unificadas por un estándar ampliamente adoptado entre herramientas [18, 19], y definir una implementación en punto fijo adecuada continúa siendo un problema activo de diseño [20].

El uso de representación numérica de precisión reducida suele tener un efecto relevante en el desempeño de un sistema de control. En particular, en un algoritmo de MPC implícito, los errores introducidos por una implementación en punto fijo pueden propagarse entre las iteraciones del *solver* y modificar la actuación calculada [6, 21]. En lazo cerrado, una diferencia en la actuación respecto de la implementación de referencia modifica el estado alcanzado por la planta y, con ello, las condiciones de los problemas de optimización resueltos en los períodos de muestreo posteriores. Por esta razón, los efectos de la representación numérica deben evaluarse sobre el sistema en lazo cerrado y no únicamente sobre el *solver* o módulos aislados, considerando su impacto sobre el desempeño del controlador y el cumplimiento de restricciones. Sin embargo, en la literatura sobre implementaciones de MPC mediante HLS, estos aspectos no se evalúan de forma conjunta, ya que el análisis se concentra principalmente en la arquitectura resultante o en los efectos numéricos de la precisión reducida por separado [13, 15, 21].

La evaluación progresiva de los efectos introducidos por la representación numérica puede apoyarse en principios del Diseño Basado en Modelos (*Model-Based Design*, MBD), donde un modelo ejecutable se mantiene como referencia a medida que el diseño avanza hacia representaciones más próximas a la implementación en hardware, permitiendo comparar su comportamiento entre las distintas etapas del desarrollo [22, 23]. En este marco, MATLAB dispone de herramientas para partir de una descripción funcional en punto flotante, analizar los rangos alcanzados durante las simulaciones, explorar representaciones en punto fijo y extender la validación hasta etapas de generación de código y pruebas sobre hardware. Este enfoque de validación progresiva también se ha utilizado en implementaciones de controladores MPC en FPGA para apoyar la selección y validación de representaciones en punto fijo [21, 24]. Cuando las distintas etapas del desarrollo se realizan dentro de un mismo entorno como MATLAB, la automatización simplifica el trabajo del diseñador al abstraer detalles asociados a la implementación en hardware, pero esa misma abstracción reduce el control directo sobre las decisiones que determinan la arquitectura generada.

Cuando se busca combinar la exploración numérica realizada sobre un modelo de alto nivel con un mayor control sobre la arquitectura mediante herramientas HLS dedicadas, el desarrollo pasa a involucrar distintos entornos que operan a diferentes niveles de abstracción. Por ejemplo, MATLAB puede utilizarse para modelar el algoritmo, explorar representaciones en punto fijo y evaluar su comportamiento, mientras que Vitis HLS permite generar una descripción RTL sintetizable a partir de una implementación en C/C++. Al pasar entre estos entornos, una configuración en punto fijo validada sobre el modelo de alto nivel no necesariamente reproduce de forma automática el mismo comportamiento numérico en la descripción HLS, debido a las diferencias en la forma en que cada herramienta define y aplica los parámetros de punto fijo. La utilización de entornos distintos a lo largo del desarrollo hace necesario verificar la correspondencia entre sus implementaciones y comprobar que las diferencias numéricas introducidas no alteren el comportamiento esperado. Por esta razón, el proceso requiere mantener trazabilidad sobre las decisiones numéricas adoptadas a medida que el diseño avanza hacia la implementación en hardware.

Considerando el contexto anterior, este trabajo propone y evalúa un flujo orientado al diseño e implementación en FPGA de controladores MPC lineales con formulación implícita, integrando la exploración de representaciones en punto fijo, su implementación mediante herramientas HLS y la validación progresiva hasta pruebas de tipo *in-the-loop*. En el grupo de investigación en el que se enmarca esta tesis, trabajos previos han abordado la implementación en FPGA de controladores MPC implícitos y la exploración de alternativas de arquitectura mediante HLS, manteniendo una representación numérica en punto flotante [10, 25]. Esta tesis da continuidad a esa línea al incorporar la representación en punto fijo como una variable de diseño y estudiar su efecto sobre la arquitectura generada y el comportamiento del controlador en lazo cerrado. A su vez, el flujo aborda una brecha no cubierta en la literatura al mantener la trazabilidad sobre las decisiones y diferencias numéricas introducidas desde la exploración de la representación hasta su implementación en hardware.

1.2. Formulación del problema

La implementación de un controlador MPC en FPGA mediante aritmética de punto fijo requiere establecer un compromiso entre la precisión numérica, el comportamiento del sistema en lazo cerrado y el costo de la arquitectura resultante. Cuando se dispone de un modelo de alto nivel como referencia funcional, la representación numérica puede explorarse durante etapas tempranas del diseño mediante simulaciones que permitan comparar distintas precisiones y evaluar su efecto respecto de la implementación original en punto flotante, incorporando la representación seleccionada antes de avanzar hacia su implementación mediante HLS y posterior síntesis en FPGA.

Para que los resultados de esta exploración puedan utilizarse posteriormente en la implementación, la configuración seleccionada debe trasladarse de manera consistente al entorno HLS, verificando las diferencias que aparecen entre etapas y distinguiendo aquellas asociadas a la reducción de precisión de las que puedan introducirse por el uso de bibliotecas con comportamientos aritméticos distintos. Esta comparación debe permitir, además, comprobar que dichas diferencias no comprometan los criterios establecidos para el comportamiento en lazo cerrado.

A partir de la literatura revisada, no se identificó para MPC implícito un flujo de implementación mediante HLS que integre, dentro de una misma evaluación sistemática, la exploración temprana de representaciones de punto fijo, su traslado al entorno de síntesis, la verificación de la correspondencia numérica entre ambas representaciones, el comportamiento del controlador en lazo cerrado y el costo de la arquitectura obtenida.

A partir de lo anterior, el problema abordado en esta tesis se formula de la siguiente manera:

"*Dado un controlador MPC lineal con formulación implícita, cuya implementación de referencia en punto flotante se encuentra validada a nivel funcional, se requiere establecer una metodología y directrices prácticas que permitan seleccionar, durante etapas tempranas del diseño, una representación en punto fijo compatible con criterios definidos de desempeño en lazo cerrado. Esta configuración debe poder trasladarse posteriormente a un entorno HLS, que puede ser distinto del utilizado para el modelado, verificando las diferencias numéricas introducidas durante la implementación y relacionándolas con el comportamiento del sistema controlado y con el costo de la arquitectura resultante.*"

1.3. Hipótesis

Es posible seleccionar durante etapas tempranas del diseño una representación en punto fijo para un controlador MPC implícito que conserve un comportamiento numérico consistente al trasladarse a través del flujo HLS, mantenga el desempeño en lazo cerrado dentro de criterios de aceptación definidos y reduzca la utilización de recursos en FPGA respecto de una implementación equivalente en punto flotante.

1.4. Alcances y contribuciones

Los principales alcances considerados en este trabajo son los siguientes:

A1. La metodología se evalúa sobre casos de estudio con controladores MPC previamente formulados y validados en punto flotante, por lo que no se modifica su formulación ni los parámetros que definen el problema de control. Esta delimitación permite concentrar el análisis en los efectos introducidos por la representación en punto fijo.

A2. Los controladores MPC considerados utilizan ADMM como solver, siguiendo trabajos previos que han demostrado que su estructura computacional resulta adecuada para explotar el paralelismo en FPGA [6,10]. Esta elección delimita los casos evaluados, pero los procedimientos asociados a la exploración de punto fijo, su traslado entre herramientas y la verificación progresiva no dependen exclusivamente de ADMM y pueden extenderse a otros algoritmos, como se ha demostrado en [26] mediante la aplicación del flujo a un controlador PI.

A3. El flujo se desarrolla y valida utilizando MATLAB R2023b y Vitis HLS 2025.1. Si bien la reproducibilidad exacta de los pasos descritos y resultados reportados no se puede garantizar para versiones distintas de las herramientas de diseño, se espera que los procedimientos y directrices sean extensibles a nuevas versiones y otras herramientas equivalentes.

A partir de los resultados obtenidos, la principal contribución de la tesis es:

C1. Se verifica que es posible seleccionar una representación en punto fijo durante etapas tempranas del desarrollo de un controlador MPC y trasladarla hasta su implementación en FPGA, manteniendo trazabilidad sobre las diferencias numéricas introducidas entre las distintas etapas. En los casos de estudio, esta trazabilidad permitió comprobar el cumplimiento de los criterios definidos de desempeño en lazo cerrado y una reducción en la utilización de recursos respecto de una implementación equivalente en punto flotante.

1.5. Estructura de la tesis

El resto de este documento se organiza de la siguiente manera:

- El Capítulo 2 presenta los conceptos necesarios para contextualizar el trabajo y revisa el estado del arte relacionado con MPC, Diseño Basado en Modelos, implementación mediante HLS y uso de representaciones de punto fijo en FPGA.
- El Capítulo 3 desarrolla los fundamentos de la aritmética de punto fijo y establece la nomenclatura y las convenciones utilizadas a lo largo de la tesis para describir las representaciones numéricas y el comportamiento de las operaciones.
- El Capítulo 4 describe las herramientas utilizadas y el flujo de trabajo propuesto para la exploración, traslado y verificación de representaciones de punto fijo. Su aplicación se ilustra inicialmente mediante una operación aritmética con vectores, utilizada como caso simple para comparar el comportamiento numérico entre las distintas etapas del flujo y su implementación en FPGA.
- El Capítulo 5 aplica el flujo propuesto a dos casos de estudio de control: un motor de corriente continua y un sistema de recursos energéticos distribuidos (*Distributed Energy Resources*, DER). En ambos casos se evalúa el comportamiento del controlador en lazo cerrado y el efecto de la representación numérica sobre la arquitectura generada.
- El Capítulo 6 presenta las conclusiones generales de la tesis, resume los principales resultados y plantea posibles líneas de trabajo futuro.
- Finalmente, como material complementario, el Apéndice A ilustra la aplicación práctica del flujo de trabajo propuesto a un controlador PI medido en una planta real, mientras que el Apéndice B enumera las publicaciones científicas generadas a partir del desarrollo de esta tesis.

La tesis se acompaña de un repositorio de GitHub[1] que reúne los modelos y scripts de simulación, las implementaciones en C/C++, los *testbenches* y los archivos utilizados en las distintas etapas de verificación e implementación, con el fin de facilitar la reproducción de los resultados. Al momento de completar este trabajo, el repositorio se encuentra disponible previa solicitud debido a restricciones asociadas a los proyectos de investigación en curso, aunque se espera habilitar su acceso público una vez finalizados.

[1] `https://github.com/FranciscaDB/Tesis`

Capítulo 2

ANTECEDENTES

En este capítulo se presentan los conceptos y trabajos previos necesarios para contextualizar la implementación en punto fijo de controladores MPC mediante HLS. Primero se revisan los aspectos de MPC relevantes para su formulación implícita, junto con el uso de FPGA y herramientas HLS para acelerar la resolución en línea del problema de optimización. Luego se introduce el Diseño Basado en Modelos como marco para relacionar el modelo funcional de referencia con las distintas etapas de implementación. Finalmente, se revisan métodos para determinar los formatos de punto fijo, antecedentes de su uso en implementaciones de MPC y estrategias para verificar el comportamiento numérico y en lazo cerrado.

2.1. Control predictivo por modelo

El Control Predictivo por Modelo utiliza un modelo de la planta para predecir su evolución sobre un horizonte finito y determinar una secuencia de actuaciones mediante la resolución de un problema de optimización [1]. Este problema incorpora una función de costo para comparar las trayectorias predichas y restricciones que permiten representar límites sobre las variables del sistema. Para un sistema lineal discreto e invariante en el tiempo, el modelo utilizado para realizar las predicciones puede expresarse como:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k, \end{aligned} \tag{2.1}$$

donde $\mathbf{x}_k$, $\mathbf{u}_k$ y $\mathbf{y}_k$ representan, respectivamente, los vectores de estados, actuaciones y salidas en el instante de muestreo k. Las matrices $\mathbf{A}$ y $\mathbf{B}$ describen la evolución de los estados y el efecto de las actuaciones sobre esta evolución, mientras que $\mathbf{C}$ relaciona los estados con las salidas del sistema.

A partir del estado actual, el modelo permite predecir la evolución del sistema para distintas secuencias de actuaciones dentro de un horizonte de predicción N. La función de costo establece el criterio utilizado para comparar estas trayectorias de acuerdo con los objetivos del controlador. Para simplificar la notación, se considera que los estados y actuaciones se expresan respecto de sus valores de equilibrio, de modo que el punto de operación corresponde al origen. Bajo esta consideración, una forma cuadrática habitual de la función de costo es:

$$J_k = \sum_{i=0}^{N-1} \left(\mathbf{x}_{i|k}^T \mathbf{\Omega} \mathbf{x}_{i|k} + \mathbf{u}_{i|k}^T \mathbf{\Gamma} \mathbf{u}_{i|k} \right) + \mathbf{x}_{N|k}^T \mathbf{\Omega}_N \mathbf{x}_{N|k}, \tag{2.2}$$

donde $\mathbf{x}_{i|k}$ y $\mathbf{u}_{i|k}$ representan el estado y la actuación predichos para el instante $k+i$ a partir de la información disponible en k. Las matrices de ponderación $\mathbf{\Omega}$ y $\mathbf{\Gamma}$ determinan la penalización asociada a

las desviaciones de los estados y de las actuaciones respecto del punto de operación, mientras que $\mathbf{\Omega}_N$ pondera el estado al final del horizonte. La minimización de J_k permite así comparar las secuencias de actuaciones según las trayectorias que producen y las ponderaciones definidas para cada término.

La secuencia de actuaciones se obtiene minimizando la función de costo y, al mismo tiempo, respetando la dinámica del modelo y las restricciones impuestas sobre las variables del sistema. Para representar las actuaciones consideradas a lo largo del horizonte, se define el vector:

$$\mathbf{U}_k = [\mathbf{u}_{0|k}^T, \ldots, \mathbf{u}_{N-1|k}^T]^T.$$

El problema de optimización puede expresarse de manera general como:

$$\begin{aligned} \mathbf{U}_k^\star = \arg\min_{\mathbf{U}_k} \quad & J_k \\ \text{sujeto a} \quad & \mathbf{x}_{0|k} = \mathbf{x}_k, \\ & \mathbf{x}_{i+1|k} = \mathbf{A}\mathbf{x}_{i|k} + \mathbf{B}\mathbf{u}_{i|k}, \quad i = 0, \ldots, N-1, \\ & \mathbf{u}_{\text{mín}} \leq \mathbf{H}\mathbf{u}_{i|k} \leq \mathbf{u}_{\text{máx}}, \quad i = 0, \ldots, N-1, \\ & \mathbf{x}_{\text{mín}} \leq \mathbf{I}\mathbf{x}_{i|k} \leq \mathbf{x}_{\text{máx}}, \quad i = 1, \ldots, N. \end{aligned} \tag{2.3}$$

Para un modelo lineal como el de (2.1), una función de costo cuadrática y restricciones lineales, este problema corresponde a un problema QP [1]. Al incorporar la dinámica del modelo en las predicciones y agrupar las variables de decisión, el problema puede escribirse en la forma compacta

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \frac{1}{2}\boldsymbol{\theta}^T\mathbf{Q}\boldsymbol{\theta} + \mathbf{q}^T\boldsymbol{\theta} \\ \text{sujeto a} \quad & \mathbf{G}\boldsymbol{\theta} \leq \mathbf{g}, \end{aligned} \tag{2.4}$$

donde $\boldsymbol{\theta}$ agrupa las variables de decisión del problema, $\mathbf{Q}$ y $\mathbf{q}$ definen su función de costo, mientras que $\mathbf{G}$ y $\mathbf{g}$ representan las restricciones. La forma específica de estas matrices y vectores depende del modelo, el horizonte de predicción y las restricciones consideradas.

La solución $\mathbf{U}_k^\star$ contiene las actuaciones óptimas calculadas para el horizonte de predicción. De acuerdo con el principio de horizonte deslizante, solo la primera actuación de esta secuencia se aplica a la planta,

$$\mathbf{u}_k = \mathbf{u}_{0|k}^\star, \tag{2.5}$$

tras lo cual el estado se mide o estima nuevamente y el problema de optimización se formula para el siguiente período de muestreo.

La actuación de control puede obtenerse mediante formulaciones explícitas o implícitas de MPC. En MPC explícito, el problema de optimización se resuelve fuera de línea, durante la etapa de diseño del controlador, para el conjunto de estados considerado. Para sistemas lineales con función de costo cuadrático y restricciones lineales, la solución puede expresarse como una función afín por partes definida sobre distintas regiones del espacio de estados [27]. Durante la ejecución se identifica la región que contiene al estado actual y se evalúa la función afín correspondiente, reduciendo la carga de cálculo en línea. Sin embargo, a medida que aumenta la dimensión del problema, el número de regiones puede crecer significativamente, aumentando tanto los requerimientos de memoria para almacenar la solución como el costo de identificar en línea la región correspondiente al estado actual [6].

En MPC implícito, el problema de optimización se resuelve en línea durante cada período de muestreo. En este caso, el QP debe resolverse dentro del tiempo disponible antes de aplicar la siguiente actuación, por lo que el costo computacional del método de optimización adquiere especial importancia cuando se trabaja con períodos de muestreo reducidos [3]. Entre los métodos utilizados para resolver problemas QP se encuentran los métodos de punto interior, de conjunto activo y de primer orden.

Dentro de los métodos de primer orden, se han estudiado algoritmos cuyas iteraciones pueden organizarse mediante operaciones de álgebra lineal con distintos grados de paralelismo, característica

Algorithm 1 Procedimiento iterativo de ADMM

Require: $\mathbf{q}$, $\mathbf{g}$
1: $\mathbf{t}^{(0)}, \mathbf{z}^{(0)}, \mathbf{w}^{(0)} \leftarrow$ valores de inicialización
2: **for** $j = 0, \ldots, IT_{\text{ADMM}} - 1$ **do**
3: $\quad \mathbf{v}^{(j)} \leftarrow \mathbf{z}^{(j)} - \mathbf{g} + \mathbf{w}^{(j)}$
4: $\quad \mathbf{t}^{(j+1)} \leftarrow \mathbf{R}^{-1}(-\rho \mathbf{G}^T \mathbf{v}^{(j)} - \mathbf{q})$
5: $\quad \mathbf{z}^{(j+1)} \leftarrow \text{máx}\{0, -\mathbf{G}\mathbf{t}^{(j+1)} - \mathbf{w}^{(j)} + \mathbf{g}\}$
6: $\quad \mathbf{w}^{(j+1)} \leftarrow \mathbf{w}^{(j)} + \mathbf{G}\mathbf{t}^{(j+1)} + \mathbf{z}^{(j+1)} - \mathbf{g}$
7: **end for**
8: **return** $\mathbf{t}^{(IT_{\text{ADMM}})}$

que ha sido aprovechada en implementaciones sobre FPGA [6, 9, 28]. Los controladores utilizados en esta tesis emplean ADMM para resolver el problema QP y mantienen la formulación utilizada en [25], donde se estudió su implementación mediante HLS y la aplicación de distintas directivas de síntesis. Por completitud, a continuación se presenta brevemente el método ADMM utilizado por los controladores considerados en este trabajo.

2.1.1. Método de multiplicadores de dirección alternante

ADMM es un método iterativo de primer orden que puede utilizarse para resolver el problema QP asociado al MPC. Para la formulación considerada en esta tesis, el método conduce a la secuencia de actualizaciones presentada en el Algoritmo 1.

Considérese el problema QP presentado en (2.4), cuya función de costo está definida por la matriz $\mathbf{Q}$ y el vector $\mathbf{q}$, mientras que las restricciones se describen mediante la matriz $\mathbf{G}$ y el vector $\mathbf{g}$. Para un controlador y una formulación MPC determinados, $\mathbf{Q}$ y $\mathbf{G}$ dependen de parámetros establecidos durante el diseño y permanecen constantes durante la operación. En cambio, $\mathbf{q}$ y $\mathbf{g}$ dependen de la información disponible en cada instante de muestreo y deben actualizarse antes de resolver el nuevo problema QP. Si el parámetro ρ de ADMM también se mantiene fijo, la matriz $\mathbf{R}^{-1} = (\mathbf{Q} + \rho \mathbf{G}^T \mathbf{G})^{-1}$ puede calcularse fuera de línea y almacenarse para su uso durante la ejecución del controlador. El Algoritmo 1 resume el procedimiento iterativo utilizado para resolver cada problema.

En el Algoritmo 1, $\mathbf{t}^{(j)}$ corresponde a la aproximación de las variables de decisión del QP en la iteración j, mientras que $\mathbf{z}^{(j)}$ es una variable de holgura y $\mathbf{w}^{(j)}$ corresponde a la variable dual escalada asociada a las restricciones. El vector $\mathbf{v}^{(j)}$ es una variable auxiliar utilizada en la actualización de $\mathbf{t}^{(j+1)}$, mientras que IT_{ADMM} establece el número de iteraciones ejecutadas para cada problema QP. Los valores obtenidos al finalizar una resolución pueden reutilizarse como valores iniciales en el siguiente instante de muestreo mediante *warm-start*.

Las iteraciones de ADMM presentan una dependencia secuencial, ya que cada iteración utiliza los resultados de la anterior. Dentro de cada iteración, en cambio, varias operaciones elemento a elemento pueden ejecutarse de manera independiente, mientras que las multiplicaciones matriz-vector permiten evaluar simultáneamente parte de sus productos. Por ejemplo, cada componente de $\mathbf{v}^{(j)} = \mathbf{z}^{(j)} - \mathbf{g} + \mathbf{w}^{(j)}$ puede calcularse de forma independiente. Esta estructura permite reducir la latencia de las operaciones mediante paralelismo en hardware, a cambio de utilizar una mayor cantidad de recursos [15].

2.2. Implementación de MPC en FPGA

La necesidad de resolver el problema de optimización dentro del período de muestreo ha motivado el uso de plataformas de cómputo especializadas para acelerar las operaciones asociadas a MPC. En particular, las FPGA permiten construir arquitecturas adaptadas a la estructura del algoritmo, ejecutar en paralelo operaciones independientes y organizar secuencias de cálculo mediante estructuras de *pipeline*. Estas capacidades han sido aprovechadas para reducir el tiempo de ejecución de distintos métodos de optimización utilizados en MPC [6, 9, 28].

Las primeras implementaciones especializadas de métodos de optimización para MPC se

desarrollaron mediante descripciones a nivel RTL, donde el diseñador define explícitamente las unidades funcionales, las memorias y la coordinación de las operaciones a lo largo de los ciclos de reloj. Una revisión de estas implementaciones se presenta en [7], considerando aspectos como la representación numérica, el paralelismo y la organización de memoria. Entre los antecedentes representativos se encuentran implementaciones de métodos de punto interior [28], del Método de Gradiente Rápido [8,11] y de ADMM [6,9]. Algunas de estas propuestas incorporan aritmética de punto fijo y analizan los errores asociados a la precisión reducida [6, 8].

El diseño a nivel RTL permite un control detallado sobre la arquitectura y, para los problemas considerados en la literatura citada, ha permitido alcanzar menores tiempos de ejecución que implementaciones en procesadores. Sin embargo, adaptar una arquitectura RTL a otros problemas o explorar nuevas configuraciones de hardware y formatos de punto fijo requiere modificar manualmente la descripción, lo que incrementa el tiempo necesario para el desarrollo y la verificación [13]. Por otra parte, aunque algunos estudios incluyen evaluaciones en lazo cerrado, el análisis de la representación numérica suele centrarse en la precisión del *solver* y en métricas de implementación, mientras que su efecto sobre el comportamiento completo del sistema controlado recibe menor atención.

Las herramientas HLS permiten trabajar con un nivel de abstracción mayor que el diseño RTL tradicional, ya que parten de algoritmos expresados en lenguajes como C o C++ para obtener una descripción de hardware sintetizable. Mediante directivas de síntesis, el diseñador puede modificar, entre otros aspectos, el desenrollado de lazos, el uso de *pipeline*, la organización de memoria y la reutilización de unidades funcionales [12]. Estas directivas facilitan la exploración de alternativas con diferentes compromisos entre latencia, tasa de procesamiento y utilización de recursos, sin describir manualmente cada arquitectura.

Las herramientas HLS se han utilizado en MPC tanto para automatizar la obtención de descripciones RTL como para explorar distintas arquitecturas y grados de paralelismo en el cálculo [13, 14]. Dentro del grupo de investigación en el que se enmarca esta tesis, en [10] se utilizó Vitis HLS para evaluar inicialmente la viabilidad de acelerar un *solver* ADMM mediante FPGA, manteniendo las configuraciones de síntesis por defecto. Posteriormente, en [15] se amplió este análisis mediante la exploración de distintas directivas de síntesis y la evaluación de su efecto sobre la latencia y la utilización de recursos.

En los estudios reportados en [13, 15], la exploración de la arquitectura se realiza manteniendo una representación numérica en punto flotante. De esta forma, la arquitectura puede modificarse sin incorporar simultáneamente la representación numérica como otra variable de diseño. Sin embargo, las herramientas HLS también permiten usar tipos de datos de precisión arbitraria, lo que abre la posibilidad de estudiar la representación numérica junto con la arquitectura.

La aritmética de punto fijo permite adaptar el rango y la resolución de los datos a las necesidades de una aplicación y, cuando se dimensiona adecuadamente, puede reducir la utilización de recursos y la latencia de las operaciones respecto de alternativas en punto flotante [6, 8, 16, 17], por lo que esta flexibilidad requiere seleccionar una cantidad de bits suficiente para representar los valores alcanzados durante la ejecución y mantener un error compatible con el comportamiento esperado del algoritmo.

Definir el formato de punto fijo mediante la longitud total de palabra y la cantidad de bits fraccionarios no determina por sí solo el comportamiento numérico de una implementación. Las bibliotecas utilizadas en los entornos de modelado y síntesis pueden diferir en las configuraciones disponibles para la cuantización, el tratamiento del desbordamiento y la precisión de los resultados intermedios [18,19], de modo que un mismo formato de bits no garantiza resultados equivalentes entre herramientas.

2.3. Diseño basado en modelos

Las diferencias que pueden aparecer al trasladar una representación numérica entre herramientas hacen necesario mantener una referencia común durante las distintas etapas de implementación. El MBD proporciona un marco para organizar este proceso mediante modelos ejecutables que permiten evaluar el comportamiento esperado del sistema antes de incorporar detalles propios de la plataforma

objetivo [22]. Estos modelos pueden utilizarse para comparar de forma incremental los resultados obtenidos e identificar las diferencias introducidas a medida que avanza el diseño [23].

En esta tesis, este enfoque resulta especialmente útil al incorporar aritmética de punto fijo, ya que la representación numérica seleccionada durante las primeras etapas debe trasladarse posteriormente al entorno HLS y verificarse durante el avance hacia la implementación en FPGA.

Durante el desarrollo basado en modelos se utilizan distintas modalidades de simulación y verificación según el nivel en que se ejecuta el algoritmo y su proximidad con la plataforma objetivo. En el entorno de MATLAB/Simulink, estas modalidades incluyen:

- **Model-in-the-Loop (MIL):** tanto el algoritmo de control como la planta se representan y ejecutan mediante modelos dentro del entorno de modelado. En esta etapa se evalúa el comportamiento funcional del sistema sin incorporar todavía el código ni los detalles de la plataforma donde se implementará el controlador.

- **Software-in-the-Loop (SIL):** el modelo del algoritmo de control se reemplaza por su implementación en software, típicamente escrita en C o C++ y destinada a utilizarse en etapas posteriores de la implementación, mientras la planta continúa simulándose en el entorno de modelado. Esta modalidad permite comparar el comportamiento del código con el modelo utilizado como referencia.

- **Processor-in-the-Loop (PIL) o FPGA-in-the-Loop (FIL) [29]:** el algoritmo se ejecuta sobre la plataforma de destino, mientras la planta continúa simulándose en el entorno de modelado. En PIL, el código se compila y ejecuta sobre el procesador objetivo. En FIL, el algoritmo debe transformarse previamente en una descripción de hardware sintetizable, por ejemplo mediante HLS a partir de código C o C++, para luego implementarse y ejecutarse sobre la FPGA. En ambos casos, el entorno de modelado proporciona los estímulos y recibe los resultados para compararlos con la referencia.

- **Hardware-in-the-Loop (HIL):** el controlador implementado en la plataforma física interactúa con un modelo de la planta ejecutado en tiempo real. Esta modalidad permite evaluar el controlador bajo condiciones temporales e interfaces próximas a las del sistema real, sin utilizar todavía la planta física.

Estas modalidades descritas no constituyen necesariamente una secuencia que deba recorrerse de forma completa, sino que representan distintas formas de verificar el diseño a medida que se incorporan elementos propios de la implementación y de la plataforma objetivo.

En un desarrollo orientado a FPGA, el diseño puede avanzar desde un modelo de alto nivel hacia código destinado a HLS y, posteriormente, hacia una arquitectura implementada en hardware. En este contexto, se han reportado herramientas orientadas a automatizar parte de esta transición. Un ejemplo es SPLIT [30], presentada para una familia de métodos de separación de operadores utilizados en MPC. La herramienta proporcionaba una interfaz en MATLAB y generaba código C destinado a implementaciones en procesadores, FPGA o plataformas heterogéneas, incorporando además soporte para el posterior despliegue sobre FPGA. Este trabajo constituye un antecedente de la automatización del traslado desde una formulación de alto nivel hacia una implementación destinada a hardware, aunque la selección de una representación de punto fijo no forma parte del procedimiento reportado.

Más allá de automatizar el traslado entre etapas, la aritmética de punto fijo introduce decisiones que deben conservarse durante este proceso. Una referencia previamente validada puede utilizarse para explorar representaciones de punto fijo antes de avanzar hacia la implementación y comprobar posteriormente su comportamiento al reproducirlas en el entorno HLS y en FPGA. Esta correspondencia adquiere especial importancia cuando la exploración numérica y la síntesis utilizan bibliotecas de punto fijo diferentes, ya que la utilidad de una exploración temprana depende de que la representación seleccionada pueda trasladarse de manera consistente entre ambos entornos.

2.4. Precisión numérica en el diseño de controladores MPC

Los antecedentes relacionados con la precisión numérica en implementaciones de punto fijo destinadas a FPGA se organizan en torno a tres aspectos: los métodos utilizados para determinar los formatos de punto fijo, las estrategias empleadas para incorporar esta aritmética en controladores MPC y las formas de verificar el comportamiento numérico durante las distintas etapas de implementación.

2.4.1. Selección de formatos de punto fijo

Determinar una representación de punto fijo adecuada no es una tarea trivial, ya que la cantidad de bits asignada a cada variable condiciona tanto los errores introducidos por la aritmética como el costo de su implementación, mientras que el número de configuraciones posibles crece rápidamente cuando se consideran formatos diferentes dentro de un mismo algoritmo. Por esta razón, la selección de precisión para algoritmos numéricos destinados a implementaciones en FPGA continúa siendo objeto de investigación y se han propuesto métodos para automatizarla bajo restricciones de error.

En [16] se aborda este problema mediante un proceso iterativo de optimización de longitudes de palabra apoyado en HLS, que considera la asignación y reutilización de operadores en la arquitectura para obtener una implementación que satisfaga una restricción de precisión. Más recientemente, en [20] se propone una estrategia basada en la propagación del error para determinar formatos específicos para las variables de un algoritmo a partir de una precisión requerida en su salida, generando posteriormente código con tipos de punto fijo destinado a síntesis mediante HLS. Ambos trabajos sistematizan la asignación de precisión y consideran su posterior implementación en hardware, aunque están planteados para algoritmos numéricos de propósito general y no incorporan criterios propios del comportamiento de un sistema de control en lazo cerrado.

En MPC, la reducción de precisión puede introducir errores durante las operaciones utilizadas para resolver el problema de optimización. Por ello, distintos trabajos han estudiado su efecto sobre los algoritmos de resolución y los requisitos de precisión. Entre los antecedentes orientados específicamente a MPC en punto fijo, en [6] se analiza el comportamiento del Método de Gradiente Rápido (*Fast Gradient Method*, FGM) y ADMM bajo aritmética de precisión finita, mientras que en [31] se estudia la precisión necesaria para mantener determinadas garantías numéricas en un método de proyección de gradiente dual. Más recientemente, en [32] se propone un procedimiento para seleccionar la precisión de una implementación en punto fijo a partir de un límite admisible para el error introducido por la cuantización. Desde una perspectiva complementaria, en [33] se muestra que la formulación numérica del problema también puede influir en su sensibilidad a la precisión reducida y, por tanto, en el comportamiento obtenido en lazo cerrado.

2.4.2. Implementaciones de MPC en punto fijo mediante herramientas HLS

Los trabajos que incorporan aritmética de punto fijo en implementaciones de MPC mediante herramientas HLS difieren tanto en la forma de seleccionar la precisión como en el apoyo proporcionado por las herramientas durante el paso hacia hardware. En algunos casos, los formatos se definen específicamente para la aplicación a partir de sus requerimientos numéricos, mientras que en otros se utilizan herramientas de análisis de rango o entornos de generación automática para apoyar la selección de tipos y las etapas posteriores de implementación.

En [34] se comparan implementaciones FPGA de un controlador MPC utilizando punto flotante de precisión simple, media precisión y punto fijo. Para esta última representación, los autores señalan que el ancho de las variables debe determinarse a partir del rango y la precisión requeridos para evitar desbordamientos y limitar los efectos del redondeo. El trabajo reporta formatos para parte de las variables, compara las representaciones en términos de recursos y tiempo de ejecución y valida el controlador mediante simulación *Hardware-in-the-Loop*. Sin embargo, no detalla un procedimiento general para obtener los formatos de punto fijo, por lo que su selección permanece ligada a los requerimientos de la aplicación estudiada.

Un antecedente particularmente cercano a esta tesis se presenta en [21], donde se aborda la

conversión a punto fijo de un controlador MPC basado en FGM utilizando MATLAB Fixed-Point Converter para registrar los rangos observados durante un conjunto de pruebas y proponer anchos de palabra de acuerdo con la precisión requerida. Una vez realizada esta exploración, el desarrollo continúa mediante Vivado HLS y Vitis hasta obtener una implementación sobre FPGA. Durante el proceso, los autores introducen modificaciones en el algoritmo original, entre ellas el escalamiento de algunas variables para reducir los anchos de palabra requeridos, y posteriormente reestructuran el código C++ e incorporan directivas de síntesis orientadas a la arquitectura FPGA. El procedimiento combina así herramientas de apoyo automático con decisiones y ajustes realizados por el diseñador, particularmente al avanzar entre entornos que cumplen funciones diferentes dentro del flujo de implementación. La arquitectura resultante se evalúa en términos de precisión numérica, tiempo de ejecución y utilización de recursos, incluyendo la comparación del error respecto de una referencia en punto flotante. Sin embargo, no se presenta una comparación directa entre las implementaciones de punto fijo utilizadas durante la exploración en MATLAB y posteriormente en HLS que permita identificar posibles diferencias numéricas introducidas al reproducir la representación entre ambos entornos, ni se evalúa en lazo cerrado la implementación FPGA resultante.

Un enfoque diferente se presenta en [24], donde se utiliza Simulink HDL Coder para implementar en FPGA un controlador *Finite Control Set - Model Predictive Control* (FCS-MPC) aplicado a un convertidor de potencia. Los formatos de punto fijo se seleccionan de acuerdo con el rango y la resolución requeridos por las señales. Para la mayoría de ellas se adopta una longitud de palabra de 32 bits, mientras que los operandos de las multiplicaciones se ajustan a los anchos disponibles en los bloques DSP del dispositivo utilizado con el propósito de limitar el consumo de estos recursos. El trabajo se centra principalmente en establecer un flujo integrado dentro de MATLAB/Simulink que abarque el modelado, la definición de los tipos numéricos, la obtención de la descripción HDL y la posterior verificación. Aunque se reporta la utilización de recursos de la arquitectura obtenida, no se exploran diferentes configuraciones de punto fijo ni se compara su costo de implementación con una versión equivalente en punto flotante, por lo que no se caracteriza de forma sistemática el compromiso entre precisión numérica y utilización de recursos.

En conjunto, estos trabajos muestran que la definición de los formatos continúa dependiendo de las características de la aplicación y de la plataforma objetivo. Los procedimientos utilizados difieren tanto en los criterios de selección como en la forma de trasladar la representación hacia las etapas posteriores de implementación. Esta variabilidad hace necesario considerar también cómo se verifica la correspondencia numérica entre dichas etapas.

2.4.3. Verificación numérica y evaluación en lazo cerrado

Además del procedimiento utilizado para determinar los formatos de punto fijo, los trabajos revisados difieren también en el alcance de la verificación realizada sobre la configuración numérica seleccionada. Cuando una misma configuración de punto fijo se reproduce en otro entorno, aplicar las mismas entradas a ambas implementaciones permite comprobar si mantienen un comportamiento numérico consistente. Si además se dispone de una referencia en punto flotante, es posible separar el error asociado a la reducción de precisión de las diferencias introducidas al reproducir la representación entre entornos. Esta verificación resulta especialmente relevante cuando el modelado y la síntesis utilizan bibliotecas distintas para representar y operar los datos de punto fijo.

Las comparaciones realizadas sobre un conjunto fijo de entradas permiten caracterizar las diferencias numéricas entre implementaciones, pero no muestran cómo estas diferencias afectan la evolución del sistema. En lazo cerrado, una variación en la actuación modifica el estado siguiente de la planta y, por consiguiente, las entradas que recibe el controlador en los instantes posteriores. Los trabajos revisados incorporan este tipo de evaluación con distintos alcances. En [21], la implementación en punto fijo se compara con una referencia utilizando un conjunto de entradas previamente definido. En [35], en cambio, se comparan trayectorias en lazo cerrado obtenidas con distintas precisiones, mientras que [24, 34] extienden la evaluación hasta pruebas *Hardware-in-the-Loop*. Estos antecedentes muestran que existen estrategias para verificar tanto la precisión numérica de la implementación como su efecto sobre la respuesta del sistema, aunque estas verificaciones no se integran de manera sistemática

con la selección de los formatos, su reproducción entre diferentes entornos y la caracterización de la arquitectura resultante.

2.5. Discusión

La literatura revisada muestra que los distintos aspectos asociados a la implementación de MPC con precisión reducida han sido estudiados desde perspectivas complementarias. Existen métodos generales para seleccionar longitudes de palabra bajo restricciones de precisión [16, 20], análisis específicos sobre los efectos de la aritmética finita en algoritmos utilizados para MPC [6, 31, 32] e implementaciones que incorporan representaciones en punto fijo, herramientas de alto nivel y validación sobre FPGA [21, 24, 34, 35].

A partir de estos antecedentes, no se identificó un flujo para MPC implícito implementado mediante HLS que integre explícitamente la exploración temprana de una representación en punto fijo, su traslado entre entornos de modelado y síntesis con bibliotecas numéricas distintas, la verificación de la correspondencia entre implementaciones, la evaluación del controlador en lazo cerrado y el análisis del uso de recursos y la latencia de las implementaciones resultantes. Abordar este problema requiere precisar los aspectos de la aritmética de punto fijo relevantes para este flujo, los cuales se desarrollan en el siguiente capítulo.

Capítulo 3

FUNDAMENTOS DE REPRESENTACIÓN EN PUNTO FIJO

La representación en punto fijo permite representar valores reales mediante estructuras binarias simples y eficientes, lo que, al implementarse en hardware digital, puede traducirse en ventajas importantes en términos de utilización de recursos, latencia y consumo energético. Sin embargo, aprovechar estas ventajas requiere comprender con claridad la relación entre la palabra binaria almacenada, el formato numérico utilizado y el valor real que dicha palabra representa. Esta relación determina, entre otros aspectos, qué valores pueden ser representados, con qué resolución, y qué ocurre cuando una operación produce un resultado que no puede ser expresado exactamente dentro del formato disponible.

Este capítulo presenta los fundamentos y la terminología que permitirán entender los análisis desarrollados en los capítulos posteriores, asociados a la implementación de sistemas en punto fijo en hardware digital. El énfasis está puesto en entregar una interpretación intuitiva de la representación binaria subyacente a la aritmética en punto fijo, centrada en la relación entre bits, formato y valor numérico, más allá de las definiciones formales habitualmente tratadas en la literatura [36].

En particular, el capítulo introduce la notación adoptada en este trabajo para describir formatos numéricos, la interpretación de palabras binarias en punto fijo, la relación entre ancho de palabra, rango y precisión, y los efectos asociados a operaciones aritméticas fundamentales, tales como crecimiento de palabra, truncamiento y desbordamiento. Para dar un enfoque práctico, las definiciones se acompañan con ejemplos disponibles en el repositorio asociado al trabajo, lo que permite reforzar las explicaciones conceptuales y facilitar la reproducción de los resultados.

3.1. Notación para representación en punto fijo

En general, un número en punto fijo se almacena en un sistema computacional como una palabra binaria de longitud fija, compuesta por un número determinado de bits. La palabra binaria se interpreta dividiendo sus bits en dos grupos, un conjunto de bits asociados a la parte entera y otro conjunto de bits asociados a la parte fraccionaria. El punto binario que separa la parte entera de la fraccionaria no corresponde a un bit adicional ni se almacena físicamente en la memoria o en los registros del sistema, ya que su ubicación es implícita y queda determinada por la convención adoptada para el formato.

Una forma ampliamente utilizada para describir el formato en punto fijo es la ***notación Q***, introducida originalmente en aplicaciones de procesamiento digital de señales y adoptada en diversas

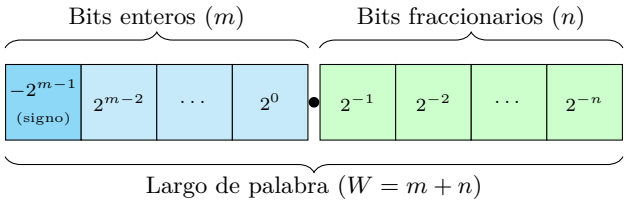


Figura 3.1: Generalización de la Notación $Q_{m,n}$ para el formato de punto fijo.

fuentes, tales como manuales técnicos y documentación de herramientas de diseño [37,38]. En términos generales, esta notación se expresa como $Q_{m,n}$, donde los parámetros m y n indican cómo se distribuyen los bits de la palabra entre la parte entera y la parte fraccionaria.

Es importante mencionar que, en la práctica, la notación Q carece de una estandarización formal, lo que ha dado lugar a múltiples convenciones para interpretar sus parámetros. Por ejemplo, el parámetro m puede representar el número de bits de la parte entera incluyendo o excluyendo el bit de signo, o incluso el número total de bits de la palabra, mientras que el parámetro n puede referirse a los bits destinados a la parte fraccionaria o también a la parte entera. Esta falta de uniformidad no es un aspecto meramente notacional, ya que una interpretación incorrecta del formato numérico puede derivar en errores de escalamiento y pérdidas de precisión, además de dificultar la reproducibilidad de resultados y la portabilidad de diseños entre distintas herramientas o plataformas. En consecuencia, la literatura reciente sigue enfatizando la necesidad de explicitar la convención adoptada al utilizar la notación Q [20,39].

Con el fin de evitar ambigüedades y asegurar la consistencia a lo largo de este trabajo, se adoptará la convención presentada en [40], resumida en la Figura 3.1. Bajo esta definición, el formato $Q_{m,n}$ representa números con signo en complemento a dos, utilizando m bits para la parte entera (incluyendo el bit de signo) y n bits para la parte fraccionaria, de modo que la longitud total de palabra queda dada por $W = m + n$ bits. Por completitud, se introduce la notación $U_{m,n}$ para números en punto fijo sin signo, con m bits para la parte entera y n bits para la parte fraccionaria.

3.2. Rango, precisión y representación binaria

La utilización de una palabra binaria de longitud fija para representar la información numérica en punto fijo hace que el conjunto de valores posibles sea finito y discreto. Como consecuencia, todo formato de punto fijo presenta dos características principales: una **resolución**, que determina la separación entre valores consecutivos, y un **rango dinámico**, que determina los valores máximo y mínimo representables.

Como se mencionó previamente, en representación de punto fijo el punto binario no se almacena explícitamente en la palabra, por lo que al utilizar la notación $Q_{m,n}$, la palabra binaria almacenada en un sistema digital puede entenderse como un número entero q_x, codificado en complemento a dos, el cual se debe escalar para obtener el valor real representado. La relación entre el entero almacenado y el valor real asociado está dada por la cantidad de bits asignados a la parte fraccionaria del número según

$$\hat{x} = q_x \cdot 2^{-n} \tag{3.1}$$

donde $\hat{x}$ representa el valor real interpretado a partir del entero almacenado q_x y el factor 2^{-n} corresponde al tamaño del paso entre dos valores consecutivos representables en el formato. A partir de lo anterior, la **resolución** del formato, comúnmente denotada como Δ, queda definida por

$$\Delta = 2^{-n} \tag{3.2}$$

De manera intuitiva, la representación en punto fijo genera una grilla discreta y uniformemente

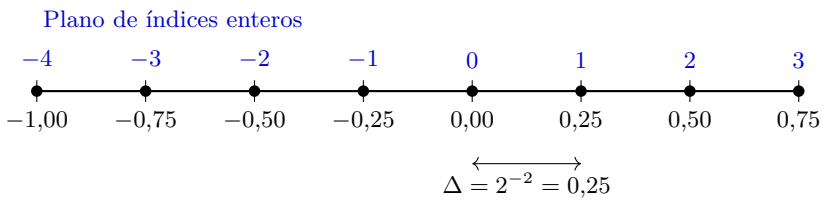


Figura 3.2: Grilla de cuantización para un formato de punto fijo $Q_{1,2}$.

espaciada sobre el eje numérico. Cada punto de esta grilla corresponde a un valor representable, y la distancia entre dos puntos consecutivos es precisamente Δ.

El **rango dinámico** se obtiene a partir del intervalo de valores posibles del entero q_x codificado en complemento a dos. Para una palabra de W bits, este entero puede tomar valores en el intervalo

$$q_x \in \left[-2^{W-1};\, 2^{W-1}-1\right] \tag{3.3}$$

Al aplicar el factor de escala 2^{-n}, el intervalo de valores reales representables queda dado por

$$\hat{x} \in \left[-2^{W-1}2^{-n};\, \left(2^{W-1}-1\right)2^{-n}\right] \tag{3.4}$$

Considerando que en la notación Q se tiene que $W = m + n$, este resultado puede escribirse como

$$\hat{x} \in \left[-2^{m-1};\, 2^{m-1}-2^{-n}\right] \tag{3.5}$$

Es relevante notar que el rango no es simétrico, lo que se debe a que en complemento a dos existe una palabra binaria adicional para los valores negativos.

A modo ilustrativo, la Figura 3.2 muestra la grilla de valores representables para el formato $Q_{1,2}$. En este caso, el ancho total de palabra es $W = 3$ bits, la resolución es

$$\Delta = 2^{-2} = 0{,}25$$

el rango de enteros representables se encuentra en el intervalo

$$q_x \in [-4;\, 3]$$

y el rango dinámico para los valores de $\hat{x}$ es

$$\text{Rango } Q_{1,2} = [-1{,}00;\, 0{,}75]$$

Desde esta perspectiva, representar un número real en punto fijo consiste en determinar qué índice entero q_x de la grilla corresponde al valor que se desea almacenar. Para un valor real x, el procedimiento general es el siguiente:

1. Se aplica el factor de escala asociado al número de bits fraccionarios:
$$x_{\text{scaled}} = x \cdot 2^n \tag{3.6}$$
2. Se obtiene el valor q_x tomando una aproximación entera de x_{scaled}. Si x_{scaled} ya es entero, entonces $q_x = x_{\text{scaled}}$.
3. El entero q_x se codifica en complemento a dos utilizando W bits. La palabra binaria resultante, denotada como q_x^{bin}, corresponde a la información que efectivamente se almacena en el sistema digital.

A continuación, se muestran ejemplos para ilustrar el procedimiento para obtener la representación en punto fijo para los números reales $x = 0{,}25$ e $y = -0{,}75$.

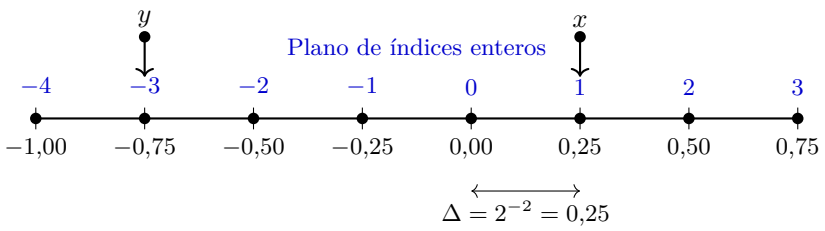


Figura 3.3: Grilla de cuantización para el formato $Q_{1,2}$. En la parte superior (azul) se ilustran los índices enteros correspondientes a cada nivel. La variable x mapea directamente sobre el índice 1, correspondiente al valor real 0,25, mientras que el ejemplo y se ilustra sobre el índice -3, correspondiente a $-0{,}75$.

Ejemplo 3.1: Representación de $x = 0{,}25$ en formato $Q_{1,2}$

En este caso, el valor escalado está dado por

$$x_{\text{scaled}} = 0{,}25 \cdot 2^2 = 1.$$

Como el resultado escalado ya es entero, se tiene $q_x = 1$. Al codificar este valor en complemento a dos con $W = 3$ bits, se obtiene

$$q_x = 1 \quad \Longrightarrow \quad q_x^{\text{bin}} = 001.$$

Aplicando la relación inversa, el valor real representado por esta palabra binaria es

$$\hat{x} = q_x \cdot 2^{-n} = 1 \cdot 2^{-2} = 0{,}25.$$

Ejemplo 3.2: Representación de $y = -0{,}75$ en formato $Q_{1,2}$

En este caso, el valor escalado está dado por

$$y_{\text{scaled}} = -0{,}75 \cdot 2^2 = -3.$$

Como el resultado escalado ya es entero, se tiene $q_y = -3$. Al codificar este valor en complemento a dos con $W = 3$ bits, se obtiene

$$q_y = -3 \quad \Longrightarrow \quad q_y^{\text{bin}} = 101.$$

Aplicando la relación inversa, el valor real representado por esta palabra binaria es

$$\hat{y} = q_y \cdot 2^{-n} = -3 \cdot 2^{-2} = -0{,}75.$$

La Figura 3.3 muestra una representación gráfica de cómo los valores se mapean a puntos de la grilla.

De forma intuitiva, la palabra binaria no almacena directamente el valor real, sino un índice entero de la grilla. El signo del índice indica el sentido del desplazamiento respecto al origen, hacia la derecha para valores positivos y hacia la izquierda para valores negativos, mientras que la magnitud determina cuántos pasos de tamaño Δ separan al valor representado del cero. Esta interpretación se ilustra en la Figura 3.3, donde los valores reales de la grilla se muestran junto con sus índices enteros asociados.

Los ejemplos anteriores corresponden a casos en que el valor escalado x_{scaled} es exactamente entero, de modo que el número real que se desea representar coincide con uno de los puntos de la grilla. En la práctica, esto no siempre ocurre y, debido a la naturaleza finita del formato, pueden presentarse dos situaciones. Primero, el valor puede estar dentro del rango dinámico, pero no coincidir exactamente con un punto de la grilla, por lo que debe aproximarse mediante cuantización. Segundo, el valor puede exceder los límites del rango representable, en cuyo caso se produce desbordamiento. Estos dos efectos,

cuantización y **desbordamiento**, son consecuencias directas de representar números reales mediante un conjunto finito de palabras binarias.

3.3. Cuantización y desbordamiento

Con el fin de mantener la uniformidad de notación, en el resto del documento se utilizarán símbolos de letras minúsculas en cursivas (por ejemplo: a, b, x) para representar una variable con valor en el dominio de los reales. Luego, se utilizarán las siguientes convenciones de notación para representar el valor numérico de la variable en distintas etapas de la conversión e interpretación en punto fijo:

x: Valor original en el dominio de los números reales que se desea almacenar o representar en formato de punto fijo.

x_{scaled}: Valor escalado. Se obtiene utilizando la ecuación (3.6).

q_x: Valor cuantizado. Es el número entero que resulta de aproximar el valor escalado a número entero mediante un proceso de cuantización. Este es el valor que se almacena físicamente en la memoria en representación binaria.

q_x^{bin}: Representación binaria del valor cuantizado q_x. Corresponde a la secuencia de bits efectivamente almacenada en memoria.

$q_x^{\text{bin}}[i]$: Notación utilizada para referirse a un bit en específico de la representación binaria. El índice i indica la posición del bit, donde $i = W - 1$ corresponde al bit de signo e $i = 0$ al bit menos significativo.

$\hat{x}$: Valor reconstruido que representa la aproximación del valor original a partir del valor cuantizado. Se calcula usando la ecuación (3.1).

3.3.1. Cuantización

La cuantización es el proceso mediante el cual una magnitud de dominio continuo o representada en alta precisión se mapea a un conjunto finito de valores discretos de menor precisión. En el contexto del diseño digital, este proceso no solo ocurre durante la conversión de señales analógicas, sino también de forma recurrente en las operaciones internas del sistema. En este sentido, la representación en punto fijo introduce una discretización inherente del dominio numérico, lo que implica que no todos los valores del dominio de origen son exactamente representables dentro de un formato dado.

Como se vio en la Sección 3.2, al representar un número real en punto fijo utilizando un formato previamente definido, se realiza un escalamiento con el fin de obtener el índice discreto asociado dentro de la grilla de representación. Si el valor escalado corresponde a un número entero, entonces el valor es exactamente representable en dicha grilla. En caso contrario, el resultado queda ubicado entre dos niveles discretos consecutivos, por lo que es necesario aplicar un método de aproximación para asignarlo a uno de los índices válidos. Para ilustrar este proceso, considérese un formato $Q_{4,4}$ (donde $W = 8$ y $n = 4$). En este caso, el rango de índices enteros válidos es $[-128;\ 127]$. Al escalarlos por la resolución del sistema:

$$\Delta = 2^{-4} = 0{,}0625$$

el rango dinámico representable es:

$$[-128;\ 127] \cdot \Delta = [-8;\ 7,9375]$$

Sea el valor real:

$$x = 3{,}27$$

Su proyección al dominio entero es:

$$x_{\text{scaled}} = 3{,}27 \cdot 2^4 = 3{,}27 \cdot 16 = 52{,}32$$

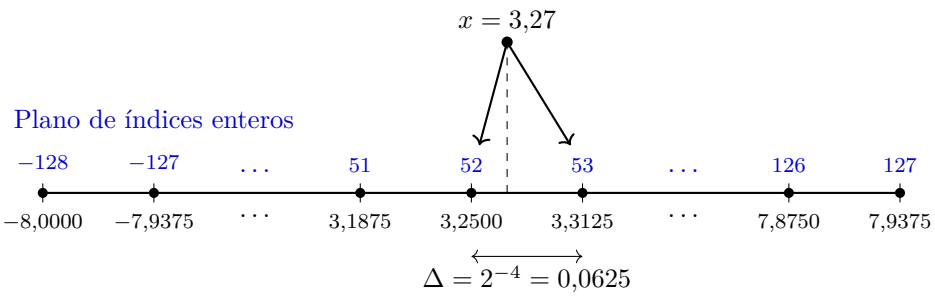


Figura 3.4: Mapeo de la grilla de cuantización para el formato $Q_{4,4}$. En la parte superior (azul) se ilustran los índices en el plano entero, delimitados por el rango de representación $[-128; 127]$ y en la parte inferior se muestran los valores de la grilla de cuantización. El valor $x = 3{,}27$ cae entre los índices 52 y 53, requiriendo una aproximación.

El valor escalado 52,32 evidencia que el número no es exactamente representable en la grilla asociada al formato de punto fijo $Q_{4,4}$. En consecuencia, al no corresponder a un índice entero de dicha grilla, es necesario aproximarlo a uno de sus niveles discretos antes de su almacenamiento, lo que introduce un **error de cuantización**, tal como se observa en la Figura 3.4.

El proceso de cuantización define la transición desde una representación continua, o de mayor precisión, hacia una representación discreta y finita, y constituye la base sobre la cual se implementan los métodos de aproximación utilizados para mapear los valores a los índices enteros de la grilla. La elección de la técnica de aproximación representa un compromiso directo entre la precisión numérica, es decir, la minimización del error de cuantización, y la complejidad de su implementación a nivel lógico. En la práctica, estas estrategias se agrupan en dos familias principales: los métodos de **truncamiento** y los métodos de **redondeo**.

3.3.1.1. Truncamiento

El truncamiento aproxima el valor hacia un entero adyacente basándose en una dirección fija, sin evaluar la magnitud de la fracción. Si x_{scaled} ya corresponde a un número entero, la cuantización mantiene dicho valor, mientras que para valores no enteros se distinguen las siguientes variantes:

- **Truncamiento hacia** $-\infty$: El valor se aproxima al número entero representable inmediatamente menor.
- **Truncamiento hacia cero**: El valor se aproxima en dirección al cero.
- **Truncamiento hacia** $+\infty$: El valor se aproxima al número entero representable inmediatamente mayor.

En el ejemplo ilustrado en la Figura 3.4 ($x = 3{,}27$, con un valor escalado de 52,32), aplicar truncamiento hacia $-\infty$ implica descartar la parte fraccionaria, resultando en el índice 52. Esto equivale al valor cuantizado $\hat{x} = 3{,}25$, introduciendo un error de cuantización $e_q = x - \hat{x} = 0{,}02$. Para valores escalados cuya parte fraccionaria se aproxima a la unidad, como 52,99, el error absoluto provocado por el truncamiento hacia $-\infty$ puede llegar a aproximarse al paso de cuantización completo (Δ). La posibilidad de tener errores cercanos a la resolución motiva el uso de técnicas de redondeo, que permiten reducir la cota máxima del error de cuantización.

3.3.1.2. Redondeo

El redondeo consiste en aproximar el valor escalado al número entero representable más cercano, garantizando que el error absoluto máximo quede acotado a $\pm\frac{\Delta}{2}$. Tomando nuevamente el ejemplo de

la Figura 3.4 ($x_{\text{scaled}} = 52{,}32$), el entero más cercano es 52, por lo que el valor cuantizado es 3,25 y el error es de 0,02. Este error cumple con el límite establecido, ya que es menor a $\frac{\Delta}{2} = 0{,}03125$.

Este criterio de proximidad funciona de manera directa cuando existe una clara diferencia de distancias entre el valor real y los dos enteros adyacentes. Sin embargo, surge una ambigüedad cuando el valor se encuentra exactamente en el punto medio entre dos enteros consecutivos (es decir, la parte fraccionaria de x_{scaled} es 0,5). En esta situación de **empate**, ambos enteros están equidistantes del valor original.

Las principales reglas de desempate determinístico pueden agruparse según el criterio empleado para seleccionar uno de los dos enteros equidistantes:

- **Desempate direccional**: selecciona uno de los enteros adyacentes según una dirección preestablecida, que puede ser hacia $+\infty$, hacia $-\infty$, hacia cero o alejándose del cero.
- **Desempate por paridad**: selecciona uno de los enteros adyacentes según su paridad, favoreciendo el entero par o el entero impar.

Las reglas basadas en paridad pueden reducir el sesgo acumulado del error de cuantización al evitar que todos los empates se resuelvan sistemáticamente en una misma dirección, especialmente cuando los datos presentan una distribución aproximadamente equilibrada.

Los principales modos de redondeo son:

- **Redondeo hacia** $+\infty$ (*direccional*): En caso de empate, el valor se aproxima hacia el infinito positivo.
- **Redondeo hacia** $-\infty$ (*direccional*): En caso de empate, el ajuste favorece el infinito negativo.
- **Redondeo dependiente del signo** (*direccional*): El empate se resuelve alejándose del cero (hacia $+\infty$ para positivos y hacia $-\infty$ para negativos).
- **Redondeo hacia cero** (*direccional*): El empate se resuelve estrictamente en dirección al origen.
- **Redondeo convergente (round-to-even)** (*paridad*): El valor se ajusta de modo que el entero resultante sea par. Es el método estándar de IEEE 754.
- **Redondeo impar (round-to-odd)** (*paridad*): Similar al anterior, pero favoreciendo enteros impares.

3.3.1.3. Efectos de truncamiento y redondeo

A modo de síntesis, la Tabla 3.1 resume el comportamiento de los distintos métodos de cuantización aplicados a valores representativos. En ella se muestran los valores cuantizados obtenidos y el error de cuantización absoluto asociado en cada caso, lo que permite comparar directamente el efecto de cada estrategia.

Para los valores que no corresponden a un empate, $x = 3{,}30$ y $x = -3{,}30$, todos los modos de redondeo seleccionan el punto de la grilla más cercano y producen un error absoluto de 0,0125. En los métodos de truncamiento, el resultado depende de la dirección establecida y puede coincidir con el punto más cercano o con el valor representable adyacente, como muestran los errores de 0,0125 y 0,05. Esta diferencia refleja las cotas de ambas estrategias, ya que el redondeo al valor más cercano limita el error absoluto a $\Delta/2$, mientras que mediante truncamiento este puede aproximarse a un paso de cuantización completo Δ.

En la condición de empate, representada por $x_{\text{scaled}} = 52{,}5$, los dos valores cuantizados posibles presentan el mismo error absoluto, $\Delta/2 = 0{,}03125$. Por lo tanto, las reglas de desempate no modifican la magnitud del error, sino la dirección de la aproximación. Las reglas direccionales seleccionan uno de los valores según el criterio establecido, mientras que las reglas basadas en paridad lo hacen según los índices adyacentes.

Tabla 3.1: Comparación de errores generados por distintos métodos de truncamiento y redondeo al almacenar valores en un formato $Q_{4,4}$.

		$x = 3{,}30$ $x_{\text{scaled}} = 52{,}8$		$x = -3{,}30$ $x_{\text{scaled}} = -52{,}8$		$x = 3{,}28125$ $x_{\text{scaled}} = 52{,}5$	
Categoría	Método	$\hat{x}$	$\lvert e_q \rvert$	$\hat{x}$	$\lvert e_q \rvert$	$\hat{x}$	$\lvert e_q \rvert$
Truncamiento	hacia $-\infty$	3,25	0,05	-3,3125	0,0125	3,25	0,03125
	hacia cero	3,25	0,05	-3,25	0,05	3,25	0,03125
	hacia $+\infty$	3,3125	0,0125	-3,25	0,05	3,3125	0,03125
Redondeo	hacia $+\infty$	3,3125	0,0125	-3,3125	0,0125	3,3125	0,03125
	hacia $-\infty$	3,3125	0,0125	-3,3125	0,0125	3,25	0,03125
	dependiente del signo	3,3125	0,0125	-3,3125	0,0125	3,3125	0,03125
	hacia cero	3,3125	0,0125	-3,3125	0,0125	3,25	0,03125
	convergente	3,3125	0,0125	-3,3125	0,0125	3,25	0,03125
	impar	3,3125	0,0125	-3,3125	0,0125	3,3125	0,03125

La diferencia entre las reglas de desempate adquiere relevancia cuando estas operaciones se repiten sobre una secuencia de datos. Las reglas direccionales establecen una misma preferencia ante empates equivalentes, por lo que su aplicación reiterada puede favorecer una determinada dirección del error. Las reglas basadas en paridad, en cambio, hacen depender la selección de la paridad de los índices adyacentes y evitan imponer una dirección única a todos los empates. En particular, el redondeo convergente puede reducir el sesgo acumulado asociado a estos casos cuando los datos presentan una distribución aproximadamente equilibrada. Por esta razón, la elección del modo de cuantización debe considerar tanto la magnitud como la tendencia del error introducido.

3.3.2. Desbordamiento

Mientras que la cuantización lidia con la pérdida de información en los bits menos significativos debido a limitaciones de resolución, el **desbordamiento** (u *overflow*) ocurre cuando una señal, o el resultado de una operación, supera los límites del rango dinámico representable por el formato. En otras palabras, aparece cuando la magnitud del valor requiere más bits enteros de los que el formato puede representar.

Para ilustrar el fenómeno de desbordamiento, considérese nuevamente el formato $Q_{4,4}$. Como se estableció previamente, el rango dinámico de este formato está acotado en el dominio real al intervalo $[-8;\ 7{,}9375]$, lo que corresponde a los índices enteros válidos $[-128;\ 127]$. Sea el valor real a almacenar:

$$x = 9{,}5$$

Al aplicar el factor de escalamiento para su proyección al dominio entero, se obtiene:

$$x_{\text{scaled}} = 9{,}5 \cdot 2^4 = 9{,}5 \cdot 16 = 152$$

El valor escalado 152 evidencia que el número excede el límite máximo del rango representable (127). A nivel binario, representar el entero 152 en complemento a dos requiere 9 bits (010011000_2), lo que genera un desbordamiento al intentar guardarlo en un registro de 8 bits. Tal como se ilustra en la Figura 3.5, el valor recae completamente fuera de la grilla de índices válidos.

Cuando el sistema detecta o produce una magnitud que excede el rango dinámico, debe aplicar un mecanismo para manejar los bits excedentes y forzar al resultado a tomar un valor válido dentro de la grilla. Al igual que en la cuantización, existen dos modos principales para resolver esta condición: **Envolvimiento** y **Saturación**.

3.3.2.1. Envolvimiento (wrap o módulo)

El envolvimiento es el comportamiento natural de la aritmética binaria. Cuando el resultado de una operación excede el rango representable, los bits más significativos que no pueden representarse

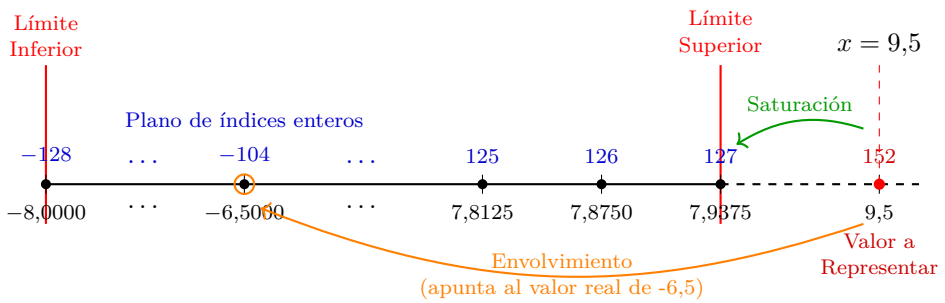


Figura 3.5: Representación del desbordamiento para $x = 9{,}5$ en formato $Q_{4,4}$. El valor escalado (152) excede el límite superior de la grilla (127). La saturación lo ancla al máximo permitido (127), mientras que el envolvimiento proyecta el valor hacia el espectro negativo del rango (debido a la interpretación del bit de signo en complemento a dos).

se descartan.

Como consecuencia, el valor numérico "da la vuelta" y vuelve a ingresar por el extremo opuesto del rango representable. En representaciones con signo (complemento a dos), este fenómeno puede provocar que un número positivo grande pase a interpretarse como un número negativo. Por lo tanto, una variación pequeña en el valor real puede producir un cambio abrupto en el valor representado, generando una discontinuidad en la representación numérica.

En la Figura 3.5, la flecha naranja ilustra el comportamiento de envolvimiento, al intentar representar el valor $x = 9{,}5$ (que correspondería al índice entero 152), el resultado excede el límite superior del rango representable (127). En lugar de mantenerse en el extremo superior, el envolvimiento hace que el valor "salte" al extremo opuesto del rango, interpretándose como $-6{,}5$ (índice -104). Este salto desde un valor positivo grande hacia un valor negativo demuestra la naturaleza discontinua del envolvimiento, donde una pequeña variación que cause desbordamiento puede resultar en una interpretación completamente diferente del número.

3.3.2.2. Saturación

La saturación es un mecanismo en el cual, si el resultado de una operación excede el límite superior del rango dinámico, el valor se fija en el máximo representable. De manera análoga, si el resultado cae por debajo del límite inferior, se fija en el mínimo representable por el formato.

En aplicaciones de procesamiento de señales y sistemas de control en lazo cerrado, la saturación suele preferirse porque preserva el sentido físico de las magnitudes representadas. Aunque este mecanismo recorta la señal e introduce no linealidades, evita la inversión de signo característica del envolvimiento.

La flecha verde de la Figura 3.5 ilustra este comportamiento. Cuando el valor $x = 9{,}5$ (índice 152) excede el límite superior del rango, la saturación lo ancla al máximo valor representable, que es 7,9375 (índice 127). A diferencia del envolvimiento, la saturación mantiene la coherencia física del valor, permaneciendo en el extremo positivo del rango sin generar discontinuidades abruptas. Aunque se pierde precisión al recortar 9,5 a 7,9375, el valor saturado conserva el sentido físico de "magnitud grande y positiva". Esta preservación del sentido de la magnitud es crucial en arquitecturas de control, ya que una inversión de signo o un desbordamiento no gestionado puede generar inestabilidades en el sistema físico. Un caso histórico representativo de este riesgo se documenta en [41], donde un desbordamiento durante la conversión de una variable de coma flotante de 64 bits a un entero de 16 bits generó una excepción de hardware que resultó en la destrucción del cohete Ariane 5.

3.4. Aritmética en punto fijo

Una vez definidas las reglas de cuantización y desbordamiento, es posible describir de forma sistemática las operaciones aritméticas fundamentales en formatos de punto fijo. Para ilustrar estas

operaciones de manera clara en las siguientes Secciones, adoptaremos la siguiente convención estándar:

x, y: Variables utilizadas para referirse a los operandos de una operación aritmética.

z: Variable de destino que almacena el resultado de la operación. Se considera que su formato exacto se encuentra definido de antemano.

Q_{m_i,n_i}: Formato específico de la variable i (donde $i \in \{x, y, z\}$). El uso de subíndices denota explícitamente que las representaciones de estas variables no tienen por qué ser iguales, cada operando y resultado puede poseer su propio formato independiente (es decir, Q_{m_x,n_x} para x, Q_{m_y,n_y} para y, y Q_{m_z,n_z} para z).

q_i': Versión extendida del entero q_i empleada durante una operación aritmética. Esta representación puede obtenerse mediante ampliación del ancho en bits y/o desplazamientos lógicos (*shifts*), típicamente para alinear el punto binario entre operandos antes de realizar la operación.

$Q_{m_{\text{ext}},n_{\text{ext}}}$: Formato temporal ampliado requerido para albergar el resultado intermedio exacto. Sus dimensiones m_{ext} (bits enteros), n_{ext} (bits fraccionales) y su ancho total de palabra W_{ext} se calculan algorítmicamente en función de la operación realizada para evitar cualquier pérdida de información o desbordamiento prematuro.

q_{ext}: Valor entero temporal que contiene el resultado matemático exacto de la operación. Este valor reside internamente en el formato ampliado $Q_{m_{\text{ext}},n_{\text{ext}}}$ antes de ser finalmente proyectado, cuantizado y acotado al formato de destino Q_{m_z,n_z}.

De manera general, cualquier operación aritmética en punto fijo se ejecuta directamente sobre los valores enteros almacenados (q_x, q_y). Dado que las operaciones matemáticas naturalmente expanden el rango o la resolución de los números, el flujo de procesamiento general se compone conceptualmente de dos etapas fundamentales:

1. **Cálculo de resultado intermedio en precisión extendida**: La operación matemática genera un resultado temporal exacto que utiliza un formato ampliado, es decir, requiere una cantidad de bits mayor que la de los operandos originales para evitar cualquier pérdida de información. Esto es necesario porque, por ejemplo, la suma de dos números puede requerir un bit extra para la parte entera, la multiplicación suma los anchos de palabra de ambos operandos, y la mezcla de distintos formatos Q exige alinear primero los puntos binarios.
2. **Proyección al formato destino**: Considerando que el resultado debe almacenarse obligatoriamente en el formato predefinido Q_{m_z,n_z} de la variable z, el resultado ampliado exacto debe adaptarse a este nuevo tamaño. Esto implica:
   a) Un ajuste de escala para coincidir con los n_z bits fraccionales de la variable de destino.
   b) La aplicación de una regla de *cuantización* (como truncamiento o redondeo) sobre los bits menos significativos que quedan fuera del formato y son descartados.
   c) El manejo de *desbordamiento*, evaluando si la parte entera del resultado cabe en los m_z bits asignados, de lo contrario, se aplica la técnica configurada, como saturación o envolvimiento.

Es importante notar que, a nivel lógico y algorítmico, el ajuste de escala y la cuantización se realizan de manera conjunta. Durante esta conversión se determinan los bits que se conservarán en el formato destino y, según el modo de cuantización, el valor puede modificarse antes de descartar los bits restantes, por ejemplo, incrementando en una unidad el último bit conservado durante ciertas operaciones de redondeo. Una vez obtenido el valor cuantizado, se comprueba si este pertenece al rango representable y, en caso contrario, se aplica la política de desbordamiento correspondiente.

A continuación se describen las operaciones básicas. A través de diversos ejemplos, se ilustrará cómo se calcula y se enfrenta cada una de las variaciones posibles al operar e integrar formatos distintos.

3.4.1. Suma y resta en punto fijo

Considerando los operandos x e y, la operación de suma queda dada por:

$$x + y$$

Dado que en la notación Q se pueden representar números positivos y negativos, la resta de x e y se puede interpretar mediante una suma con el opuesto aditivo:

$$x - y = x + (-y)$$

De esta forma, ambas operaciones se analizan e implementan mediante la misma lógica de suma entera sobre los patrones binarios. Sin embargo, para que esta operación sea matemáticamente válida, los operandos deben compartir la misma posición del punto binario, de no ser así, deben alinearse previamente.

La suma se implementa mediante tres etapas consecutivas: alineación del punto binario, cálculo de resultado intermedio en precisión extendida, y proyección del valor intermedio al formato de destino.

Alineación del punto binario

Si $n_x \neq n_y$, los operandos representan sus valores con distinta resolución fraccional. Para poder operarlos, es necesario expresarlos en una misma escala, definiendo una cantidad de bits fraccionales extendida (n_{ext}):

$$n_{\text{ext}} = \text{máx}(n_x, n_y) \tag{3.7}$$

El operando con menor número de bits fraccionales se reescala mediante desplazamientos lógicos hacia la izquierda (lo que equivale a multiplicar por potencias de dos) para igualar la resolución:

$$q'_x = q_x \ll (n_{\text{ext}} - n_x), \qquad q'_y = q_y \ll (n_{\text{ext}} - n_y)$$

Es importante notar que este reescalamiento no introduce errores de cuantización, siempre y cuando no se exceda el rango del registro que contiene el valor temporal, en cuyo caso se produciría un desbordamiento temprano. Tras esta alineación, ambos operandos comparten el mismo factor de escala $2^{n_{\text{ext}}}$.

Formato intermedio y precisión extendida

El resultado exacto de la suma pertenece a un formato temporal ampliado $Q_{m_{\text{ext}},n_{\text{ext}}}$, definido típicamente por:

$$m_{\text{ext}} = \text{máx}(m_x, m_y) + 1, \qquad n_{\text{ext}} = \text{máx}(n_x, n_y) \tag{3.8}$$

Es decir, se reserva un bit entero adicional para capturar el posible acarreo (carry) de la suma y se conserva el mayor número de bits fraccionales de los operandos.

El valor entero $q_{\text{ext}} = q'_x + q'_y$ se calcula internamente sobre este formato, por lo que la suma exacta real equivalente sería:

$$x + y = \frac{q_{\text{ext}}}{2^{n_{\text{ext}}}}$$

Solo después de obtener este resultado intermedio, el valor se proyecta al formato destino Q_{m_z,n_z} de la variable z, aplicando el ajuste de escala (descartando bits fraccionales si $n_z < n_{\text{ext}}$), aplicando cuantización y detectando un posible desbordamiento. En implementaciones más restrictivas donde el sumador tiene un ancho fijo de W_z bits y no existe esta precisión extendida, el sumador entrega simplemente la suma módulo 2^{W_z}, y la discrepancia matemática se interpreta como el efecto directo del desbordamiento.

Detección de desbordamiento en complemento a dos

En la suma de enteros, el resultado puede exceder el ancho de la representación de destino. Al almacenarlo, únicamente se conservan los W_z bits menos significativos y se descarta cualquier bit de acarreo que exceda dicho ancho. En representación con signo en complemento a dos, el desbordamiento suele detectarse a partir de los bits de signo.

Sea $q_x^{\text{bin}}[W_x - 1]$ el bit de signo del primer operando, $q_y^{\text{bin}}[W_y - 1]$ el del segundo operando, y $q_{\text{ext}}^{\text{bin}}[W_{\text{ext}} - 1]$ el bit de signo del resultado temporal entregado por el sumador. Existe **overflow** si y sólo si los operandos tienen el mismo signo y el resultado presenta un signo diferente:

$$\text{overflow} = \left(q_x^{\text{bin}}[W_x - 1] = q_y^{\text{bin}}[W_y - 1]\right) \wedge \left(q_{\text{ext}}^{\text{bin}}[W_{\text{ext}} - 1] \neq q_x^{\text{bin}}[W_x - 1]\right) \tag{3.9}$$

- En modo de **envolvimiento** (*wrap*), el valor entero almacenado q_z es simplemente el patrón entregado por el sumador (aritmética modular), ignorando la señal de overflow.
- En modo de **saturación**, una lógica adicional utiliza la señal de overflow para sustituir el valor entero q_z por el valor máximo o mínimo representable por el formato de la variable destino, según corresponda al signo original de los operandos.

El **Ejemplo 3.3** muestra el escenario más simple posible: una suma donde tanto los operandos como la variable de destino comparten exactamente la misma resolución fraccional y ancho de palabra. Como se observa, al mantener la misma escala no se requiere descartar bits fraccionales, y debido a que los operandos poseen signos opuestos, el valor resultante se acomoda de forma natural dentro del rango numérico permitido sin generar desbordamiento.

Ejemplo 3.3: Operandos y destino en representación $Q_{4,4}$, sin cuantización ni desbordamiento

Considérese la suma de $x = 1{,}25$ e $y = -0{,}75$, donde ambos operandos y la variable destino z comparten el formato $Q_{4,4}$.

1. **Escalamiento y codificación en entero**:

   $$x_{\text{scaled}} = 1{,}25 \cdot 2^4 = 20, \quad y_{\text{scaled}} = -0{,}75 \cdot 2^4 = -12$$

   En complemento a dos de 8 bits:

   $$q_x = 20 \Rightarrow \texttt{00010100}, \qquad q_y = -12 \Rightarrow \texttt{11110100}$$

2. **Suma a nivel de bits**:

   $$\begin{array}{r} \texttt{00010100} \\ +\ \texttt{11110100} \\ \hline \texttt{00001000} \end{array}$$

   El entero resultante es:

   $$q_z = \texttt{00001000} = 8$$

3. **Overflow y formato de destino**: Los bits de signo son $q_x^{\text{bin}}[7] = 0$ y $q_y^{\text{bin}}[7] = 1$. Como los signos son distintos, por definición matemática es imposible que ocurra un desbordamiento en la suma:

   $$q_x^{\text{bin}}[7] \neq q_y^{\text{bin}}[7] \;\Rightarrow\; \text{overflow} = 0$$

   Dado que el destino z posee los mismos 4 bits fraccionales que los operandos, no existe necesidad de cuantizar. El valor entero 8 se ubica perfectamente dentro del rango de 8 bits con signo ($-128 \leq q_z \leq 127$). Así, el valor real recuperado en $Q_{4,4}$ es:

   $$\hat{z} = \frac{q_z}{2^4} = \frac{8}{16} = 0{,}5$$

El **Ejemplo 3.4** expone el caso en que se mantiene una representación uniforme $Q_{4,4}$ para todas las variables, pero la magnitud de la suma matemática excede la capacidad del registro de destino. Aquí se ilustra cómo la detección de overflow a través de los bits de signo activa las distintas políticas de desbordamiento, alterando el resultado final y la interpretación del valor almacenado.

Ejemplo 3.4: Operandos y destino en representación $Q_{4,4}$, con desbordamiento en resultado

Considérese ahora la suma de operandos:

$$x = 6{,}75, \qquad y = 3{,}50$$

ambos en formato $Q_{4,4}$, con un destino z también predefinido en $Q_{4,4}$.

1. **Escalamiento y codificación**:

   $$x_{\text{scaled}} = 6{,}75 \cdot 2^4 = 108, \qquad y_{\text{scaled}} = 3{,}50 \cdot 2^4 = 56$$

   En binario de 8 bits con signo:

   $$q_x = 108 \Rightarrow \texttt{01101100}, \qquad q_y = 56 \Rightarrow \texttt{00111000}$$

2. **Suma a nivel de bits en 8 bits**:

   $$\begin{array}{r} \texttt{01101100} \\ +\ \texttt{00111000} \\ \hline \texttt{10100100} \end{array}$$

   El patrón temporal resultante es `10100100`.

3. **Detección de overflow**: Los bits de signo son $q_x^{\text{bin}}[7] = 0$, $q_y^{\text{bin}}[7] = 0$, $q_{\text{ext}}^{\text{bin}}[7] = 1$. Como los operandos son positivos y el resultado aparente es negativo, hay desbordamiento evidente:

   $$(q_x^{\text{bin}}[7] = q_y^{\text{bin}}[7]) \wedge (q_{\text{ext}}^{\text{bin}}[7] \neq q_x^{\text{bin}}[7]) \;\Rightarrow\; \text{overflow} = 1$$

   La suma matemática real exige un entero de valor 164 ($10{,}25 \cdot 16$), el cual excede el máximo permitido en 8 bits (127).

4. **Interpretación en destino $\hat{z}$ según política de desbordamiento**: La elección de la política determina el comportamiento del error ante la incapacidad del formato de destino para albergar el valor real:

   - **Envolvimiento (wrap)**: Genera un salto hacia los números negativos (alterando completamente el significado de la magnitud matemática). q_z almacena directamente el patrón de los 8 bits:

     $$q_z^{\text{wrap}} = \texttt{10100100} = -92 \qquad \hat{z}^{\text{wrap}} = \frac{-92}{2^4} = -5{,}75$$

   - **Saturación**: El sistema detecta el overflow de valores positivos, contiene el error recortando la magnitud y fuerza al registro a almacenar el máximo valor representable del formato:

     $$q_z^{\text{sat}} = \texttt{01111111} = 127 \qquad \hat{z}^{\text{sat}} = \frac{127}{2^4} = 7{,}9375$$

El **Ejemplo 3.5** presenta un caso donde las representaciones de los operandos y el resultado difieren. Este ejemplo ilustra cómo se propaga el error de cuantización a lo largo del cálculo en dos etapas distintas: primero, mediante la cuantización inicial de los operandos, y finalmente, a través

del descarte de bits fraccionales necesario para empaquetar el resultado temporal en su formato de destino. A pesar de esta degradación numérica progresiva, se destaca que todo el proceso, incluyendo la conversión de escalas, el alineamiento y el redondeo, se resuelve utilizando exclusivamente aritmética entera y desplazamientos de bits (*shifts*).

Ejemplo 3.5: Formatos distintos y cuantización de destino

Sean $x = 1{,}23$ almacenado en $Q_{2,2}$, $y = -0{,}68$ almacenado en $Q_{3,4}$ y un resultado z que debe almacenarse en formato $Q_{2,2}$ utilizando regla de **redondeo**.

1. **Cuantización de los operandos en sus propios formatos (redondeo)**: Estas conversiones introducen el primer error de representación previo a la suma al adaptar el valor real a la memoria digital. Para x en $Q_{2,2}$:

$$q_x = \text{round}(1{,}23 \cdot 2^2) = \text{round}(4{,}92) = 5 \Rightarrow \texttt{0101}$$

   Para y en $Q_{3,4}$:

$$q_y = \text{round}(-0{,}68 \cdot 2^4) = \text{round}(-10{,}88) = -11 \Rightarrow \texttt{1110101}$$

2. **Alineación del punto binario**: Se define la escala extendida interna $n_{\text{ext}} = \text{máx}(2, 4) = 4$. Se reescala el operando x desplazándolo 2 posiciones a la izquierda:

$$q'_x = q_x \ll 2 = \texttt{010100} = 20$$

   El operando y ya tiene 4 bits fraccionales: $q'_y = q_y = -11$.

3. **Cálculo en precisión extendida**: La suma en la escala común ($n_{\text{ext}} = 4$) es $q_{\text{ext}} = 20 + (-11) = 9$.

$$\begin{array}{rr} \texttt{00010100} & (20) \\ +\,\texttt{11110101} & (-11) \\ \hline \texttt{00001001} & (9) \end{array}$$

   El valor temporal real exacto interno es $\frac{9}{16} = 0{,}5625$. Una vez extendidos ambos operandos a un ancho interno común, no hay riesgo de overflow interno ya que los operandos son de distinto signo.

4. **Proyección al formato destino z en $Q_{2,2}$ (Redondeo)**: Como el destino z exige $n_z = 2$ bits fraccionales, deben descartarse $k = n_{\text{ext}} - n_z = 2$ bits del resultado interno. Para redondear al valor más cercano antes de este descarte, se suma un *bias* 2^{k-1}, correspondiente a la mitad de la separación entre dos índices consecutivos del formato destino expresada en la escala interna. Luego,

$$q_z = \left\lfloor \frac{q_{\text{ext}} + 2^{k-1}}{2^k} \right\rfloor = \left\lfloor \frac{9+2}{4} \right\rfloor = 2(\text{parte entera}).$$

   A nivel de bits, primero se incorpora el *bias* y luego se realiza el desplazamiento de $k = 2$ posiciones:

$$\begin{array}{rr} \texttt{00001001} & (q_{\text{ext}} = 9) \\ +\,\texttt{00000010} & (bias = 2) \\ \hline \texttt{00001011} & (11) \end{array} \quad \xrightarrow{\gg 2} \quad \texttt{00000010} = 2.$$

   Esta reducción de precisión proyecta el valor interno 0,5625 al valor representable 0,5 en $Q_{2,2}$. Por tanto, el valor finalmente almacenado es

$$\hat{z} = \frac{q_z}{2^2} = \frac{2}{4} = 0{,}5.$$

3.4.2. Multiplicación

La multiplicación exacta en el dominio real se calcula operando directamente sobre los valores enteros almacenados (q_x, q_y) y combinando sus respectivos factores de escala, lo que resulta en:

$$x \cdot y = \frac{q_x \cdot q_y}{2^{n_x+n_y}}$$

Como evidencia esta expresión, la multiplicación en punto fijo genera un resultado con mayor número de bits. El producto de las magnitudes enteras $(q_x \cdot q_y)$ requiere más espacio para la parte entera, mientras que la multiplicación de los denominadores suma directamente las resoluciones fraccionales. Por lo tanto, para albergar este resultado temporal exacto sin pérdida de información, se requiere obligatoriamente un formato *ampliado*, cuyas dimensiones corresponden a la suma de las partes de ambos operandos:

$$m_{\text{ext}} = m_x + m_y, \qquad n_{\text{ext}} = n_x + n_y \tag{3.10}$$

Es decir, a nivel binario, multiplicar operandos de W_x y W_y bits produce siempre un resultado exacto temporal de $W_{\text{ext}} = W_x + W_y$ bits. En la posterior etapa de proyección al formato destino Q_{m_z,n_z} de la variable z, pueden presentarse tres situaciones:

- Si $n_{\text{ext}} > n_z$, es necesario descartar bits fraccionales. La forma en que se descartan (simple truncamiento o algún esquema de redondeo) determina la *cuantización* del resultado.
- Si $m_{\text{ext}} > m_z$, se deben descartar bits de la parte entera. Si, tras aplicar la cuantización, el entero resultante supera el rango permitido por los m_z bits del destino, se produce *desbordamiento* y se aplica la política configurada (saturación o envolvimiento).
- Si $m_{\text{ext}} \leq m_z$ y $n_{\text{ext}} \leq n_z$, el resultado puede almacenarse directamente en z sin pérdida de información por cuantización ni riesgo de desbordamiento.

El **Ejemplo 3.6** ilustra la multiplicación de dos variables y cómo la reducción al formato destino puede generar efectos inesperados en los límites del rango numérico. Específicamente, demuestra un comportamiento particular: cómo el error introducido por la cuantización (al aplicar la regla de redondeo) puede empujar un valor matemáticamente válido hacia un desbordamiento, forzando la intervención de la lógica de saturación para mantener la coherencia del sistema.

Ejemplo 3.6: Multiplicación con reducción de formato, cuantización y desbordamiento inducido

Supóngase la multiplicación de $x = 1{,}5$ e $y = 1{,}25$, donde ambos operandos y la variable destino z comparten el formato $Q_{2,2}$ (ancho total $W = 4$ bits, rango en el dominio real de $[-2{,}0, 1{,}75]$).

1. **Representación de los operandos**

   En $Q_{2,2}$, con $n_x = n_y = 2$ bits fraccionales:

   $$q_x = 1{,}5 \cdot 2^2 = 6, \qquad q_y = 1{,}25 \cdot 2^2 = 5$$

   Como ambos son positivos, se codifican en 4 bits con signo como:

   $$q_x = 6 \Rightarrow \texttt{0110}, \qquad q_y = 5 \Rightarrow \texttt{0101}$$

   Para calcular el producto exacto internamente, la arquitectura extiende el ancho a $W_{\text{ext}} = 8$ bits (rellenando con el bit de signo, que es 0):

   $$q'_x = \texttt{00000110}, \qquad q'_y = \texttt{00000101}$$

2. **Producto entero en precisión extendida**

A nivel de bits, la multiplicación se obtiene como la suma de productos parciales:

00000110	(q'_x)	
$\cdot$ 00000101	(q'_y)	
00000110		LSB de $q_y = 1$, $q_{x,\text{ext}} \ll 0$
00000000		bit 1 de $q_y = 0$, $q_{x,\text{ext}} \ll 1$
00011000		bit 2 de $q_y = 1$, $q_{x,\text{ext}} \ll 2$
00000000		bit 3 de $q_y = 0$, $q_{x,\text{ext}} \ll 3$
00011110	(30)	suma de productos parciales

El producto entero temporal es:

$$q_{\text{ext}} = 30 \Rightarrow 00011110_2$$

La escala extendida es $n_{\text{ext}} = n_x + n_y = 4$, por lo que el valor matemático exacto temporal es:

$$x \cdot y = \frac{q_{\text{ext}}}{2^{n_{\text{ext}}}} = \frac{30}{2^4} = 1{,}875$$

El resultado exacto vive temporalmente en un formato ampliado $Q_{4,4}$.

3. **Proyección al formato destino z en $Q_{2,2}$ (Redondeo) y Desbordamiento**

   El formato destino z dispone de $n_z = 2$ bits fraccionales, por lo que es necesario reducir la resolución desde $n_{\text{ext}} = 4$, descartando:

   $$k = n_{\text{ext}} - n_z = 2 \text{ bits fraccionales}$$

   Para aplicar *redondeo* entero al descartar k bits, se suma un *bias* de 2^{k-1} antes de aplicar el desplazamiento a la derecha (*shift*):

   $$q_z = \frac{q_{\text{ext}} + 2^{k-1}}{2^k} = \frac{30 + 2}{2^2} = \frac{32}{4} = 8$$

   A nivel de bits:

   $$\begin{array}{rr} 00011110 & (q_{\text{ext}} = 30) \\ + \; 00000010 & (\text{bias} = 2) \\ \hline 00100000 & (32) \end{array} \quad \xrightarrow{>>2} \quad 00001000 = 8$$

   El valor entero obtenido tras el redondeo es:

   $$q_z = 8 \Rightarrow 1000_2$$

   Sin embargo, en el formato con signo $Q_{2,2}$ de 4 bits, el máximo valor positivo representable corresponde a:

   $$q_{\text{máx}} = 7 \Rightarrow \hat{z}_{\text{máx}} = \frac{7}{2^2} = 1{,}75$$

   Por lo tanto, el valor $q_z = 8$ excede el rango representable del formato destino. En complemento a dos, el patrón binario 1000_2 se interpreta como el valor negativo $\hat{z} = -2{,}0$, lo que evidencia un desbordamiento inducido durante la etapa de cuantización. Para evitar esta reinterpretación fuera de rango, la lógica de saturación fuerza el resultado al máximo valor positivo representable:

   $$q_z = 0111_2 \qquad \Longrightarrow \qquad \hat{z} = \frac{7}{2^2} = 1{,}75$$

3.4.3. División

La división es la operación aritmética más compleja desde el punto de vista algorítmico, ya que suele requerir procesos iterativos, como restas sucesivas o métodos de convergencia. Desde la perspectiva de la representación en punto fijo, esta operación requiere especial atención porque el cociente puede presentar un rango mayor que el de los operandos y requerir una resolución elevada. Además, a diferencia de la suma y la multiplicación, el resultado de una división no siempre puede representarse exactamente mediante un formato de punto fijo con una cantidad finita de bits fraccionales.

La relación exacta entre los valores reales y los enteros almacenados puede expresarse como:

$$\frac{x}{y} = \frac{q_x/2^{n_x}}{q_y/2^{n_y}} = \frac{q_x}{q_y} \cdot 2^{n_y - n_x}$$

Para analizar el rango y la resolución que puede requerir el cociente, resulta útil considerar los casos extremos asociados a los operandos:

- **Crecimiento de la parte entera:** El máximo valor del cociente se alcanza al dividir el mayor valor representable de x por la menor magnitud positiva representable de y. Dado que esta última está determinada por la resolución de y, el cociente puede requerir hasta $m_x + n_y$ bits enteros para evitar desbordamiento.

- **Crecimiento de la parte fraccionaria:** La menor magnitud no nula del cociente se obtiene al dividir la mínima magnitud positiva representable de x por un valor de y próximo al máximo de su rango. Para representar valores de esta escala puede requerirse una cantidad de bits fraccionales del orden de $n_x + m_y$.

A partir de estos casos extremos, puede utilizarse como referencia un formato ampliado $Q_{m_{\text{ext}},n_{\text{ext}}}$ con:

$$m_{\text{ext}} = m_x + n_y, \qquad n_{\text{ext}} = n_x + m_y$$

Estas expresiones permiten estimar el rango y la resolución necesarios para el cociente, pero no garantizan una representación exacta. En general, la razón q_x/q_y puede producir una expansión binaria no finita, por lo que la implementación debe definir una precisión fraccionaria para el resultado y aplicar la cuantización correspondiente.

A nivel lógico, no es necesario materializar explícitamente un formato temporal con estas cotas. Para calcular la división y expresar el resultado directamente en el formato destino Q_{m_z,n_z} de la variable z, el procesamiento puede estructurarse en las siguientes etapas:

1. **Ajuste de escala:** A partir de la relación entre las escalas de los operandos y del resultado, se define

   $$k = n_z + n_y - n_x.$$

   Este factor permite expresar el cociente en la escala correspondiente a los n_z bits fraccionales del formato destino. Cuando $k > 0$, el dividendo q_x se desplaza k posiciones hacia la izquierda antes de realizar la división. Si $k < 0$, debe realizarse el ajuste de escala equivalente en sentido contrario.

2. **División entera:** Se calcula el cociente entre el dividendo reescalado y el divisor q_y, considerando $q_y \neq 0$. Cuando la división no es exacta, el residuo determina la información que debe descartarse o utilizarse para aplicar la regla de cuantización seleccionada.

3. **Proyección al formato destino:** El cociente cuantizado se asigna a la variable q_z. En esta etapa se comprueba si su magnitud se encuentra dentro del rango definido por los m_z bits enteros y, en caso contrario, se aplica la política de desbordamiento configurada.

El **Ejemplo 3.7** ilustra una división en la que el formato de destino se determina previamente. En particular, muestra cómo el preescalamiento del dividendo mediante desplazamientos permite obtener un cociente con la resolución fraccional requerida. Además, destaca la importancia de disponer de suficientes bits enteros en el formato de destino para evitar desbordamiento durante la representación del resultado.

Ejemplo 3.7: División con ajuste de escala al formato de destino

Considérese la división de $x = 1{,}5$ entre $y = 0{,}5$, donde ambos operandos están almacenados en formato $Q_{2,2}$ (ancho total de 4 bits). El formato exacto de la variable destino z se encuentra predefinido como $Q_{3,2}$ (5 bits totales: 3 enteros incluyendo signo y 2 fraccionales).

1. **Representación de los operandos en $Q_{2,2}$**

   Con $n_x = n_y = 2$ bits fraccionales:

   $$q_x = 1{,}5 \cdot 2^2 = 6, \qquad q_y = 0{,}5 \cdot 2^2 = 2$$

   En binario de 4 bits con signo:

   $$q_x = 6 \Rightarrow \texttt{0110}, \qquad q_y = 2 \Rightarrow \texttt{0010}$$

2. **Escalamiento del dividendo**

   Para obtener mediante división entera un índice cuya escala corresponda a los $n_z = 2$ bits fraccionales del formato destino, es necesario compensar las escalas de los operandos. A partir de la relación entre sus índices enteros, el desplazamiento requerido está dado por $k = n_z + n_y - n_x$. Como en este caso $n_x = n_y$, simplemente desplazamos el dividendo en $n_z = 2$ posiciones hacia la izquierda:

   $$q'_x = q_x \cdot 2^2 = 6 \cdot 4 = 24$$

   A nivel de bits (requiriendo temporalmente más bits para no perder información):

   $$\texttt{00000110} \xrightarrow{<<2} \texttt{00011000} = 24$$

3. **División entera a nivel de bits**

   La arquitectura ejecuta la división entera sobre los valores binarios:

   $$q_z = \frac{q'_x}{q_y} = \frac{24}{2} = 12$$

   En este caso, la división es exacta, por lo que no se introduce un error de cuantización adicional. El índice obtenido puede almacenarse en los 5 bits del formato destino $Q_{3,2}$:

   $$q_z = 12 \Rightarrow \texttt{01100}$$

4. **Interpretación en el formato destino**

   La variable z en formato $Q_{3,2}$ recupera el valor en el dominio real aplicando su factor de escala:

   $$\hat{z} = \frac{q_z}{2^{n_z}} = \frac{12}{2^2} = 3{,}0$$

   El resultado obtenido corresponde a la división analítica $1{,}5/0{,}5 = 3{,}0$. En este ejemplo, el preescalamiento del dividendo mediante un desplazamiento lógico produce un cociente entero compatible con la resolución fraccional requerida ($n_z = 2$). Asimismo, el formato de destino $Q_{3,2}$ proporciona un rango de representación suficiente para almacenar el valor resultante sin desbordamiento.

3.5. Discusión

En resumen, las operaciones aritméticas en punto fijo requieren considerar tanto la precisión utilizada durante el cálculo como el formato en que finalmente se almacena el resultado. En la suma, la resta y la multiplicación es posible determinar un formato ampliado capaz de representar exactamente el resultado de los operandos cuantizados. En la división, en cambio, el cociente puede presentar una expansión binaria no finita, por lo que debe establecerse una precisión para su cálculo.

De manera general, el procesamiento puede organizarse en las siguientes etapas:

1. **Cálculo interno:** Se obtiene el resultado de la operación utilizando una precisión suficiente para el cálculo considerado.
2. **Proyección al formato destino:** El resultado obtenido se mapea a la variable destino z, la cual posee un formato predefinido Q_{m_z,n_z}. Esta proyección puede involucrar los siguientes mecanismos:
   - Un ajuste de escala para expresar el resultado con los n_z bits fraccionales requeridos.
   - Una regla de cuantización, como truncamiento o redondeo, cuando el resultado no puede representarse exactamente en el formato destino.
   - La detección y tratamiento del desbordamiento, mediante saturación o envolvimiento, en caso de que la magnitud calculada exceda el rango representable por los m_z bits enteros.

La conclusión principal de este análisis es que el comportamiento numérico de una implementación en punto fijo no queda determinado únicamente por la especificación del formato $Q_{m,n}$. En este trabajo, se denominará **formato** a la estructura en bits de una variable, definida por la longitud total de palabra y su distribución entre las partes entera y fraccionaria. El resultado de las operaciones depende además de la **configuración** utilizada, entendida como el conjunto de decisiones sobre el modo de cuantización, el tratamiento del desbordamiento y, cuando corresponda, la precisión de los resultados intermedios. En adelante, se utilizará el término **representación numérica** para referirse conjuntamente al formato y a la configuración adoptados. Por lo tanto, dos implementaciones que utilizan el mismo formato pueden producir resultados distintos si difieren en su configuración.

Desde el punto de vista del diseño digital, resulta importante distinguir entre el cálculo interno en precisión extendida y la representación final en el formato de destino. Esta separación permite analizar de forma explícita los efectos de la precisión finita y modelar con exactitud el comportamiento aritmético del sistema a nivel de bits (*bit-true*). De esta manera, los modelos simulados en software pueden reproducir el mismo comportamiento numérico que tendrá posteriormente la implementación en hardware.

Como se estudiará en el siguiente capítulo, la definición de las operaciones en punto fijo constituye la base para implementar algoritmos e interfaces digitales cuyo comportamiento numérico sea consistente entre la simulación y la arquitectura física del sistema.

Capítulo 4

IMPLEMENTACIÓN DE OPERACIONES EN PUNTO FIJO EN FPGA

En el Capítulo 3 se establecieron las convenciones numéricas utilizadas en este trabajo para describir, interpretar y operar valores en punto fijo. A partir de esta base, el presente capítulo aborda la aplicación práctica de dichas convenciones dentro de un flujo de trabajo orientado a la exploración, implementación y validación de algoritmos en FPGA.

La selección de MATLAB y Vitis HLS mantiene la continuidad con herramientas utilizadas en trabajos previos del grupo de investigación [25, 42]. En particular, el trabajo presentado en [25] parte de una implementación funcional en MATLAB, traslada a C++ las operaciones que se implementarán en hardware y utiliza Vitis HLS para generar una descripción RTL destinada a FPGA. Esta secuencia es compatible con el enfoque de Diseño Basado en Modelos descrito en la Sección 2.3, donde un modelo ejecutable se mantiene como referencia mientras el diseño avanza hacia representaciones más próximas a la implementación.

A partir de esta estructura, el flujo propuesto incorpora criterios específicos para trabajar con punto fijo. En una primera etapa, los formatos se exploran sobre el modelo de alto nivel bajo una configuración definida para las operaciones. Este análisis se realiza al inicio del flujo, cuando las iteraciones de diseño suelen requerir menos tiempo y esfuerzo que en etapas posteriores, lo que permite evaluar tempranamente el error numérico antes de trasladar la representación numérica seleccionada a C/C++ para su implementación mediante Vitis HLS. Posteriormente, se comparan los resultados obtenidos en los distintos entornos para identificar posibles diferencias numéricas. El flujo incorpora finalmente una validación mediante *FPGA-in-the-Loop*, que permite evaluar el diseño implementado en FPGA en interacción con el modelo del sistema en el que se integra, en lugar de analizarlo únicamente como un componente aislado.

Además de describir el flujo, este capítulo presenta un ejemplo basado en el cálculo de un producto punto, utilizado como caso simple para recorrer las etapas propuestas y aplicar los criterios de comparación numérica que posteriormente se emplean en los controladores MPC del siguiente capítulo.

4.1. Flujo propuesto para diseño e implementación de algoritmos en FPGA

La transición desde una descripción funcional de alto nivel hacia una implementación en C/C++ compatible con síntesis de hardware mediante HLS requiere verificar el comportamiento esperado del

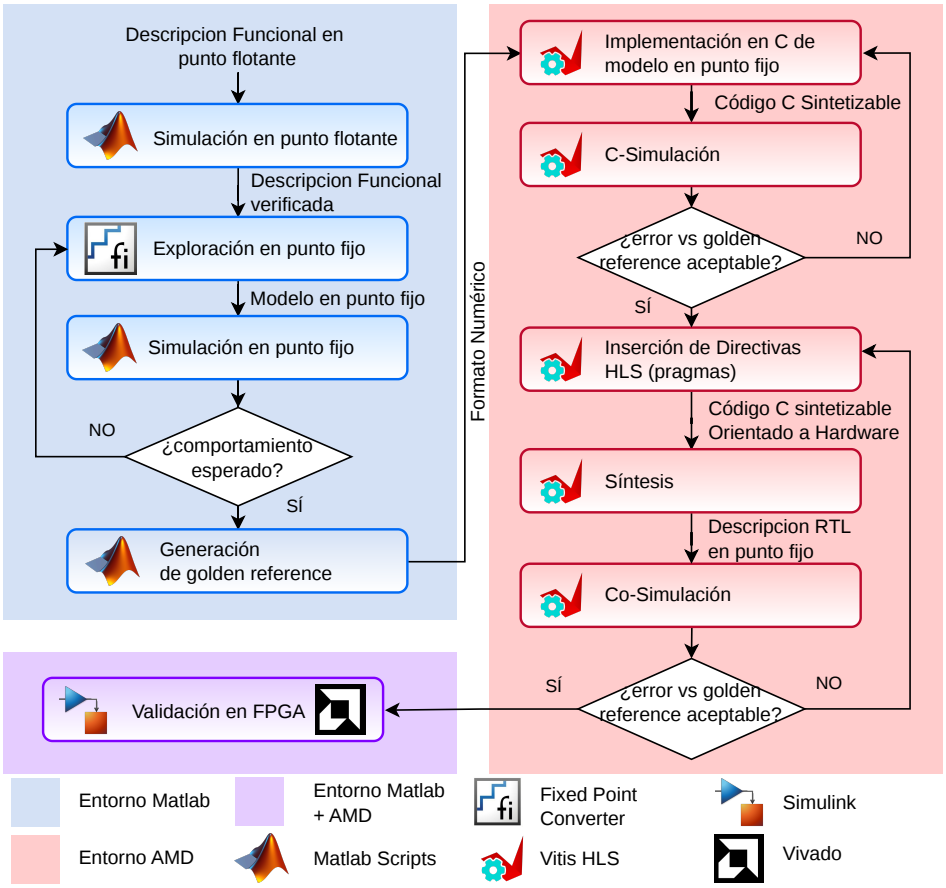


Figura 4.1: Flujo general de herramientas propuesto para la implementación y validación de operaciones en punto fijo.

algoritmo a través de distintas etapas y herramientas, las cuales incorporan progresivamente detalles de implementación. En el caso de algoritmos en punto fijo, esta transición exige además definir formatos numéricos, reglas de cuantización, tratamiento de desbordamiento y criterios de comparación entre las etapas de validación. Para ordenar estas decisiones, en este trabajo se propone un flujo de diseño que organiza las etapas de exploración, implementación y verificación utilizadas durante el proceso.

La Figura 4.1 resume el flujo de trabajo propuesto para sistematizar la implementación y validación de algoritmos en punto fijo, desde una descripción basada en modelos de alto nivel hasta su ejecución en FPGA. El flujo propuesto considera dos entornos de desarrollo principales, MATLAB y AMD, junto con herramientas específicas asociadas a cada etapa, tales como *Fixed-Point Converter*, Vitis HLS, Vivado y Simulink. La elección de estos entornos se alinea con el marco metodológico presentado en [25, 42], donde también se aborda la transición desde modelos de alto nivel hacia implementaciones en hardware. En el presente trabajo, el flujo incorpora un enfoque particular en la consistencia numérica entre las distintas etapas del proceso, tanto dentro de un mismo entorno como al trasladar el diseño hacia otras herramientas de implementación y validación.

El diagrama utiliza un código de colores para distinguir las etapas según el entorno en que se desarrollan y el rol que cumplen dentro del flujo. Los bloques con fondo *azulado* corresponden al entorno MATLAB, donde se realiza la exploración numérica y la validación inicial del modelo en punto fijo. Los bloques con fondo *rojizo* corresponden al entorno AMD, donde el algoritmo se traslada hacia una descripción sintetizable mediante Vitis HLS. Finalmente, el bloque con fondo *lila* representa la etapa de integración y validación experimental, en la cual se combinan herramientas de ambos entornos para comprobar el comportamiento de la implementación ejecutada en una FPGA.

De esta manera, el flujo propuesto permite seguir la evolución de una misma operación o algoritmo desde su formulación algorítmica inicial hasta su ejecución en hardware, manteniendo trazabilidad sobre las decisiones numéricas adoptadas y sobre las diferencias introducidas en cada etapa. Las etapas principales del flujo se describen con mayor detalle en las siguientes subsecciones.

4.1.1. Entorno MATLAB

El entorno MATLAB constituye la etapa inicial del flujo, orientada a definir y validar el comportamiento numérico del algoritmo antes de su implementación en hardware. Esta fase comienza con una descripción algorítmica basada en modelos de alto nivel y aritmética de punto flotante. Luego, se exploran representaciones de punto fijo y se genera una referencia numérica que será utilizada para verificar las implementaciones desarrolladas en las etapas posteriores. Los pasos considerados son:

- **Simulación en punto flotante:** Este paso inicia con una descripción algorítmica basada en modelos del sistema que se desea implementar. Esta descripción se utiliza como modelo de referencia para estudiar el comportamiento esperado del algoritmo antes de incorporar las restricciones de precisión finita o detalles asociados a la implementación física. A partir de un conjunto de estímulos o casos de prueba, se verifica que la respuesta obtenida sea consistente con los criterios definidos para el sistema, tales como el nivel de error admisible, la estabilidad del lazo de control o el cumplimiento de restricciones de operación. Si el comportamiento funcional no es adecuado, el modelo debe revisarse antes de avanzar hacia la exploración en punto fijo, de modo que las discrepancias observadas en etapas posteriores puedan atribuirse directamente a decisiones de representación numérica o implementación.

- **Exploración en punto fijo:** A partir del modelo funcional previamente validado, se utiliza la aplicación *Fixed-Point Converter* para analizar posibles representaciones en punto fijo de las variables del algoritmo. Esta exploración permite estudiar el rango dinámico de las variables y proponer combinaciones de longitud de palabra y longitud fraccional candidatas para la aplicación. Las representaciones obtenidas en esta etapa sirven como base para las simulaciones en punto fijo y para la posterior implementación en C/C++.

- **Simulación en punto fijo:** En este paso se simula el algoritmo utilizando las representaciones seleccionadas durante la etapa de exploración. El objetivo es verificar que el comportamiento del modelo se mantenga dentro de los criterios definidos, aun bajo restricciones de precisión finita. Para ello, los resultados se comparan con la referencia en punto flotante, lo que permite evaluar el efecto de la representación numérica adoptada.

- **Generación de *golden reference*:** Una vez validado el comportamiento del modelo en punto fijo, se genera una *golden reference* compuesta por los estímulos de entrada y las salidas correspondientes producidas por la versión en punto fijo del algoritmo en MATLAB. Esta referencia define el comportamiento numérico esperado para la configuración seleccionada y se utiliza posteriormente para verificar la implementación en C/C++, la co-simulación RTL y la validación en hardware.

Con la *golden reference* definida, el flujo deja de centrarse en la exploración numérica y avanza hacia la descripción en C/C++ compatible con síntesis mediante HLS.

4.1.2. Entorno AMD

En esta etapa, el modelo previamente validado se implementa en C/C++ dentro del entorno AMD y se sintetiza mediante Vitis HLS para obtener una arquitectura destinada a su ejecución en FPGA. El objetivo es, además, verificar en cada etapa del flujo que el comportamiento numérico se mantenga consistente con la referencia obtenida en MATLAB.

Los pasos principales se describen a continuación:

- **Implementación en C/C++ de modelo en punto fijo:**

  El algoritmo en punto fijo previamente validado en MATLAB se reescribe en C/C++ para obtener una descripción compatible con síntesis mediante HLS. En el contexto de esta tesis, el proceso de conversión de MATLAB a C/C++ se realiza manualmente. Si bien MATLAB dispone de herramientas de generación automática o asistida de código, en este trabajo se omite intencionalmente su uso con el propósito de mantener control explícito sobre la estructura del código C/C++ que será utilizado posteriormente para generar hardware.

  La implementación en C/C++ debe reproducir, en la medida de lo posible, la representación de punto fijo seleccionada durante la etapa de exploración, incluyendo formatos y configuraciones. Sin embargo, la correspondencia entre representaciones numéricas por sí sola no garantiza una equivalencia exacta entre la descripción en MATLAB y la implementación en C/C++. Como se discutirá en el resto del capítulo, otros aspectos también pueden afectar el resultado, tales como la estructura del código, el uso de variables intermedias y las diferencias entre las bibliotecas utilizadas en cada entorno. Por esta razón, el objetivo del flujo no es imponer necesariamente resultados idénticos en todas las etapas, sino asegurar que las potenciales diferencias observadas sean explicables, controladas e intencionales.

- **Simulación funcional en C/C++:** Se ejecuta una simulación funcional del código C/C++ utilizando los mismos estímulos considerados en la *golden reference*. Esta etapa permite verificar si la traducción manual del algoritmo conserva el comportamiento numérico esperado respecto del modelo validado en MATLAB, o bien identificar diferencias introducidas por la forma en que las operaciones fueron descritas en C/C++. De esta manera, se separan las discrepancias asociadas a la conversión manual del modelo de aquellas que podrían aparecer posteriormente durante la síntesis o la co-simulación RTL.

- **Inserción de directivas HLS (*pragmas*):** Se agregan directivas al código C/C++ para guiar la síntesis de hardware. Estas directivas permiten modificar aspectos de la arquitectura generada, por ejemplo mediante *pipeline*, desenrollado de ciclos o partición de arreglos. Su propósito es alterar características de implementación, tales como latencia, intervalo de iniciación, uso de recursos o paralelismo, sin modificar la descripción funcional del algoritmo.

- **Síntesis de alto nivel:** En este paso, Vitis HLS transforma el código C/C++ sintetizable en una descripción RTL del algoritmo, la cual puede ser exportada como IP o utilizada como entrada para etapas posteriores de implementación. Además, la herramienta entrega reportes estimados de latencia, intervalo de iniciación, uso de recursos y características de la arquitectura generada. Estos reportes permiten evaluar el impacto de las decisiones de diseño antes de avanzar hacia la implementación física en FPGA.

- **Co-Simulación:** En esta etapa se verifica, mediante simulación, el comportamiento del RTL generado durante la síntesis. Para ello, se emplea un script de prueba o *testbench* que aplica los estímulos de entrada y compara las salidas obtenidas con la *golden reference* generada desde el modelo en punto fijo validado en MATLAB. Esta comparación permite evaluar la consistencia numérica entre el modelo de referencia y la descripción de hardware obtenida mediante HLS.

Una vez que la co-simulación confirma que el RTL mantiene el comportamiento esperado, el flujo avanza hacia la etapa de validación en hardware. En esta etapa, el RTL generado mediante Vitis HLS se integra en un proyecto de implementación para programar la FPGA y verificar el comportamiento del algoritmo directamente en hardware.

4.1.3. Entorno MATLAB + AMD

En el flujo propuesto, se considera el uso de *FIL Wizard* para generar la infraestructura necesaria para comunicar el diseño implementado en la FPGA con un modelo en Simulink. A partir del RTL generado previamente, FIL Wizard configura la infraestructura necesaria para integrar el diseño con Simulink y programarlo en la FPGA, utilizando Vivado durante el proceso. Durante la validación,

Simulink actúa como entorno de prueba, enviando estímulos hacia la FPGA, recibiendo las salidas calculadas por el hardware y permitiendo su comparación con la *golden reference* o con otros modelos de referencia disponibles. Así, la validación incorpora la ejecución del diseño directamente en la FPGA.

El uso de FIL ofrece una ventaja práctica frente a una verificación basada exclusivamente en *testbenches* tradicionales. En un *testbench* descrito en HDL, los estímulos y las condiciones de prueba deben describirse explícitamente para verificar el módulo bajo prueba. Este enfoque resulta adecuado para operaciones aisladas o bloques de complejidad acotada, pero puede requerir un mayor esfuerzo de integración cuando el diseño debe evaluarse dentro de un entorno de simulación más amplio, especialmente si ya se dispone de modelos desarrollados en herramientas de alto nivel.

FIL permite mantener en Simulink el modelo de la planta, la generación de estímulos y la adquisición de resultados, mientras que el controlador o bloque de interés se ejecuta directamente en la FPGA. De esta forma, es posible reutilizar modelos de simulación previamente desarrollados y analizar dentro de un mismo entorno las señales y respuestas obtenidas con la implementación en hardware, sin necesidad de trasladar todo el contexto de prueba a un *testbench* HDL. En los casos de estudio de esta tesis, esta integración permite además evaluar el efecto de la implementación en punto fijo sobre el comportamiento del sistema en lazo cerrado.

A partir de este flujo general, las siguientes secciones detallan las particularidades de cada herramienta y las consideraciones necesarias para reproducir la representación numérica entre los distintos entornos.

4.2. Exploración de representación numérica en MATLAB

En MATLAB, las variables numéricas se representan por defecto en punto flotante de doble precisión, salvo que el usuario especifique otro tipo de dato. Esta característica permite utilizar MATLAB inicialmente como entorno de referencia para describir y simular el comportamiento funcional del algoritmo.

Para estudiar la transición desde esta referencia en punto flotante hacia una representación en punto fijo, MATLAB dispone de tipos de datos específicos. En particular, los objetos `fi` permiten representar variables en punto fijo mediante la definición explícita del formato y de las configuraciones numéricas asociadas. Además, la aplicación *Fixed-Point Converter* permite analizar rangos, comparar señales e identificar formatos candidatos antes de trasladar el diseño a una descripción sintetizable.

4.2.1. Representación de punto fijo mediante objetos `fi`

Los objetos `fi` de MATLAB permiten representar datos de punto fijo mediante la especificación explícita del formato numérico asociado al valor almacenado [38]. Una forma básica de construir un objeto `fi` es:

$$\texttt{fi}(v,\ s,\ W,\ f),$$

donde, tomando en cuenta la convención adoptada en el Capítulo 3 para el formato $Q_{m,n}$, se tiene que

- v: corresponde al valor numérico o variable que contenga un valor numérico que se desea cuantizar.
- s: indicador de signo del formato usado para representar el valor. Para formatos con signo, se utiliza $s = 1$.
- W: longitud total de palabra $W = m + n$.
- f: longitud fraccional n.

Por ejemplo, para representar el valor numérico de una variable v en formato $Q_{5,6}$ se tiene $m = 5$, $n = 6$ y, por tanto, $W = m + n = 11$. En MATLAB, esta configuración se expresa como:

$$v \ \text{ en } \ Q_{5,6} \longleftrightarrow \texttt{fi}(v, 1, 11, 6),$$

Tabla 4.1: Modos de cuantización de `fimath`.

Modo de Cuantización	**Descripción**
Nearest	Redondea al valor representable más cercano; en caso de que el valor fraccional sea exactamente 0,5, siempre redondea hacia el infinito positivo.
Ceiling	Trunca al infinito positivo.
Convergent	Redondea al valor representable más cercano; en caso de que el valor fraccional sea exactamente 0,5, redondea hacia el entero par más cercano.
Zero	Trunca eliminando la parte fraccionaria.
Floor	Trunca al infinito negativo.
Round	Redondea al valor representable más cercano; en caso de que el valor fraccional sea exactamente 0,5, redondea hacia el infinito positivo o negativo dependiendo del signo del valor correspondiente.

Además de especificar directamente estos parámetros en el constructor `fi`, MATLAB permite agrupar las propiedades del formato de almacenamiento mediante un objeto `numerictype`. Este objeto describe el tipo numérico asociado a una variable de punto fijo, incluyendo la presencia de signo, la longitud total de palabra y la longitud fraccional. De esta forma, el mismo formato $Q_{5,6}$ también puede declararse como:

$$\texttt{T = numerictype}(1, 11, 6),$$

para luego construir el objeto de punto fijo mediante:

$$\texttt{fi}(v, \texttt{T}).$$

Esta forma de escritura resulta útil cuando varias variables comparten el mismo formato de representación, ya que permite declarar el tipo numérico una sola vez y reutilizarlo en el código.

De acuerdo con lo establecido en el Capítulo 3, una representación en punto fijo no queda completamente caracterizada solo por su formato $Q_{m,n}$, sino que también es necesario establecer las reglas con que un valor, o el resultado de una operación, se ajusta a dicho formato cuando no puede representarse de manera exacta.

Para completar la especificación, MATLAB utiliza objetos `fimath`. Las propiedades principales de `fimath` consideradas en este trabajo son:

- **`RoundingMethod`:** define la regla de cuantización utilizada cuando un resultado debe ajustarse al formato representable. Los principales modos disponibles se resumen en la Tabla 4.1.
- **`OverflowAction`:** establece el tratamiento frente a desbordamiento, permitiendo seleccionar entre envolvimiento (`Wrap`) o saturación (`Saturate`), de acuerdo con los comportamientos descritos en el Capítulo 3.
- **`ProductMode`** y **`SumMode`:** determinan la precisión utilizada en multiplicaciones y sumas, respectivamente. Si se configuran como `FullPrecision`, MATLAB expande el formato del resultado intermedio para conservar los bits necesarios de la operación. En cambio, si se especifica manualmente la precisión, el resultado se ajusta a las longitudes de palabra y fraccional definidas por el usuario.

El Código 4.1 muestra un ejemplo de declaración de una variable en punto fijo utilizando `numerictype` y `fimath`.

```
% Valores originales en punto flotante
x1 = 1/16;
x2 = 1/256;

% Formato Q_{2,4}: 2 bits enteros incluyendo signo
% y 4 bits fraccionales, con palabra total de 6 bits
T = numerictype(1, 6, 4);

% fimath con redondeo
F_nearest = fimath( ...
    'RoundingMethod', 'Nearest', ...
    'OverflowAction', 'Saturate', ...
    'ProductMode', 'FullPrecision', ...
    'SumMode', 'FullPrecision');

% fimath con truncamiento hacia arriba
F_ceiling = fimath( ...
    'RoundingMethod', 'Ceiling', ...
    'OverflowAction', 'Saturate', ...
    'ProductMode', 'FullPrecision', ...
    'SumMode', 'FullPrecision');

% Conversion de los valores a punto fijo usando Nearest
x1_nearest = fi(x1, T, F_nearest);
x2_nearest = fi(x2, T, F_nearest);

% Conversion de los valores a punto fijo usando Ceiling
x1_ceiling = fi(x1, T, F_ceiling);
x2_ceiling = fi(x2, T, F_ceiling);

% Impresion de resultados
fprintf('=== Comparacion ===\n');
fprintf('Valor         Original     Nearest    Ceiling\n');
fprintf('1/16        %10.8f  %10.8f  %10.8f\n', ...
    x1, double(x1_nearest), double(x1_ceiling));
fprintf('1/256       %10.8f  %10.8f  %10.8f\n', ...
    x2, double(x2_nearest), double(x2_ceiling));
```

Código 4.1: Ejemplo de declaración de un objeto `fi` usando `numerictype` y `fimath`.

```
=== Comparacion ===
Valor         Original     Nearest    Ceiling
1/16        0.06250000  0.06250000  0.06250000
1/256       0.00390625  0.00000000  0.06250000
```

Código 4.2: Salida del Código 4.1.

El Código 4.1 muestra el script utilizado para convertir las variables mediante un mismo objeto `numerictype` asociado al formato $Q_{2,4}$, cuya resolución está dada por $2^{-4} = 1/16$. A partir de la salida generada en el Código 4.2, se observa que el valor almacenado en `x1` permanece inalterado debido a que $x_1 = 1/16$ coincide exactamente con un nivel representable del formato. Por el contrario, como $x_2 = 1/256$ no puede representarse de forma exacta bajo la resolución seleccionada, este valor se cuantiza automáticamente al momento de instanciar el objeto `fi`.

El resultado obtenido para x_2 depende del método de cuantización definido en el objeto `fimath`. Por ejemplo, al utilizar `Nearest`, el valor se aproxima al nivel representable más cercano, mientras que al utilizar `Ceiling` se aproxima al nivel representable superior. De este modo, el ejemplo muestra que, aun manteniendo fijo el `numerictype`, distintas configuraciones de `fimath` pueden producir valores almacenados diferentes cuando el valor original no pertenece exactamente al conjunto de valores representables.

En este caso, las distintas configuraciones de `fimath` se utilizan solo con fines comparativos, por lo que las variables generadas no se operan entre sí. En una implementación completa, distintas variables pueden emplear distintos objetos `numerictype`, de acuerdo con el rango y la precisión requeridos por

cada señal o parámetro. La configuración `fimath`, en cambio, debe mantenerse consistente durante las operaciones: si ambos operandos poseen configuraciones locales, estas deben ser idénticas, si solo uno posee una configuración local, MATLAB utiliza dicha configuración y la propaga al resultado. Por ello, conviene definir explícitamente estas reglas y verificar que las operaciones se ejecuten bajo un comportamiento numérico controlado.

4.2.2. Consideraciones sobre la estructura algorítmica

Una vez presentada la forma en que MATLAB permite definir formatos y configuraciones numéricas asociadas a objetos `fi`, es necesario considerar que el resultado de operaciones entre números representados en punto fijo también puede depender de la forma en que se estructura la secuencia de cálculo. En aritmética real, distintas expresiones algebraicamente equivalentes conducen al mismo resultado. En aritmética de punto fijo, en cambio, la forma de agrupar, descomponer o almacenar operaciones intermedias puede modificar el valor final, debido a que los resultados parciales pueden ajustarse a un formato determinado antes de continuar con el cálculo.

Este efecto se vuelve especialmente relevante cuando una expresión se fragmenta en varias operaciones que requieren almacenar resultados parciales. Cada variable intermedia puede introducir un nuevo punto de ajuste numérico, dependiendo del formato asignado y de las configuraciones utilizadas. Por lo tanto, expresiones algebraicamente equivalentes no necesariamente producen el mismo resultado en una implementación en punto fijo.

Para ilustrar este efecto, el Código 4.3 compara tres formas aritméticamente equivalentes de evaluar una misma expresión. El ejemplo define operaciones entre tres números a, b, y c con formato $Q_{2,4}$ y una configuración Floor, Saturate y FullPrecision para sumas y productos. En el primer caso, se calcula directamente la expresión $ab + ac$. En el segundo caso, se calcula la forma factorizada $a(b + c)$, almacenando primero $b + c$ en el formato $Q_{2,4}$. En el tercer caso, los productos ab y ac se calculan y almacenan explícitamente como variables intermedias en el formato $Q_{2,4}$ antes de realizar la suma final. Además, las operaciones de suma y multiplicación se configuran mediante un objeto `fimath` con `ProductMode` y `SumMode` en modo `FullPrecision`.

```
% 1. Configuracion del tipo de dato y reglas aritmeticas
% Formato Q_{2,4}: 2 bits enteros incluyendo signo y 4 bits fraccionales
T = numerictype(1, 6, 4);

F = fimath( ...
    'RoundingMethod', 'Floor', ...
    'OverflowAction', 'Saturate', ...
    'ProductMode', 'FullPrecision', ...
    'SumMode', 'FullPrecision');

% 2. Inicializacion de variables
% La resolucion minima con 4 bits fraccionales es 1/16 = 0.0625
a = fi(1/16,  T, F);
b = fi(1/16,  T, F);
c = fi(15/16, T, F);

fprintf('=== Valores iniciales en formato Q_{2,4} ===\n');
fprintf('a = %f\n', double(a));
fprintf('b = %f\n', double(b));
fprintf('c = %f\n\n', double(c));

% 3. Caso 1: evaluacion directa de la expresion
r_directo = fi(a*b + a*c, T, F);

fprintf('=== Caso 1: expresion evaluada directamente ===\n');
fprintf('r_directo = fi(a*b + a*c, T, F) = %f\n\n', ...
    double(r_directo));

% 4. Caso 2: forma factorizada
bc = fi(b + c, T, F);
```

```
r_factorizado = fi(a * bc, T, F);

fprintf('=== Caso 2: forma factorizada ===\n');
fprintf('bc = fi(b+c, T, F) = %f\n', double(bc));
fprintf('r_factorizado = fi(a*bc, T, F) = %f\n\n', ...
    double(r_factorizado));

% 5. Caso 3: almacenamiento de productos intermedios
ab = fi(a*b, T, F);
ac = fi(a*c, T, F);
r_fragmentado = fi(ab + ac, T, F);

fprintf('=== Caso 3: productos almacenados previamente ===\n');
fprintf('ab = fi(a*b, T, F) = %f\n', double(ab));
fprintf('ac = fi(a*c, T, F) = %f\n', double(ac));
fprintf('r_fragmentado = fi(ab + ac, T, F) = %f\n\n', ...
    double(r_fragmentado));

% 6. Comparacion final
fprintf('=== Comparacion ===\n');
fprintf('r_directo     = %f\n', double(r_directo));
fprintf('r_factorizado = %f\n', double(r_factorizado));
fprintf('r_fragmentado = %f\n', double(r_fragmentado));
```

Código 4.3: Ejemplo de efecto de la estructura algorítmica en punto fijo.

La ejecución del script anterior entrega la siguiente salida en la ventana de comandos de MATLAB:

```
=== Valores iniciales en formato Q_{2,4} ===
a = 0.062500
b = 0.062500
c = 0.937500

=== Caso 1: expresion evaluada directamente ===
r_directo = fi(a*b + a*c, T, F) = 0.062500

=== Caso 2: forma factorizada ===
bc = fi(b+c, T, F) = 1.000000
r_factorizado = fi(a*bc, T, F) = 0.062500

=== Caso 3: productos almacenados previamente ===
ab = fi(a*b, T, F) = 0.000000
ac = fi(a*c, T, F) = 0.000000
r_fragmentado = fi(ab + ac, T, F) = 0.000000

=== Comparacion ===
r_directo     = 0.062500
r_factorizado = 0.062500
r_fragmentado = 0.000000
```

Código 4.4: Salida generada por el script del Código 4.3.

Los resultados muestran que los Casos 1 y 2 entregan el mismo valor final, mientras que el Caso 3 produce un resultado distinto. Esta diferencia no se debe a un cambio en los valores de entrada ni en el formato numérico utilizado, sino a la forma en que se estructuran las operaciones y se almacenan los resultados intermedios.

Caso 1: Evaluación directa. En el primer caso, la expresión se evalúa directamente como:

$$r_{\text{directo}} = \texttt{fi}(ab + ac, T, F).$$

Los productos exactos que aparecen en la expresión original son:

$$ab = \frac{1}{256} = 0{,}00390625, \qquad ac = \frac{15}{256} = 0{,}05859375.$$

Ambos valores son menores que la resolución del formato $Q_{2,4}$ ($\Delta = 2^{-4} = 1/16 = 0{,}0625$), por lo que no podrían representarse individualmente en dicho formato sin pérdida de información. Sin embargo, debido a que las operaciones se realizan con `ProductMode` y `SumMode` configurados como `FullPrecision`, MATLAB utiliza formatos intermedios con la precisión necesaria para conservar el resultado completo de las operaciones entre los objetos `fi` asociados, evitando que estos sean reducidos inmediatamente al formato de los operandos. De este modo, la suma se realiza con precisión completa:

$$ab + ac = \frac{1}{256} + \frac{15}{256} = \frac{16}{256} = \frac{1}{16}.$$

El resultado $\frac{1}{16}$ debe reducirse al formato $Q_{2,4}$, pero este valor final sí es representable exactamente en $Q_{2,4}$, por lo que se obtiene:

$$r_{\text{directo}} = 0{,}0625.$$

Caso 2: Evaluación factorizada. En esta estructura de cálculo, la única variable intermedia almacenada antes de la multiplicación final corresponde a $b + c$:

$$b + c = \frac{1}{16} + \frac{15}{16} = 1,$$

como este valor es representable exactamente en el formato $Q_{2,4}$, su almacenamiento no introduce error de cuantización. Luego, la multiplicación:

$$a(b + c) = \frac{1}{16} \cdot 1 = \frac{1}{16}$$

también produce un resultado exactamente representable en el formato de destino. Por esta razón, se obtiene el resultado exacto:

$$r_{\text{factorizado}} = 0{,}0625.$$

Caso 3: Evaluación fragmentada con almacenamiento intermedio. En el tercer caso, los productos parciales se calculan y almacenan explícitamente como variables `fi` antes de realizar la suma:

`ab = fi(a·b,T,F),  ac = fi(a·c,T,F).`

Estas asignaciones introducen puntos de cuantización explícitos en la secuencia de cálculo, ya que cada producto debe ajustarse al formato $Q_{2,4}$ antes de continuar con la operación siguiente. Como los productos parciales son menores que la resolución del formato, no son directamente representables. Bajo el método de cuantización `Floor`, ambos valores positivos se cuantizan al nivel representable inmediatamente inferior:

$$\frac{1}{256} \to 0, \qquad \frac{15}{256} \to 0.$$

Por lo tanto, la suma final ya no se realiza sobre los productos exactos ni sobre resultados temporales en precisión completa, sino sobre sus versiones previamente cuantizadas:

$$r_{\text{fragmentado}} = 0 + 0 = 0.$$

Este último caso muestra que, en aritmética de punto fijo, no basta con verificar que dos implementaciones representen la misma expresión matemática o utilicen el mismo formato numérico. También es necesario considerar la secuencia de cálculo, las variables intermedias introducidas por la implementación y los puntos en que los resultados intermedios se ajustan a un formato y la configuración utilizada para las operaciones. Aunque el caso analizado es deliberadamente simple, ilustra un efecto que puede volverse más relevante en algoritmos de mayor tamaño, como los controladores digitales evaluados en el siguiente capítulo. En estos casos, las diferencias asociadas al manejo de resultados intermedios pueden ser más difíciles de aislar y pueden acumularse a lo largo de múltiples etapas de cálculo.

4.2.3. Fixed-Point Converter y análisis de rangos

Dentro del flujo propuesto, *Fixed-Point Converter* se utiliza como apoyo para la transición desde una descripción funcional en punto flotante hacia una representación en punto fijo. La herramienta permite ejecutar el algoritmo con un conjunto de pruebas, registrar los rangos numéricos alcanzados por cada variable y proponer tipos de datos candidatos para cada una de ellas a partir de los valores obtenidos durante la ejecución. Estas propuestas pueden generarse con las opciones por defecto de la herramienta o ajustarse mediante parámetros de conversión definidos por el usuario. De esta manera, automatiza parte del proceso de exploración numérica, aunque la selección final de formatos sigue dependiendo del análisis del diseñador y de los criterios de precisión requeridos por la aplicación.

El proceso de conversión requiere dos archivos principales: una función de MATLAB que contiene el algoritmo u operación que se desea convertir, y un script de prueba, o *testbench*, encargado de llamar a dicha función con un conjunto definido de entradas. Esta separación es importante, ya que la herramienta analiza e instrumenta la función objetivo, mientras que el *testbench* se utiliza para evaluar su comportamiento con señales, vectores de datos o casos representativos de operación. En consecuencia, la función que será convertida debe escribirse de forma compatible con las restricciones del flujo de conversión, evitando construcciones o funciones de MATLAB no soportadas por los procesos de análisis de rangos y generación de código [43].

El *testbench* cumple un rol central en la calidad de la propuesta generada. Durante su ejecución, *Fixed-Point Converter* registra los valores mínimos, máximos y la frecuencia de los valores alcanzados por entradas, salidas y variables internas. A partir de esta información, la herramienta informa los rangos observados y propone formatos en punto fijo para las variables consideradas. Sin embargo, estos rangos no representan necesariamente todos los valores posibles del algoritmo, sino solo aquellos que aparecen durante las pruebas realizadas. Por ello, si los estímulos no cubren adecuadamente el rango de operación de la aplicación, las configuraciones propuestas pueden subestimar los bits enteros requeridos.

Antes de generar la propuesta de tipos de datos, el usuario puede definir distintos criterios de conversión. Una opción consiste en fijar la longitud total de palabra y permitir que la herramienta proponga la longitud fraccional para cada variable, buscando cubrir los rangos observados con el número de bits disponible. Otra opción consiste en fijar la longitud fraccional y permitir que la herramienta proponga la longitud total de palabra requerida para representar los rangos registrados. Esta segunda alternativa puede generar longitudes de palabra distintas entre variables.

Además de proponer formatos para las variables, la herramienta permite definir la configuración de las operaciones mediante las propiedades asociadas a `fimath`, incluyendo el método de cuantización, el tratamiento de desbordamiento y la precisión utilizada en sumas y multiplicaciones.

Una vez definidas las opciones de conversión, *Fixed-Point Converter* propone tipos de datos candidatos, indicando propiedades como signo, longitud total de palabra y longitud fraccional. Estos tipos pueden aceptarse directamente, modificarse manualmente o recalcularse después de ajustar las opciones de conversión. Si se habilita la visualización de información durante el análisis, la herramienta puede mostrar histogramas de rango dinámico, los cuales permiten relacionar los valores observados con intervalos de magnitud asociados a potencias de dos. Esta visualización ayuda a interpretar la cantidad de bits enteros requerida por una variable, aunque no reemplaza la validación numérica posterior.

Como resultado del proceso, se obtiene una versión en punto fijo de la función original. En esta versión, las variables del algoritmo pasan a representarse mediante tipos de datos de punto fijo, de acuerdo con los formatos definidos durante la etapa de conversión. Además, se genera un archivo auxiliar, o *wrapper*, que convierte las entradas en punto flotante utilizadas por el *testbench* hacia los tipos de punto fijo definidos para la función convertida. Con estos archivos, el usuario puede simular el modelo convertido y comparar sus resultados con los obtenidos por la función original en punto flotante, ya sea dentro de la herramienta o mediante scripts propios. Esta comparación permite verificar que el error introducido por la conversión se mantenga dentro de los márgenes admisibles para la aplicación, y puede requerir varias iteraciones, ya que cubrir los rangos observados no garantiza por sí solo que la precisión resultante sea suficiente.

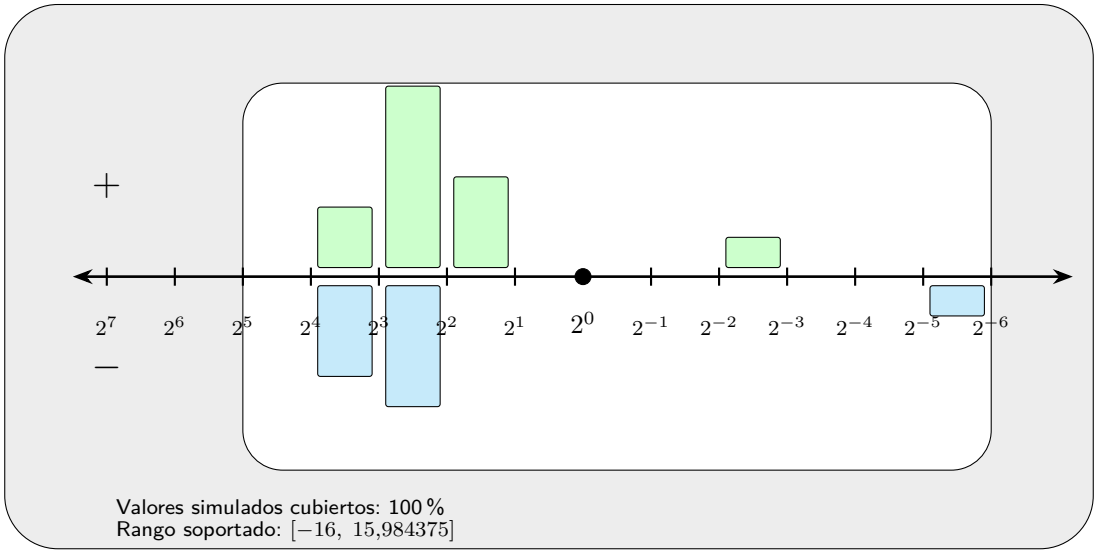


Figura 4.2: Representación del histograma de rango dinámico para los valores registrados de la variable X. La figura reproduce de forma ordenada el tipo de visualización entregada por la herramienta, con el propósito de describir cómo se organiza la información en intervalos de magnitud asociados a potencias de 2.

Ejemplo ilustrativo de selección de formatos de punto fijo.

Para ilustrar intuitivamente el uso de *Fixed-Point Converter* en el análisis de rangos, considérese una variable X cuyos valores fueron registrados durante la ejecución de un *testbench*:

$$X = \{5{,}21;\ -6{,}61;\ 4{,}29;\ 9{,}06;\ -5{,}50;\ 2{,}25;\ -9{,}92;\ 4{,}43;\ -0{,}029;\ 8{,}35;\\ 2{,}67;\ -6{,}03;\ 5{,}42;\ -9{,}58;\ 0{,}24;\ 3{,}70;\ 6{,}25;\ -4{,}16;\ 4{,}97;\ -8{,}23\}.$$

Esta variable puede corresponder a una entrada, una salida, una variable interna o una expresión intermedia del algoritmo instrumentado.

A partir de estos datos, la herramienta puede generar un histograma de rango dinámico, como el mostrado en la Figura 4.2. Este tipo de histograma agrupa las muestras observadas en intervalos de magnitud asociados a potencias de dos. La altura de cada barra indica, de manera relativa, cuántos valores fueron observados dentro de cada intervalo. Por lo tanto, el histograma no muestra directamente el valor exacto de cada muestra, sino la distribución de las magnitudes alcanzadas durante la simulación.

La representación del histograma es útil porque los rangos representables de una variable en punto fijo también quedan determinados por potencias de dos. En particular, la cantidad de bits enteros define la magnitud máxima que puede representarse sin desbordamiento. Por ello, el histograma permite identificar rápidamente si las muestras observadas requieren un mayor número de bits enteros, o si el rango disponible resulta suficiente para cubrir los datos registrados.

Para facilitar la interpretación del ejemplo, la Tabla 4.2 resume en valores absolutos cuántas muestras positivas y negativas aparecen en cada intervalo de magnitud. En el caso de los valores negativos, la asignación al intervalo se realiza según su valor absoluto, ya que el histograma agrupa las muestras en función de $|X|$. Así, los valores registrados pueden separarse como:

$$X^+ = \{0{,}24;\ 2{,}25;\ 2{,}67;\ 3{,}70;\ 4{,}29;\ 4{,}43;\ 4{,}97;\ 5{,}21;\ 5{,}42;\ 6{,}25;\ 8{,}35;\ 9{,}06\},$$
$$X^- = \{-0{,}029;\ -4{,}16;\ -5{,}50;\ -6{,}03;\ -6{,}61;\ -8{,}23;\ -9{,}58;\ -9{,}92\}.$$

La presencia de muestras positivas y negativas indica que la variable requiere una representación

Tabla 4.2: Distribución de la variable X en intervalos de magnitud definidos por potencias de dos.

Límite superior del intervalo	**Rango de $\|X\|$**	**N.° de valores positivos**	**N.° de valores negativos**
2^{-5}	$2^{-6} < \|X\| \leq 2^{-5}$	0	1
2^{-2}	$2^{-3} < \|X\| \leq 2^{-2}$	1	0
2^{2}	$2^{1} < \|X\| \leq 2^{2}$	3	0
2^{3}	$2^{2} < \|X\| \leq 2^{3}$	6	4
2^{4}	$2^{3} < \|X\| \leq 2^{4}$	2	3

con signo. Además, los valores de mayor magnitud se ubican en el intervalo:

$$2^3 < |X| \leq 2^4,$$

es decir,

$$8 < |X| \leq 16.$$

Para el formato $Q_{m,n}$ se utilizan m bits para la parte entera, incluyendo el bit de signo. Para los valores registrados en este ejemplo se requiere al menos $m = 5$, ya que:

$$[-2^{m-1};\, 2^{m-1} - 1] = [-2^4;\, 2^4 - 1] = [-16;\, 15].$$

Este rango permite cubrir los valores 9,06 y −9,92, que corresponden al máximo y mínimo registrados para X. En cambio, si se utilizara $m = 4$, el rango entero sería:

$$[-8;\, 7],$$

por lo que dichos valores quedarían fuera del intervalo representable y podrían producir desbordamiento.

El histograma permite orientar la selección de bits enteros, pero no determina por sí solo la cantidad de bits fraccionales necesaria. Los bits fraccionales definen la resolución del formato, es decir, la separación entre dos valores representables consecutivos. Por ejemplo, si se seleccionan $n = 6$ bits fraccionales, la resolución queda dada por:

$$\Delta = 2^{-6} = 0{,}015625.$$

A partir de esta resolución, si se adopta el formato candidato $Q_{5,6}$ para este ejemplo, el rango representable queda dado por:

$$[-2^{m-1};\, 2^{m-1} - 2^{-n}] = [-2^4;\, 2^4 - 2^{-6}] = [-16;\, 15{,}984375].$$

Se puede observar que, para el formato seleccionado, todos los valores registrados para X se encuentran dentro de este intervalo. Por lo tanto, el formato candidato cubre el 100 % de los datos observados durante la ejecución de prueba. Esta afirmación debe interpretarse de manera acotada, solo indica que, para las entradas utilizadas en el *testbench*, no se registraron valores fuera del rango representable. Si se evalúan otras entradas, condiciones iniciales o escenarios de operación, podrían aparecer valores no considerados durante esta exploración.

Junto con verificar la cobertura del rango observado, la elección de n debe considerar el error de cuantización que la aplicación puede tolerar. Por ello, no basta con comprobar que el formato evita desbordamientos para los datos del *testbench*, también es necesario simular el algoritmo convertido y comparar su salida con la referencia en punto flotante.

En síntesis, *Fixed-Point Converter* proporciona una etapa de exploración asistida para proponer formatos de punto fijo a partir de rangos observados. El histograma de rango dinámico ayuda a interpretar la cantidad de bits enteros requerida, mientras que la selección de bits fraccionales debe validarse mediante el error numérico observado en la salida del algoritmo.

Una vez validado el modelo en punto fijo, el siguiente paso del flujo consiste en trasladar manualmente las representaciones numéricas seleccionadas hacia una descripción sintetizable en C/C++ para su implementación mediante Vitis HLS.

4.3. Implementación en punto fijo mediante Vitis HLS

Vitis HLS se incorpora al flujo propuesto como el entorno encargado de generar una descripción RTL sintetizable a partir de código C/C++. En este trabajo, el traslado del modelo validado en MATLAB hacia C/C++ se realiza de manera manual como parte del flujo propuesto. Esta decisión permite mantener un control fino sobre la forma en que se implementa el algoritmo, considerando que una descripción orientada a hardware puede requerir una estructura distinta a la de un código escrito únicamente con fines de simulación en software. En particular, la organización de operaciones, el uso de tipos de datos, el manejo de resultados intermedios y la inserción posterior de directivas HLS pueden influir tanto en el comportamiento numérico como en la arquitectura generada.

En los casos de estudio analizados más adelante, esta etapa toma como base códigos en C previamente desarrollados y validados en punto flotante en trabajos anteriores, sobre los cuales se incorporan las configuraciones de punto fijo seleccionadas durante la exploración numérica. A partir de esta descripción, Vitis HLS permite simular el comportamiento del código, aplicar directivas HLS y generar el RTL correspondiente para su posterior verificación e implementación física.

Los aspectos de Vitis HLS asociados al diseño de arquitecturas de hardware, uso de directivas de optimización y generación de interfaces han sido abordados en un trabajo previo desarrollado en el mismo entorno de investigación [25]. Por esta razón, dichos aspectos quedan fuera del alcance de esta tesis. En cambio, esta sección se concentra en los aspectos necesarios para reproducir en C/C++ el comportamiento numérico del modelo de referencia, considerando los tipos de datos `ap_fixed`, su equivalencia con la notación $Q_{m,n}$ y los modos de cuantización y desbordamiento.

4.3.1. Tipos `ap_fixed` y equivalencia con la notación Q

Una vez definidas las representaciones de punto fijo en MATLAB, el siguiente paso del flujo propuesto consiste en trasladar dichas decisiones hacia una descripción C/C++ compatible con Vitis HLS. Para este propósito, Vitis HLS proporciona tipos de datos de precisión arbitraria que permiten representar números enteros y de punto fijo con anchos de palabra definidos por el usuario. En el caso de números de punto fijo, se debe incluir la librería correspondiente mediante el encabezado:

```
1 #include "ap_fixed.h"
```

Los tipos principales para representar valores de punto fijo son `ap_fixed` y `ap_ufixed`. El primero se utiliza para representaciones con signo, mientras que el segundo se utiliza para representaciones sin signo. Para el caso con signo, la forma general del tipo es:

$$\texttt{ap_fixed}\langle W,\ I,\ C,\ O,\ N\rangle,$$

donde los parámetros principales se interpretan de la siguiente manera:

W: Longitud total de palabra. En la convención $Q_{m,n}$ utilizada en el Capítulo 3, corresponde a $W = m + n$.

I: Cantidad de bits enteros, incluyendo el bit de signo. Para un formato $Q_{m,n}$, este parámetro corresponde a $I = m$.

C: Modo de cuantización aplicado cuando un resultado debe ajustarse a una cantidad finita de bits fraccionales. Cumple un rol equivalente a la propiedad `RoundingMethod` definida en MATLAB mediante `fimath`. Los modos disponibles se listan en la Tabla 4.3 .

O: Modo de desbordamiento aplicado cuando un resultado excede el rango representable por el formato destino. Cumple un rol equivalente a la propiedad `OverflowAction` definida en MATLAB mediante `fimath`. Los modos disponibles se listan en la Tabla 4.4.

Tabla 4.3: Modos de cuantización seleccionados de `ap_fixed` en Vitis HLS.

Modo	Descripción
`AP_TRN`	Trunca el resultado hacia $-\infty$.
`AP_TRN_ZERO`	Trunca el resultado aproximándose siempre hacia cero.
`AP_RND`	Redondea al valor representable más cercano. En caso de empate, redondea hacia $+\infty$.
`AP_RND_ZERO`	Redondea al representable más cercano. En caso de empate, redondea en dirección hacia el cero.
`AP_RND_INF`	Redondea al representable más cercano. En caso de empate, redondea alejándose del cero (hacia $+\infty$ o $-\infty$ según el signo).
`AP_RND_MIN_INF`	Redondea al representable más cercano. En caso de empate, redondea hacia $-\infty$.
`AP_RND_CONV`	Aplica redondeo convergente (hacia el valor par más cercano en empates).

Tabla 4.4: Modos de desbordamiento seleccionados de `ap_fixed` en Vitis HLS.

Modo	Descripción
`AP_WRAP`	Conserva los bits menos significativos del resultado, produciendo un comportamiento de envolvimiento.
`AP_WRAP_SM`	Conserva el signo original del resultado y aplica el comportamiento de envolvimiento únicamente sobre la magnitud.
`AP_SAT`	Satura el resultado, asignando el valor máximo representable ante un desbordamiento positivo, o el mínimo representable ante uno negativo.
`AP_SAT_ZERO`	Asigna un valor de cero al resultado ante cualquier evento de desbordamiento, independientemente del signo original.
`AP_SAT_SYM`	Satura el resultado de forma simétrica respecto a cero. Limita el valor mínimo negativo a la magnitud del máximo positivo, eliminando la asimetría propia del complemento a dos.

N: Parámetro opcional asociado a algunos modos de saturación. Permite ajustar detalles específicos del comportamiento de desbordamiento, pero no es necesario especificarlo en las configuraciones básicas consideradas en este trabajo.

Por ejemplo, para representar en Vitis HLS un formato $Q_{5,6}$, se utiliza la equivalencia:

$$Q_{5,6} \longleftrightarrow \texttt{ap_fixed}\langle 11, 5\rangle,$$

Es importante notar que el segundo parámetro de `ap_fixed` corresponde a la cantidad de bits enteros, incluyendo el bit de signo, y no a la longitud fraccional. De manera análoga, para una representación sin signo se utiliza `ap_ufixed`. En este caso, la correspondencia con un formato $U_{m,n}$ se obtiene usando nuevamente $W = m + n$ e $I = m$, pero sin reservar un bit para signo.

Además del ancho total de palabra y de la posición del punto binario, es necesario especificar el comportamiento frente a cuantización y desbordamiento. En MATLAB, estas reglas se definen mediante las propiedades del objeto `fimath` asociado a las variables `fi`. En Vitis HLS, en cambio, estas reglas forman parte del tipo `ap_fixed` o `ap_ufixed`. En ausencia de una configuración explícita, Vitis HLS utiliza modos de cuantización y desbordamiento por defecto. En general, los modos por defecto pueden diferir de los utilizados en otras herramientas. Por lo tanto, en este trabajo se declaran explícitamente para mantener la consistencia numérica entre etapas del flujo.

Por ejemplo, si se desea reproducir una configuración de MATLAB para un tipo $Q_{m,n}$, `RoundingMethod = Floor` y `OverflowAction = Saturate`, una declaración compatible en Vitis HLS puede escribirse como:

Tabla 4.5: Correspondencias prácticas entre configuraciones de MATLAB y Vitis HLS.

Comportamiento	**MATLAB `fimath`**	**Vitis HLS `ap_fixed`**
Truncamiento hacia $-\infty$	`Floor`	`AP_TRN`
Truncamiento hacia cero	`Zero`	`AP_TRN_ZERO`
Redondeo al más cercano	`Nearest`	`AP_RND`
Redondeo convergente	`Convergent`	`AP_RND_CONV`
Redondeo con empates alejándose de cero	`Round`	`AP_RND_INF`
Envolvimiento	`Wrap`	`AP_WRAP`
Saturación	`Saturate`	`AP_SAT`

```
1 typedef ap_fixed<11, 5, AP_TRN, AP_SAT> data_t;
```

La Tabla 4.5 resume las correspondencias entre configuraciones de MATLAB y Vitis HLS. Estas se emplean como referencia práctica para configurar implementaciones equivalentes en las siguientes secciones.

En Vitis HLS, los resultados intermedios de operaciones entre variables `ap_fixed` no se reducen necesariamente al ancho de palabra de los operandos. Para estos tipos de datos, la librería determina el tipo del resultado intermedio de acuerdo con la operación realizada y con los formatos de entrada. Por lo tanto, el ancho del resultado puede considerar el crecimiento de bits requerido por la operación, pero dicho crecimiento se define de manera estática a partir de los tipos conocidos en C++ antes de la síntesis. Esto lo diferencia de una interpretación dinámica de crecimiento de palabra como ocurre con el modo `ProductMode = FullPrecision` de MATLAB, ya que en Vitis HLS todos los anchos quedan determinados durante la compilación y síntesis del código, y no cambian dinámicamente durante la ejecución en hardware. Finalmente, cuando un resultado intermedio se almacena en una variable `ap_fixed` declarada explícitamente, el formato de dicha variable determina la cuantización y el tratamiento de desbordamiento aplicados al valor almacenado.

Además de reproducir los formatos y configuraciones, es necesario mantener una estructura de cálculo compatible entre MATLAB y C/C++. Como se mostró en la Sección 4.2.2, expresiones algebraicamente equivalentes pueden producir resultados distintos en punto fijo si cambia el orden de evaluación o si se introducen puntos de almacenamiento intermedio. Por ello, el traslado hacia Vitis HLS debe considerar también la secuencia de operaciones y las asignaciones que determinan cuándo un resultado se ajusta a un formato de destino.

En resumen, los tipos `ap_fixed` y `ap_ufixed` permiten reproducir en Vitis HLS la representación numérica definida previamente en MATLAB. Para obtener un comportamiento equivalente entre ambos entornos, deben corresponder el formato, la configuración de las operaciones y la estructura de cálculo. Si alguna de estas condiciones difiere, puede no ser posible obtener equivalencia exacta entre etapas. En esos casos, el objetivo del flujo no es imponer necesariamente igualdad bit a bit, sino asegurar que las diferencias observadas sean explicables, trazables y acotadas de acuerdo con los criterios de precisión de la aplicación.

4.4. Caso de estudio: Implementación de operación de producto punto

En esta sección se aplica el flujo propuesto a una operación de producto punto entre dos vectores, con el objetivo de analizar cómo decisiones sobre la representación en punto fijo afectan el resultado numérico y su posterior reproducción en otro entorno de implementación.

Para dos vectores $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$, el producto punto se define como

$$z = \sum_{i=1}^{N} x_i y_i.$$

El producto punto aparece de forma recurrente en algoritmos de control y optimización, por ejemplo como parte de productos matriz-vector y acumulaciones presentes en métodos como ADMM. Por ello,

se utiliza como un caso simple y representativo para aplicar el flujo propuesto.

Su estructura permite analizar de forma acotada el efecto de la precisión finita sobre una secuencia de multiplicaciones y una acumulación. En particular, se estudia cómo el formato, las configuraciones de cuantización y desbordamiento, y el tratamiento de los resultados intermedios afectan la salida final, antes de aplicar el flujo a los controladores presentados en el capítulo siguiente.

El desarrollo del ejemplo se organiza siguiendo las etapas del flujo propuesto. Primero, se realiza la exploración y conversión a punto fijo en MATLAB, evaluando distintos formatos y configuraciones. Luego, la representación numérica seleccionada se traslada a Vitis HLS mediante tipos `ap_fixed`, con el propósito de comparar el comportamiento numérico entre implementaciones. En esta etapa también se sintetizan e implementan distintas alternativas de representación numérica para evaluar su efecto sobre el uso de recursos en FPGA. Finalmente, se considera la validación en hardware mediante *FPGA-in-the-Loop*, donde el diseño generado se ejecuta en la FPGA y se compara con la *golden reference* generada en MATLAB para la representación numérica seleccionada.

4.4.1. Archivos de entrada para la conversión

El punto de partida del proceso de implementación corresponde a una descripción funcional en punto flotante del algoritmo. En este caso, la función de MATLAB que posteriormente se convertirá a punto fijo se muestra en el Código 4.5.

```
function [result] = producto_punto(x, y)
% Funcion que calcula el producto punto entre dos vectores x e y
% La operacion se realiza en dos etapas:
% 1) calculo de los productos elemento a elemento
% 2) acumulacion de los productos parciales

% Numero de elementos del vector de entrada
% Se asume que x e y tienen la misma longitud
N = length(x);

% Inicializacion de la variable que almacena el resultado final
result = 0;

% Inicializacion del vector que almacena los productos parciales
mult = zeros(1, N);

% Calculo de los productos elemento a elemento
for i = 1:N
    mult(i) = x(i) * y(i);
end

% Suma acumulativa de los productos parciales
for u = 1:N
    result = result + mult(u);
end

end
```

Código 4.5: Función MATLAB en punto flotante para el producto punto.

Además de la función que contiene el algoritmo, *Fixed-Point Converter* requiere un script de prueba encargado de ejecutar la función con entradas definidas por el usuario. Durante esta ejecución, la herramienta registra los rangos alcanzados por las variables.

En este ejemplo se emplean vectores de 100 elementos como caso de prueba. El Código 4.6 muestra el script utilizado.

```
N = 100; % Numero de elementos de los vectores de entrada
% Semilla del generador aleatorio
% Permite generar siempre los mismos vectores de prueba
```

```
4  rng(42);
5
6  % Generacion de vectores aleatorios en punto flotante
7  % Los valores generados se encuentran en el intervalo [0, 1)
8  A = rand(1, N);
9  B = rand(1, N);
10
11 % Evaluacion de la funcion de producto punto
12 y = producto_punto(A,B);
13
14 % Visualizacion de entradas y salida
15 disp(A);
16 disp(B);
17 disp(y);
```

Código 4.6: Script de prueba para el producto punto.

Los rangos registrados durante la exploración dependen directamente de las entradas generadas por este script. Al configurar una semilla fija, se obtienen los mismos vectores en cada ejecución, lo que permite repetir la prueba bajo las mismas condiciones. Además, dado que `rand` genera valores en el intervalo [0; 1), los tipos de datos propuestos y los errores obtenidos deben interpretarse en función de este conjunto de entradas y no como una caracterización de todos los valores posibles del algoritmo.

4.4.2. Exploración de formatos de punto fijo en MATLAB

A partir de la función `producto_punto` y del script de prueba, se realiza la conversión del producto punto a punto fijo en MATLAB. Para analizar el efecto de las decisiones de conversión, se consideran tres casos de exploración.

- **Caso 1 - Conversión manteniendo opciones por defecto:** se utiliza la propuesta de formatos generada por *Fixed-Point Converter*, sin ajustar manualmente el ancho de palabra ni los modos de cuantización y desbordamiento.
- **Caso 2 - Conversión usando anchos de palabra especificados por el usuario:** se modifica el ancho de palabra propuesto por la herramienta a partir del análisis de rangos de las variables del algoritmo. La configuración de cuantización y desbordamiento se mantiene por defecto.
- **Caso 3 - Conversión usando anchos de palabra y modos de cuantización y desbordamiento especificados por el usuario:** se conservan los formatos especificados en el caso anterior, pero además se especifica explícitamente el comportamiento numérico asociado a la cuantización y el tratamiento de desbordamiento.

Los tres casos se evalúan con el mismo conjunto de entradas, calculando el producto punto para 50 pares de vectores de entrada de 100 elementos cada uno. Para cada caso, se compara el valor del producto punto obtenido en punto fijo con la referencia correspondiente en punto flotante. El objetivo de esta evaluación es ilustrar cómo las decisiones sobre el formato y la configuración de las operaciones inciden en el error numérico y, posteriormente, en el costo de implementación.

Para la conversión del producto punto que se ve en el Código 4.5, las variables de interés son las entradas `x` e `y`, la variable interna `mult` y la salida `result`. Estas variables se consideran para el análisis de los tres casos evaluados.

Caso 1: Conversión manteniendo opciones por defecto

En el primer caso se utiliza *Fixed-Point Converter* sin modificar sus opciones de conversión. En la versión de MATLAB utilizada, la herramienta propone por defecto tipos de datos manteniendo fija una longitud total de palabra $W = 16$ bits. Para cada variable, se asigna un número de bits enteros necesarios para cubrir el rango observado en la ejecución del script de prueba, y los bits restantes se

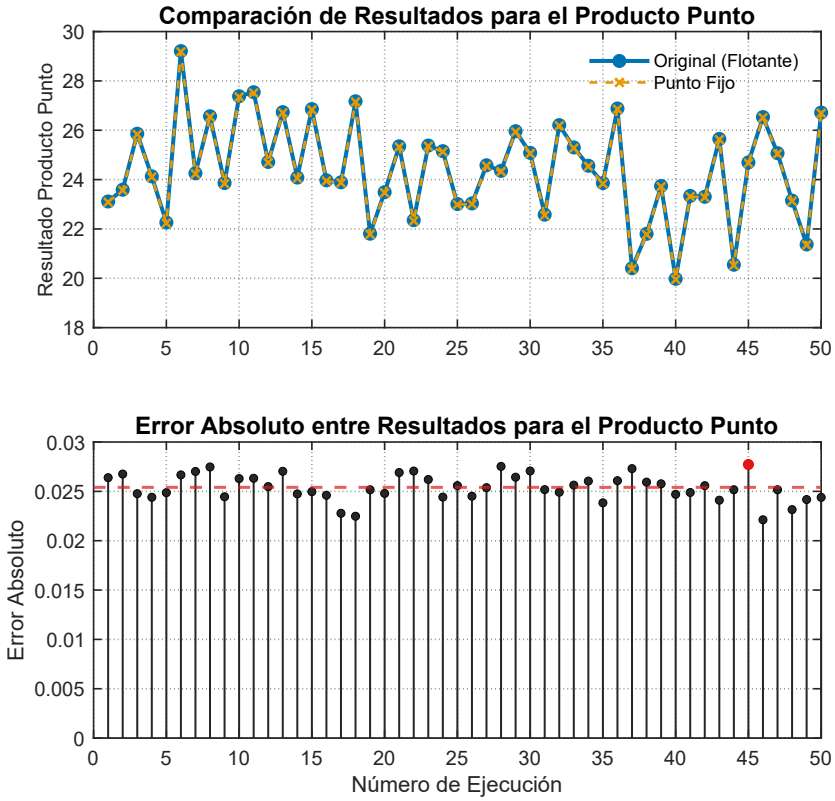


Figura 4.3: Error absoluto entre el producto punto en punto flotante y la versión en punto fijo obtenida con las opciones por defecto de *Fixed-Point Converter*.

asignan a la parte fraccionaria. Además, se mantienen los modos por defecto, correspondientes a `Floor` para cuantización y `Wrap` para desbordamiento.

Para el script de prueba, las variables x, y y mult son números positivos menores que 1, por lo que la herramienta propone el formato $U_{0,16}$ y, para los datos de entrada, el valor del producto punto es siempre menor a 32, por lo que la herramienta propone el formato $U_{5,11}$ para la salida result.

La Figura 4.3 muestra el error absoluto entre la versión en punto flotante y la versión en punto fijo obtenida para la configuración por defecto para las 50 operaciones de producto punto. Para este caso se obtiene un error absoluto máximo de 0,027720 y un error absoluto medio de 0,025408.

Caso 2: Conversión con formatos ajustados a partir del análisis de rangos

En el segundo caso se ajustan manualmente los formatos numéricos a partir del análisis de rango mediante histogramas. A diferencia del caso anterior, el diseñador interviene en la selección de bits con el objetivo de reducir la longitud de palabra utilizada.

La Figura 4.4 muestra los histogramas generados por la herramienta, donde se observa que el rango de las entradas x e y se encuentra en [0;1) y para la salida result es [0;32). Por ello, se requieren 0 bits enteros para las entradas y 5 bits enteros para la salida. En este caso, para las entradas, se restringe la parte fraccionaria a 11 bits en lugar de 16. Para la salida, se restringe a 10 bits en lugar de 11. Estos formatos mantienen el rango observado, pero reducen la resolución respecto del Caso 1.

En particular, el histograma de la Figura 4.4b muestra que el formato seleccionado para la salida cubre con margen el rango registrado durante las pruebas. Sin embargo, esta cobertura solo permite verificar que los valores evaluados pueden representarse sin desbordamiento y no determina si la resolución fraccional es suficiente para conservar la fidelidad numérica del resultado. Este aspecto resulta especialmente relevante en una variable acumuladora como la salida del producto punto, donde



(a) Histograma de entradas para un formato $U_{0,11}$.

(b) Histograma de salida para un formato $U_{5,10}$.

Figura 4.4: Histogramas utilizados para seleccionar formatos de punto fijo en el producto punto.

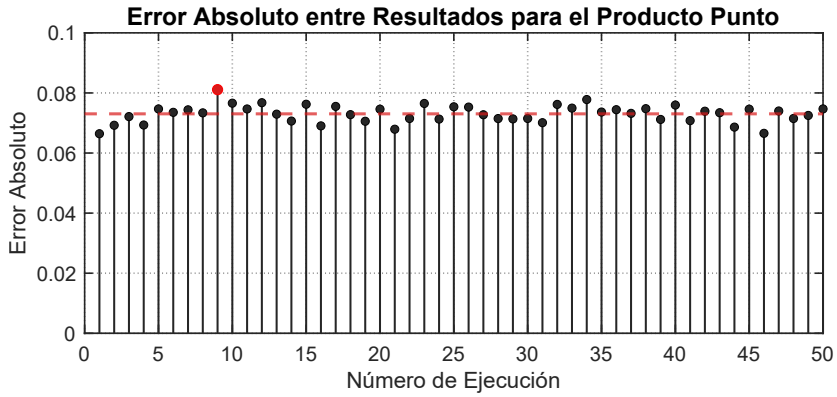


Figura 4.5: Error absoluto entre el producto punto en punto flotante y la versión en punto fijo con formatos ajustados a partir del análisis de rangos.

los efectos de la cuantización pueden propagarse a través de las sumas sucesivas.

Con los formatos seleccionados, se realiza la comparación entre la versión en punto flotante y la versión en punto fijo. La Figura 4.5 muestra el error absoluto obtenido. Bajo esta configuración se obtiene un error absoluto máximo de 0,081095 y un error absoluto medio de 0,073044, valores superiores a los obtenidos previamente, tal como se esperaba debido a la menor precisión.

Para ilustrar que la cobertura del rango no es suficiente como criterio de selección, se repite la prueba reduciendo la cantidad de bits fraccionales de la variable que almacena el resultado, al utilizar el formato $U_{5,7}$, el histograma de la Figura 4.6 mantiene una cobertura del 100 %, mientras que la comparación con la referencia en punto flotante, presentada en la Figura 4.7, entrega un error absoluto máximo de 0,493540 y un error absoluto medio de 0,406072. Estos valores son más de cinco veces superiores a los obtenidos con el formato $U_{5,10}$.

Por ello, la selección de bits fraccionales no debe basarse únicamente en la cobertura del rango representable, sino también en métricas de error que permitan evaluar la fidelidad numérica del resultado.

Caso 3: Conversión con formatos y reglas aritméticas explícitas

En el tercer caso se mantienen los formatos $U_{0,11}$ para las entradas y $U_{5,10}$ para la salida, pero se especifican explícitamente los modos de cuantización y desbordamiento mediante `fimath`, utilizando redondeo al valor más cercano y saturación frente a desbordamiento.

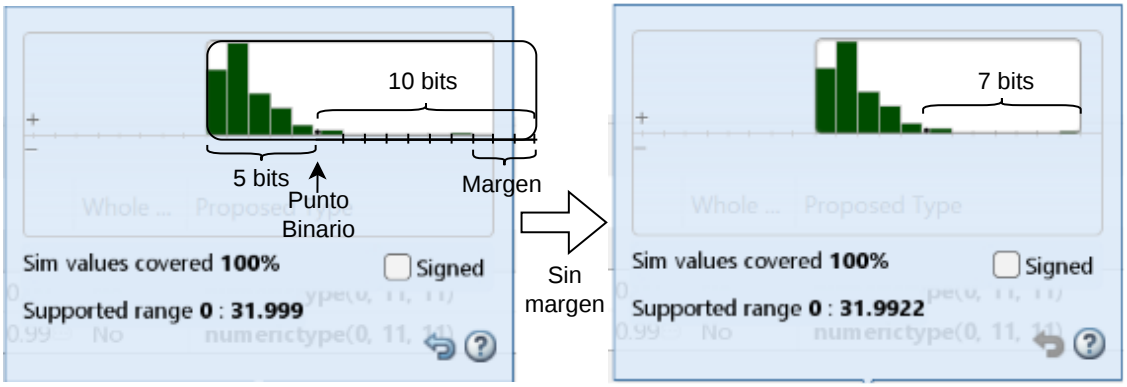


Figura 4.6: Histograma de salida para el formato $U_{5,7}$, obtenido al reducir los bits fraccionales respecto del formato $U_{5,10}$.

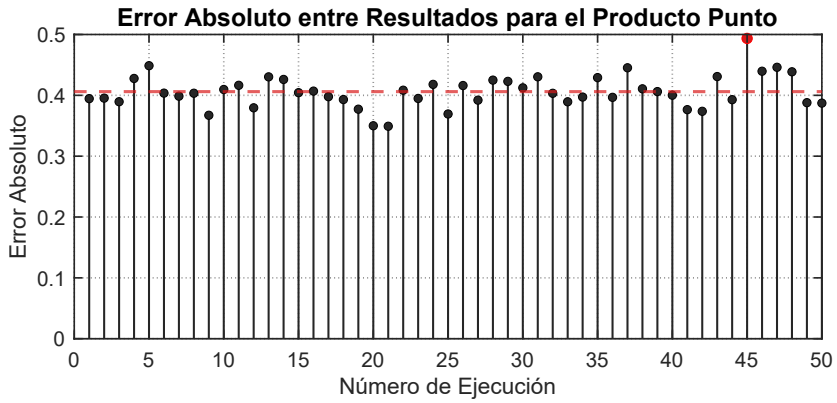


Figura 4.7: Error absoluto entre el producto punto en punto flotante y la versión en punto fijo con formatos ajustados a partir del análisis de rangos, con menos bits para acumulador.

La Figura 4.8 muestra el error absoluto entre la versión en punto flotante y la versión en punto fijo con formatos y configuraciones explícitas. Para este caso se obtiene un error absoluto máximo de 0,033977 y un error absoluto medio de 0,024592, valores menores que los obtenidos en el caso 2 sin aumentar la longitud de palabra. Estos resultados muestran que, aun manteniendo los mismos formatos, la configuración de las operaciones puede modificar de forma apreciable el error numérico de la implementación. Por ello, su definición debe considerarse explícitamente durante la exploración y el posterior traslado de la representación numérica entre herramientas.

Resumen de los casos evaluados en MATLAB

La Tabla 4.6 resume los casos evaluados en MATLAB y los errores obtenidos para cada uno de ellos. Esta síntesis permite comparar tres decisiones de diseño: utilizar la configuración automática propuesta por la herramienta, ajustar manualmente los formatos a partir del análisis de rangos y especificar de forma explícita las reglas aritméticas mediante `fimath`.

Los resultados muestran que el caso 1 presenta el menor error absoluto máximo, aunque utiliza una representación de mayor longitud para las entradas. En el caso 2, la cantidad de bits se reduce mediante formatos ajustados a partir del análisis de rangos, pero esta reducción aumenta tanto el error absoluto máximo como el error absoluto medio. Esta comparación evidencia el compromiso entre disminuir la cantidad de bits utilizados y conservar la fidelidad numérica del resultado.

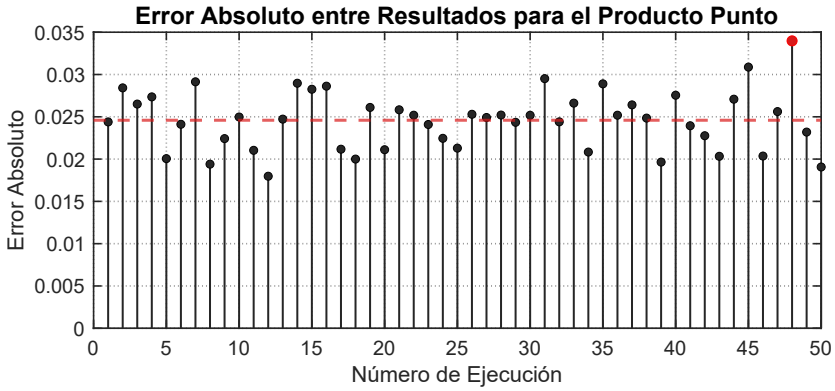


Figura 4.8: Error absoluto entre el producto punto en punto flotante y la versión en punto fijo con formatos, métodos de cuantización y desbordamiento explícitos.

Tabla 4.6: Configuraciones consideradas para la conversión del producto punto en MATLAB.

Parámetro	**Caso 1: por defecto**	**Caso 2: bits ajustados**	**Caso 3: bits y `fimath` ajustados**
Formato de entrada	$U_{0,16}$	$U_{0,11}$	$U_{0,11}$
Formato de salida	$U_{5,11}$	$U_{5,10}$	$U_{5,10}$
Método de cuantización	`Floor`	`Floor`	`Nearest`
Acción desbordamiento	`Wrap`	`Wrap`	`Saturate`
Error absoluto máximo	0,027720	0,081095	0,033977
Error absoluto medio	0,025408	0,073044	0,024592

El caso 3 mantiene los formatos reducidos del caso 2, pero modifica las configuraciones de cuantización y desbordamiento. Al reemplazar el truncamiento por redondeo al valor más cercano y definir saturación frente a desbordamiento, el error disminuye respecto del caso 2 sin aumentar la longitud de palabra, de hecho, el error absoluto medio obtenido resulta menor al del caso 1, pero con formatos más compactos.

A partir de esta exploración, se selecciona el caso 3 como modelo de punto fijo de referencia para el traslado a Vitis HLS. El conjunto de entradas utilizado en su evaluación y las salidas obtenidas en MATLAB con esta representación conforman la *golden reference* empleada en las comparaciones posteriores entre entornos.

4.4.3. Traslado de la representación numérica a Vitis HLS

Validación de equivalencia numérica entre entornos

Con el caso 3 definido como representación numérica de referencia, el siguiente paso consiste en implementar el producto punto en C/C++ para Vitis HLS. En esta etapa se utilizan tipos `ap_fixed` con parámetros equivalentes a los definidos en MATLAB, trasladando la longitud total de palabra, la cantidad de bits enteros, el modo de cuantización y el modo de desbordamiento seleccionados para dicho caso.

El Código 4.7 muestra la implementación del producto punto en Vitis HLS utilizando tipos de punto fijo sin signo. Dado que los formatos seleccionados en MATLAB no requieren bit de signo, se emplea la plantilla `ap_ufixed`. Para las entradas y para el vector `res`, que almacena los productos $x_i y_i$ antes de su acumulación, se define el tipo `data_tin`, equivalente al formato $U_{0,11}$ seleccionado en MATLAB. Por lo tanto, se declara con una longitud total de palabra de 11 bits y sin bits enteros:

$$\texttt{ap_ufixed}\langle 11,\ 0,\ \texttt{AP_RND},\ \texttt{AP_SAT}\rangle.$$

Para almacenar el resultado acumulado se define el tipo `data_tout`, equivalente al formato $U_{5,10}$. En este caso, la longitud total de palabra es de 15 bits, con 5 bits destinados a la parte entera:

$$\texttt{ap_ufixed}\langle 15,\ 5,\ \texttt{AP_RND},\ \texttt{AP_SAT}\rangle.$$

En ambos tipos se especifican explícitamente los modos de cuantización y desbordamiento correspondientes a los utilizados en MATLAB, de acuerdo con las equivalencias de la Tabla 4.5. Con ello se busca reproducir en Vitis HLS la representación numérica seleccionada previamente.

```
#include "ap_fixed.h"
#include "hls_config.h"
#include <stdio.h>

typedef ap_ufixed<11,0,AP_RND,AP_SAT> data_tin;
typedef ap_ufixed<15,5,AP_RND,AP_SAT> data_tout;
void hls_pp(data_tout *pp, data_tin x[N], data_tin y[N]);

void hls_pp(data_tout *pp, data_tin x[N], data_tin y[N])

{
    data_tin res[N];
    data_tout r=0;
    MainLoop: for (int i = 0; i < N; ++i)
    {
        res[i] = x[i]*y[i];
    }

    SecondLoop: for (int i = 0; i < N; ++i)
        {
            r+= res[i];
        }

    *pp = r;
}
```

Código 4.7: Producto punto en Vitis HLS usando formatos y modos equivalentes a MATLAB.

La comparación entre entornos se realiza utilizando el conjunto de entradas que contiene la *golden reference* generada a partir del caso 3. Para cada prueba, el error relativo entre un resultado evaluado z_{eval} y un valor de referencia z_{ref} se calcula como:

$$e_{\text{rel}} = \frac{|z_{\text{eval}} - z_{\text{ref}}|}{|z_{\text{ref}}|} \times 100\,\%$$

A partir de esta expresión se realizan tres comparaciones con el fin de identificar posibles diferencias introducidas durante el traslado entre herramientas:

- resultado en punto fijo de MATLAB respecto de la referencia en punto flotante.
- resultado en punto fijo de Vitis HLS respecto de la referencia en punto flotante.
- resultado en punto fijo de Vitis HLS respecto del resultado en punto fijo de MATLAB.

La Figura 4.9 presenta la distribución de estos errores para los 50 productos punto evaluados.

Los gráficos de caja en la Figura 4.9 ilustran que las representaciones en punto fijo de MATLAB y Vitis HLS presentan un error relativo inferior al 0,15 % con respecto a la referencia en punto flotante. Más relevante que la magnitud del error, es importante notar que ambos errores tienen exactamente la misma distribución. Esto último se comprueba al calcular el error relativo entre la representación en punto fijo de MATLAB y Vitis HLS, el cual es nulo para el conjunto de pruebas considerado en el testbench. Esto indica que ambas representaciones son exactamente equivalentes. Esta coincidencia es el resultado esperado del traslado controlado de la configuración, ya que se mantuvieron los formatos de representación, los modos aritméticos y la estructura de cálculo utilizada para el producto punto.

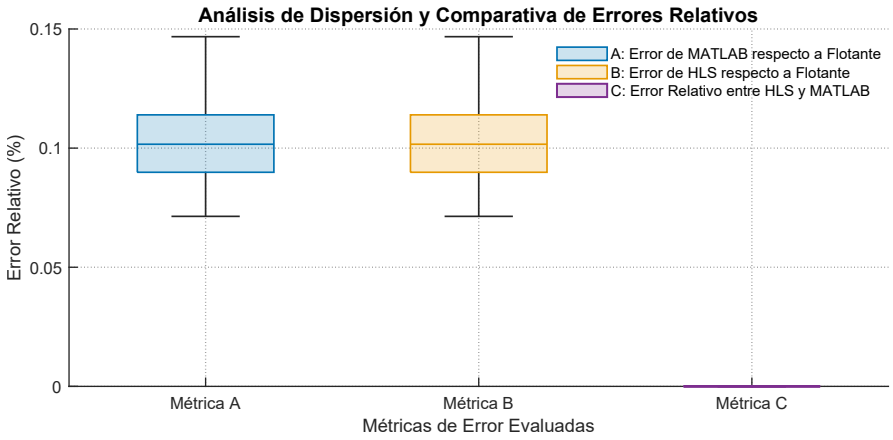


Figura 4.9: Comparación de errores entre la referencia en punto flotante, la implementación en punto fijo de MATLAB y la implementación en punto fijo de Vitis HLS.

Tabla 4.7: Uso de recursos para los tres casos evaluados en la implementación del producto punto.

Recurso	**Caso 1** $U_{0,16} \rightarrow U_{5,11}$ AP_TRN, AP_WRAP	**Caso 2** $U_{0,11} \rightarrow U_{5,10}$ AP_TRN, AP_WRAP	**Caso 3** $U_{0,11} \rightarrow U_{5,10}$ AP_RND, AP_SAT
LUTs	50	49	70
FFs	66	65	67
BRAMs	0,5	0,5	0,5
DSPs	1	1	1

Evaluación del costo de implementación de distintas representaciones en punto fijo

Luego de verificar la consistencia numérica al trasladar el algoritmo desde MATLAB hacia Vitis HLS, se consideran nuevamente los tres casos definidos en la Sección 4.4.2 para el cálculo del producto punto en punto fijo. En esta etapa, los diseños correspondientes a los tres casos se sintetizan e implementan en FPGA con el objetivo de analizar el costo asociado a cada representación. Las pruebas se realizan sin incorporar directivas HLS adicionales, de modo que las diferencias observadas se asocien principalmente a los formatos de punto fijo y a los modos de cuantización y desbordamiento, y no a optimizaciones de arquitectura como pipeline o desenrollado de ciclos. Para la evaluación, se considera la implementación sobre la FPGA Artix-7, con una frecuencia objetivo de 100 MHz.

La Tabla 4.7 muestra que la reducción de los formatos entre los casos 1 y 2 produce cambios pequeños en el uso de recursos para la implementación evaluada. El número de LUTs disminuye de 50 a 49 y el de FFs de 66 a 65, mientras que el uso de BRAMs y DSPs permanece sin cambios. Por lo tanto, en este ejemplo, la reducción de la longitud de palabra no se traduce en una disminución apreciable de los recursos utilizados.

La diferencia más visible aparece en el caso 3, que mantiene los mismos formatos del caso 2 pero utiliza `AP_RND` para cuantización y `AP_SAT` para desbordamiento. En este caso, el número de LUTs aumenta de 49 a 70, mientras que las variaciones en FFs son menores y el uso de BRAMs y DSPs permanece sin cambios. Este aumento no puede atribuirse únicamente a la saturación, ya que también se modifica el modo de cuantización. Sin embargo, el resultado es coherente con lo indicado en la documentación de Vitis HLS, donde se señala que los modos de saturación `AP_SAT*` pueden requerir lógica adicional para detectar valores fuera de rango y ajustar la salida al límite representable, incrementando con ello el uso de recursos lógicos [19]. A cambio de este mayor uso de LUTs, el caso 3 presenta una mejora en el comportamiento numérico respecto del caso 2.

En conjunto, los resultados muestran que el costo de implementación no depende únicamente de los formatos seleccionados. Para este ejemplo, la reducción de la longitud de palabra tiene un

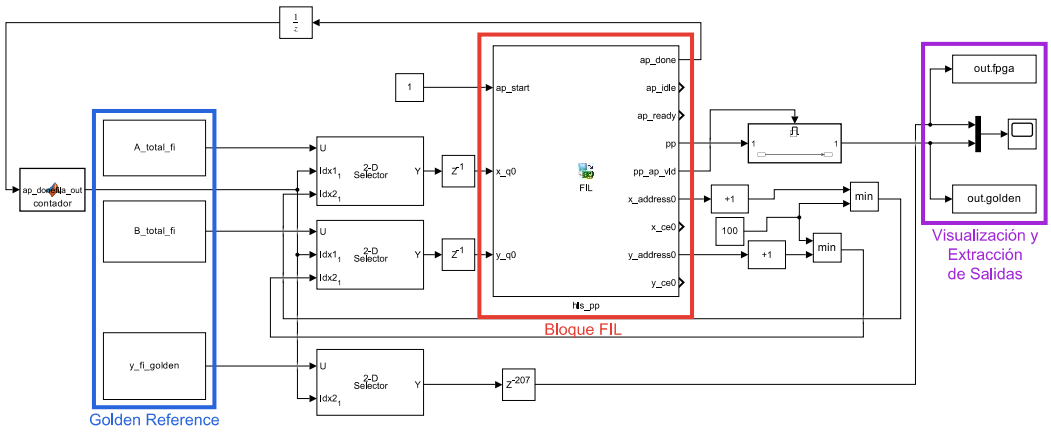


Figura 4.10: Esquema de validación mediante FPGA-in-the-Loop para el producto punto.

efecto limitado sobre los recursos utilizados, mientras que el cambio en las configuraciones sobre la cuantización y desbordamiento produce una variación más evidente en el número de LUTs y, al mismo tiempo, modifica el error numérico obtenido. Esto refuerza la necesidad de considerar de manera conjunta el formato, la configuración y el costo de implementación al seleccionar una representación numérica.

4.4.4. Validación en hardware mediante FPGA-in-the-Loop

La etapa final del flujo de diseño propuesto corresponde a validación en hardware mediante FIL. En esta etapa, el diseño RTL generado se implementa y ejecuta en la FPGA de la placa de desarrollo y se comunica con un modelo en MATLAB/Simulink que actúa como entorno de estímulo, comunicación, adquisición y comparación de resultados.

La incorporación de FIL extiende la validación realizada previamente en MATLAB, C/C++ y mediante co-simulación RTL hasta la ejecución física del diseño en la FPGA. De esta forma, es posible comprobar que la implementación ya sintetizada, implementada y programada en hardware mantiene el comportamiento numérico esperado para el conjunto de pruebas considerado. Esta capacidad resulta especialmente útil en los casos de estudio del capítulo siguiente, donde el bloque implementado en la FPGA se integra con el modelo de la planta en Simulink para evaluar el sistema en lazo cerrado.

La Figura 4.10 muestra el esquema implementado en Simulink para la validación mediante FIL. El bloque FIL, destacado en rojo, actúa como interfaz de comunicación con el diseño cargado en la FPGA. La sección en azul corresponde a la *golden reference*, que reúne las entradas de prueba y las salidas esperadas obtenidas a partir del modelo de referencia. Estas entradas se utilizan como estímulo para el bloque FIL, mientras que la respuesta producida por el hardware se alinea temporalmente con la referencia correspondiente. Finalmente, la sección en lila agrupa los bloques utilizados para visualizar y extraer ambas señales, permitiendo su comparación.

Para que la comparación sea consistente con las etapas anteriores, las señales intercambiadas mediante FIL deben utilizar la misma representación numérica definida en MATLAB y Vitis HLS. Además, la salida de la FPGA debe alinearse temporalmente con la *golden reference*, considerando la latencia del diseño y la secuencia de comunicación con el bloque RTL, de modo que cada resultado se compare con la muestra correspondiente.

Los detalles de configuración de la interfaz, sincronización de señales, carga de los vectores y alineación temporal utilizados en esta validación se encuentran disponibles en el repositorio asociado a esta tesis.

La Figura 4.11 presenta la comparación entre la *golden reference* y la salida obtenida mediante FIL. Para todos los datos evaluados, la respuesta del hardware coincide exactamente con la referencia en punto fijo generada en MATLAB. El error nulo indica que el paso desde el modelo en punto fijo hacia

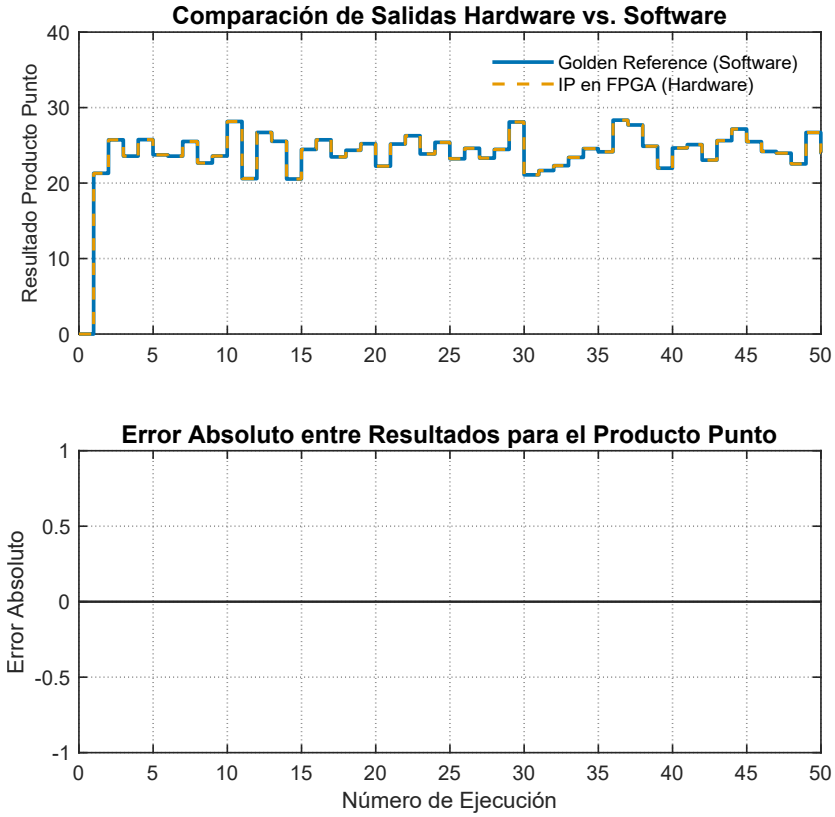


Figura 4.11: Comparación entre la *golden reference* y la salida obtenida mediante FPGA-in-the-Loop.

la implementación final en FPGA no introduce discrepancias numéricas adicionales para el conjunto de pruebas considerado, siempre que la representación de datos, la interfaz de comunicación y la alineación temporal se configuren correctamente.

Con esta validación se completa el traslado del producto punto desde el modelo de referencia en punto fijo hasta su ejecución en hardware. Para el conjunto de pruebas considerado, los resultados muestran una coincidencia exacta entre las salidas obtenidas en MATLAB y las producidas por la implementación en FPGA mediante FIL. Este resultado no implica que dicha equivalencia se obtenga automáticamente en otros diseños, pero confirma que, en este caso, el flujo propuesto permitió mantener el comportamiento numérico de referencia hasta la ejecución en hardware, al reproducir la representación numérica y configurar adecuadamente el entorno de validación entre etapas del flujo.

4.4.5. Restricciones prácticas para equivalencia entre entornos

El análisis anterior muestra que, para las condiciones evaluadas, es posible obtener resultados equivalentes entre las distintas etapas del flujo cuando se reproduce la representación numérica y se mantiene una estructura de cálculo compatible. En el producto punto considerado, la simplicidad de la operación permite identificar explícitamente los formatos de las variables, la configuración de las operaciones, los puntos de almacenamiento intermedio y la secuencia de acumulación. Esto permite reproducir estas decisiones de forma explícita al trasladar el algoritmo desde MATLAB hacia C/C++ y Vitis HLS, obteniendo resultados coincidentes entre las implementaciones evaluadas.

Sin embargo, mantener formatos y configuraciones equivalentes no garantiza por sí solo una coincidencia bit a bit entre entornos. El resultado también depende de la estructura de cálculo, incluyendo el orden de las operaciones y los momentos en que un resultado se ajusta a un formato determinado. Por esta razón, dos implementaciones que utilizan las mismas representaciones numéricas

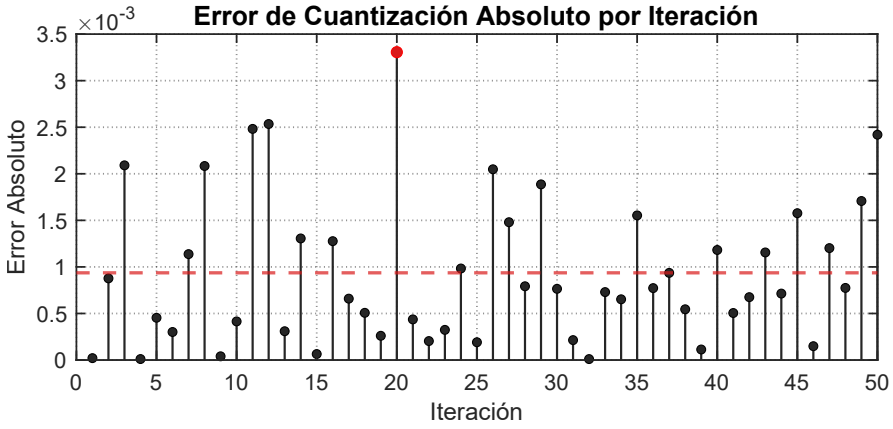


Figura 4.12: Comparación entre la referencia en punto flotante y la implementación en punto fijo obtenida a partir de la descripción alternativa del producto punto.

pueden producir resultados distintos si la secuencia de cálculo no se reproduce de la misma forma.

Esta situación resulta especialmente relevante al trasladar un algoritmo desde MATLAB hacia C/C++ para HLS. MATLAB permite expresar operaciones vectoriales y matriciales de forma compacta, mientras que en la implementación C/C++ utilizada en este trabajo estas operaciones se describen de manera más explícita mediante ciclos, acumuladores y variables intermedias. Aunque ambas descripciones representen la misma operación matemática, la secuencia efectiva de cálculo puede diferir y, con ello, también los resultados intermedios generados antes de obtener la salida final.

Para ilustrar este efecto, se considera una variante del producto punto en la que se mantienen los formatos y la configuración de las operaciones utilizados en el caso 3, pero se modifica la estructura de cálculo en MATLAB. En lugar de expresar explícitamente la multiplicación elemento a elemento y la acumulación mediante ciclos, el producto punto se describe mediante una operación vectorial, como se muestra en el Código 4.8.

```
1 function [result] = producto_punto(x,y)
2 result = x*y';
3 end
```

Código 4.8: Función alternativa de MATLAB en punto flotante para el producto punto.

La Figura 4.12 muestra el error absoluto obtenido al comparar la versión alternativa en punto fijo de MATLAB con la referencia en punto flotante. Para esta descripción se obtiene un error absoluto máximo de 0,003306 y un error absoluto medio de 0,000937, valores menores que los reportados en los casos 1, 2 y 3. Esta reducción se relaciona con la estructura utilizada para realizar el producto punto. Bajo la configuración de precisión completa, MATLAB utiliza formatos intermedios con la precisión necesaria para representar los resultados de las multiplicaciones y acumulaciones sin introducir una reducción adicional de precisión antes de ajustar la salida a su formato de destino. De esta forma, la descripción vectorial introduce menos pérdidas de precisión que la implementación que almacena explícitamente los productos parciales antes de acumularlos.

La implementación en C/C++ utilizada en Vitis HLS mantiene, en cambio, la estructura explícita de productos parciales y acumulación empleada en el caso 3. Por ello, aunque ambas implementaciones corresponden al mismo producto punto y utilizan los mismos formatos y modos de cuantización y desbordamiento, los resultados intermedios no se generan ni se almacenan de la misma forma.

La Figura 4.13 presenta una comparación análoga a la realizada anteriormente para el caso 3. Se muestran los errores de la implementación alternativa en punto fijo de MATLAB respecto de la referencia en punto flotante, los errores de la implementación en punto fijo de Vitis HLS respecto de

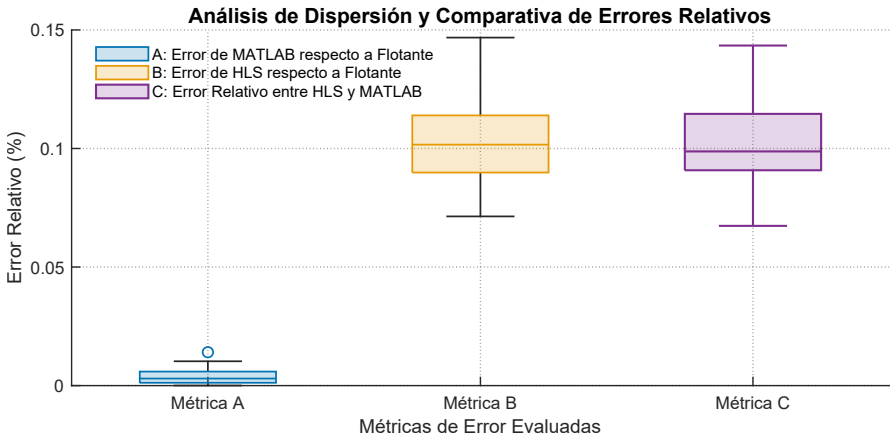


Figura 4.13: Errores relativos obtenidos al comparar la descripción alternativa del producto punto en MATLAB y Vitis HLS respecto de la referencia en flotante.

esa misma referencia y el error directo entre ambas implementaciones en punto fijo. A diferencia del caso 3, el error entre MATLAB y Vitis HLS deja de ser nulo, aun cuando se mantienen los mismos formatos y modos de cuantización y desbordamiento.

Este resultado reafirma que la coincidencia exacta entre entornos depende también de la estructura de cálculo. En ejemplos simples, esta estructura puede reproducirse de forma explícita entre herramientas y permitir una coincidencia bit a bit. En algoritmos más complejos, reproducir exactamente todos los resultados intermedios puede resultar poco práctico o innecesario. En estos casos, la validación debe centrarse en identificar y cuantificar las diferencias numéricas introducidas durante el traslado y verificar que se mantengan dentro de los criterios definidos para la aplicación.

4.5. Observaciones y discusión del capítulo

El ejemplo del producto punto permitió aplicar y validar el flujo propuesto sobre una operación simple que combina multiplicaciones y una acumulación. A partir de este caso fue posible analizar cómo el formato, la configuración de cuantización y desbordamiento y el tratamiento de los resultados intermedios afectan tanto el error numérico como el costo de implementación.

La exploración en MATLAB mostró que la selección de una representación numérica no puede basarse únicamente en cubrir los rangos observados durante la simulación. Esta condición permite evitar desbordamientos para los datos evaluados, pero no garantiza una resolución suficiente en la salida. Por ello, la selección de los bits enteros y fraccionales debe complementarse con métricas de error y criterios definidos para la aplicación. Asimismo, los resultados mostraron que la configuración de cuantización y desbordamiento puede modificar el error obtenido aun cuando se mantienen los mismos formatos.

El traslado hacia Vitis HLS mostró que es posible mantener la correspondencia numérica con MATLAB cuando se reproducen la representación numérica y la estructura de cálculo. Posteriormente, la validación mediante FIL extendió esta verificación hasta la ejecución del diseño en la FPGA y mostró que, para el conjunto de pruebas considerado, la implementación reprodujo exactamente la *golden reference* generada en MATLAB.

La evaluación de recursos mostró además que el costo de implementación depende tanto de los formatos como de la configuración de cuantización y desbordamiento. La reducción de la longitud de palabra puede disminuir los recursos requeridos, mientras que configuraciones como el redondeo y la saturación pueden requerir lógica adicional. En consecuencia, la selección de una representación numérica requiere considerar conjuntamente el rango, la resolución, el error aceptable y el costo de la arquitectura resultante.

Finalmente, el análisis de una descripción alternativa del producto punto mostró que utilizar los

mismos formatos y configuraciones no garantiza por sí solo una coincidencia exacta entre entornos. El orden de las operaciones y los puntos donde los resultados intermedios se almacenan y ajustan a un formato determinado también pueden modificar la salida. Por esta razón, el flujo propuesto busca identificar, explicar y cuantificar las diferencias introducidas entre etapas, en lugar de asumir que siempre debe alcanzarse una equivalencia bit a bit. Esta consideración será especialmente relevante en el capítulo siguiente, donde el flujo se aplica a controladores de mayor complejidad y la evaluación debe considerar tanto las diferencias numéricas como su efecto sobre el comportamiento del sistema en lazo cerrado y el costo de implementación.

Capítulo 5

IMPLEMENTACIÓN Y VERIFICACIÓN DE LAZOS MPC

Este capítulo aplica el flujo de diseño propuesto a dos controladores MPC con formulación implícita previamente desarrollados y validados en punto flotante. A diferencia del ejemplo del capítulo anterior, donde el flujo se verificó sobre una operación aritmética aislada, en este capítulo se estudia el efecto de la representación en punto fijo sobre controladores evaluados en lazo cerrado. Para cada caso se exploran distintos formatos de punto fijo en MATLAB, se traslada a Vitis HLS la representación seleccionada y se evalúan la respuesta del sistema, el uso de recursos y la latencia de la implementación.

Los casos considerados corresponden a un motor de corriente continua (DC) y a un sistema de recursos energéticos distribuidos (DER). El motor DC ha sido utilizado previamente en trabajos del grupo de investigación y permite aplicar inicialmente el procedimiento de selección de la representación numérica y analizar su efecto sobre la implementación. El sistema DER presenta una exigencia temporal mayor, con un período de muestreo de 200 μs, e incorpora más estados y actuaciones, además de perturbaciones medibles y una mayor cantidad de restricciones. Estas características permiten aplicar el flujo en un caso con una mayor cantidad de señales y una restricción temporal más exigente.

5.1. Metodología de evaluación

Los dos casos de estudio se evalúan a partir de los modelos y scripts de simulación en punto flotante desarrollados en [25]. Estos modelos, previamente validados, se utilizan como referencia funcional para estudiar la implementación de los controladores con una representación en punto fijo. Se mantienen sin cambios la formulación de cada controlador y los parámetros que definen el problema MPC, como el horizonte de predicción, las ponderaciones de la función de costo y las restricciones, junto con los parámetros del *solver* ADMM definidos en el trabajo de referencia. De esta manera, el análisis se centra en los efectos asociados al cambio de representación numérica.

5.1.1. Modelo y formulación común de los lazos de control

Los dos casos de estudio parten del modelo lineal discreto presentado en la ecuación (2.1) para formular el controlador. En el sistema DER, este modelo se extiende para incorporar un vector de perturbaciones medibles $\mathbf{d}_k$ mediante el término $\mathbf{B}_p\mathbf{d}_k$. Las dimensiones de las variables y los valores de las matrices dependen del sistema considerado, por lo que se especifican posteriormente en las secciones dedicadas al motor DC y al sistema DER.

Los controladores utilizan una formulación implícita de MPC, cuyo problema QP se resuelve

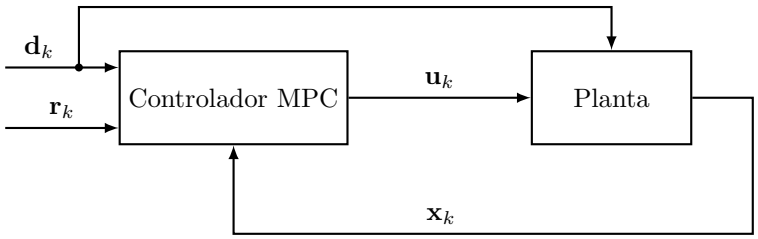


Figura 5.1: Esquema general de simulación en lazo cerrado utilizado en los casos de estudio. El vector de perturbaciones medibles $\mathbf{d}_k$ se utiliza únicamente en el sistema DER.

mediante ADMM. En cada instante de muestreo, el controlador utiliza el estado de la planta y la referencia para formular el problema de optimización. A partir de su solución se calcula una secuencia de actuaciones sobre un horizonte de predicción N, de la cual se aplica únicamente la primera actuación, de acuerdo con la estrategia de horizonte deslizante. Ambos casos conservan las 10 iteraciones de ADMM utilizadas en las implementaciones de referencia, mientras que el horizonte de predicción, las dimensiones del problema QP y las restricciones se definen según cada sistema.

Para la formulación densa utilizada en este trabajo, con M_{SYS} actuaciones y un horizonte de predicción N, la cantidad de variables de decisión queda dada por:

$$N_{\mathrm{QP}} = N\, M_{\mathrm{SYS}}. \tag{5.1}$$

Las simulaciones consideran disponibilidad completa de los estados de la planta, por lo que no se incorporan observadores para su estimación. Tampoco se incluyen ruido de medición, cuantización asociada a los conversores ni retardos externos al controlador. Estas condiciones permiten concentrar la evaluación en el comportamiento del controlador con las representaciones numéricas consideradas.

La Figura 5.1 resume el esquema utilizado para evaluar ambos controladores en lazo cerrado.

5.1.2. Preparación del entorno y etapas de evaluación

Durante la exploración en punto fijo en MATLAB, el entorno de evaluación se adapta para concentrar el análisis en el controlador y aproximar su estructura a la implementación posterior en C/C++. Los principales cambios realizados corresponden a:

- La separación entre el modelo de la planta y la función que implementa el controlador MPC, de modo que la conversión a punto fijo se aplique únicamente al controlador.
- La descomposición de expresiones que agrupan varias operaciones aritméticas en operaciones individuales, de modo que el valor obtenido después de cada operación se asigne a una variable. Esta reestructuración aproxima en MATLAB la secuencia de asignaciones utilizada en la implementación C/C++ y permite considerar la reducción de precisión asociada a cada una de ellas, con el fin de obtener resultados numéricos similares entre ambos entornos, de acuerdo con lo discutido en el Capítulo 4.
- El cálculo previo de las matrices invariantes y su incorporación como constantes, siguiendo la estructura utilizada posteriormente en el código destinado a síntesis.

Una vez adaptado el entorno de exploración en MATLAB, los dos casos de estudio siguen las etapas del flujo presentado en el Capítulo 4:

1. Obtener una referencia funcional mediante simulación en lazo cerrado con aritmética de punto flotante.

2. Explorar distintos formatos de punto fijo en MATLAB y seleccionar una representación numérica a partir de las métricas de error y de la respuesta del sistema.
3. Trasladar la representación seleccionada a C/C++, reproduciendo el formato y la configuración numérica definidos en MATLAB, y comprobar su correspondencia mediante simulación en C.
4. Verificar mediante cosimulación RTL que la descripción generada por Vitis HLS reproduzca el comportamiento de la implementación C/C++.
5. Sintetizar e implementar el diseño para caracterizar su latencia y el uso de recursos.

En el caso del motor DC, la evaluación se extiende además a la ejecución del RTL sobre la FPGA mediante FPGA-in-the-Loop.

Un aspecto central de esta metodología es que las distintas versiones del controlador se evalúan en lazo cerrado. El *testbench* utilizado originalmente para verificar la implementación C/C++ evaluaba el controlador de manera aislada, mediante una secuencia predefinida de estados y la comparación de las actuaciones obtenidas con una *golden reference*. Este procedimiento permite comprobar el comportamiento funcional del controlador para entradas conocidas, pero no permite observar cómo las diferencias numéricas en la actuación afectan posteriormente la evolución de la planta.

Por esta razón, el *testbench* utilizado en Vitis HLS se adapta para incorporar el modelo de la planta y cerrar el lazo de control. En cada instante de muestreo, la actuación calculada por el controlador se utiliza para actualizar el estado de la planta antes de ejecutar la siguiente llamada. De esta manera, la simulación de referencia en punto flotante, la implementación en punto fijo de MATLAB y las versiones evaluadas en Vitis HLS generan trayectorias propias, coherentes con las actuaciones calculadas en cada caso. Así, es posible observar cómo una diferencia numérica en la actuación modifica los estados posteriores y cómo estos cambios afectan las entradas recibidas por el controlador en los siguientes instantes de muestreo. Este efecto no se observa cuando todas las versiones reciben una misma secuencia predefinida de estados.

5.1.3. Representación numérica común

En ambos casos de estudio se adopta una representación homogénea, en la que todas las variables internas del controlador utilizan un mismo formato $Q_{m,n}$. La exploración se centra así en evaluar cómo la distribución de bits entre las partes entera y fraccional afecta el comportamiento numérico y la implementación del controlador. La asignación independiente de formatos según el rango y la resolución requeridos por cada variable queda fuera del alcance de este trabajo.

Junto con el uso de un formato homogéneo, se define una configuración numérica común para ambos casos de estudio:

- **Modo de cuantización:** Se utiliza el redondeo al valor representable más cercano (*Nearest*), evitando el sesgo asociado a una cuantización sistemática por truncamiento.
- **Modo de desbordamiento:** Se utiliza saturación. Cuando un resultado excede el rango representable, su valor se limita al extremo correspondiente, evitando los cambios de magnitud o signo que puede producir el *wrap-around*.
- **Precisión de los resultados:** En MATLAB se utiliza precisión completa (*FullPrecision*) para sumas y productos, con el fin de aproximar el comportamiento de `ap_fixed`, donde la precisión de las operaciones no se configura mediante un modo independiente. Al asignar un resultado a una variable, este se ajusta al formato definido para ella.

5.1.4. Métricas y criterios de comparación

La comparación cuantitativa se realiza sobre las señales de actuación obtenidas con las distintas versiones evaluadas del controlador. Para cada instante de muestreo k, el error absoluto de una señal de actuación se define como

$$e_u(k) = |u_{\text{ref}}(k) - u_{\text{eval}}(k)|\,, \tag{5.2}$$

donde $u_{\text{ref}}(k)$ corresponde a la señal utilizada como referencia y $u_{\text{eval}}(k)$ a la señal que se compara con ella. Durante la selección de formatos, la referencia corresponde al modelo en punto flotante y la señal evaluada a la versión en punto fijo. La misma definición se utiliza para comparar posteriormente los resultados obtenidos en Vitis HLS con la implementación en punto fijo de MATLAB.

A partir de esta diferencia se calculan el error absoluto medio y el error absoluto máximo:

$$\overline{e}_u = \frac{1}{N_s}\sum_{k=1}^{N_s} e_u(k), \tag{5.3}$$

$$e_{u,\text{máx}} = \max_{1\le k\le N_s} e_u(k), \tag{5.4}$$

donde N_s es la cantidad total de instantes de muestreo considerados. En el motor DC se evalúa una única actuación, mientras que en el sistema DER las métricas se calculan por separado para u_d y u_q.

Estas métricas se complementan con el análisis de la respuesta en lazo cerrado. Para cada caso se revisan el seguimiento de las referencias, la evolución de los estados y el cumplimiento de las restricciones. Como cada versión del controlador genera su propia trayectoria, una vez que los estados comienzan a diferir, el error de actuación incorpora tanto las diferencias numéricas introducidas en el controlador como el efecto de recibir estados distintos en los instantes posteriores. Por esta razón, los errores medio y máximo se interpretan junto con la evolución de la respuesta del sistema y con el instante en que se producen las mayores diferencias.

La caracterización temporal utiliza el número de ciclos requeridos para completar una llamada al controlador y el período de reloj considerado. El tiempo de cálculo se obtiene como

$$t_{\text{c}} = N_{\text{ciclos}} T_{\text{clk}}, \tag{5.5}$$

y la frecuencia de cálculo correspondiente queda dada por

$$f_{\text{c}} = \frac{1}{N_{\text{ciclos}} T_{\text{clk}}}. \tag{5.6}$$

La frecuencia f_{c} representa la cantidad de actuaciones que pueden calcularse por segundo bajo el período de reloj considerado y no debe confundirse con la frecuencia de reloj de la FPGA. La evaluación de hardware se completa con el número de LUTs, FFs, DSPs y BRAMs reportado para cada implementación.

5.2. Motor de corriente continua

El motor DC se utiliza como primer caso de estudio debido a que su modelado y control están ampliamente documentados, lo que permite trabajar con una formulación previamente establecida y concentrar la evaluación en los aspectos de implementación. En [14] se emplea este sistema para evaluar una metodología de implementación basada en HLS, mientras que en [25] se utiliza como caso de referencia para analizar, en punto flotante, el efecto de distintas directivas de paralelización sobre la latencia y el uso de recursos.

Este caso permite evaluar si una representación en punto fijo puede seleccionarse durante una etapa temprana del diseño y mantenerse a lo largo del flujo de implementación, conservando el comportamiento del controlador dentro de los criterios establecidos.

5.2.1. Modelo, controlador y criterios de evaluación

El modelo del motor y la formulación del controlador se toman de [25]. A continuación se resumen únicamente los elementos necesarios para describir el caso de estudio y mantener el desarrollo autocontenido.

Modelo de la planta

Para el motor DC, el modelo presentado en la ecuación (2.1) utiliza como estados la velocidad angular ω_k y la posición angular θ_k, por lo que:

$$\mathbf{x}_k = \begin{bmatrix} \omega_k \\ \theta_k \end{bmatrix}.$$

Dado que el controlador solo utiliza la posición angular como variable de salida (es decir, $y_k = \theta_k$), la matriz de salida $\mathbf{C}$ se define para extraer únicamente este estado:

$$\mathbf{C} = \begin{bmatrix} 0 & 1 \end{bmatrix}.$$

Las matrices que describen la evolución discreta de los estados son:

$$\mathbf{A} = \begin{bmatrix} e^{-T_s/\tau} & 0 \\ \tau\left(1 - e^{-T_s/\tau}\right) & 1 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} \kappa\left(1 - e^{-T_s/\tau}\right) \\ T_s\kappa + \tau\kappa\left(e^{-T_s/\tau} - 1\right) \end{bmatrix}.$$

Esta formulación corresponde a la discretización exacta de un modelo continuo de primer orden para la velocidad, mientras que la posición se obtiene mediante su integración. Se asume que la entrada u_k permanece constante durante cada intervalo de muestreo. Los parámetros utilizados son un período de muestreo $T_s = 1$ ms, una constante de tiempo $\tau = 1/27{,}92$ s y una ganancia $\kappa = 39{,}08/27{,}92$.

Características del controlador predictivo

Para el motor DC se utiliza un horizonte de predicción $N = 4$, manteniendo sin cambios los parámetros del controlador definidos en [25]. El sistema cuenta con dos estados ($N_{\mathrm{SYS}} = 2$) y una actuación ($M_{\mathrm{SYS}} = 1$). Por lo tanto, para la formulación densa considerada, la cantidad de variables de decisión es:

$$N_{\mathrm{QP}} = N \cdot M_{\mathrm{SYS}} = 4,$$

mientras que el número de restricciones de desigualdad queda dado por:

$$M_{\mathrm{QP}} = 2 \cdot N\left(N_{\mathrm{SYS}} + M_{\mathrm{SYS}}\right) = 24.$$

Las restricciones consideradas establecen límites sobre la actuación y los estados a lo largo del horizonte de predicción. El voltaje aplicado al motor se restringe a $u_k \in [-3, 3]$ V, mientras que la velocidad y la posición angular se restringen a $\omega_k \in [-5, 5]$ rad/s y $\theta_k \in [-2, 2]$ rad, respectivamente.

Criterios de comparación y selección

De acuerdo con las métricas definidas en la Sección 5.1.4, la comparación cuantitativa se realiza sobre la actuación del controlador, correspondiente al voltaje aplicado al motor u_k.

Para la exploración de formatos se establece como criterio principal una tolerancia equivalente al 5 % del rango total de la actuación. Dado que el voltaje se encuentra restringido al intervalo $[-3, 3]$ V, este porcentaje corresponde a $0,3$ V. El criterio principal se define entonces como:

$$\overline{e}_u < 0{,}3 \text{ V}.$$

Este umbral se utiliza como referencia para comparar los formatos evaluados y no corresponde a un límite normativo ni a una condición derivada del modelo del motor. Para el error absoluto máximo, el mismo valor se utiliza como referencia secundaria, pero no como un criterio de descarte estricto, ya que una diferencia puntual se interpreta junto con el instante en que ocurre y con su efecto sobre la respuesta del sistema.

La comparación se complementa con el análisis del seguimiento de posición y del cumplimiento de las restricciones de posición, velocidad y actuación.

5.2.2. Exploración de formatos de punto fijo en MATLAB

La exploración de formatos se realiza mediante las simulaciones en lazo cerrado descritas en la Sección 5.1. Para este caso, el *testbench* del trabajo de referencia se extiende en duración y se incorporan varios cambios en la referencia de posición, con el propósito de evaluar el comportamiento del controlador durante distintos transitorios.

Escenarios de simulación

Se consideran dos escenarios que difieren en la condición inicial y en la referencia aplicada durante el primer tramo de la simulación. En el primero, la posición inicial se fija en -1 rad y la referencia en 1 rad, lo que corresponde a un error inicial de seguimiento de 2 rad. En el segundo, estos valores se establecen en -2 rad y 2 rad, respectivamente, aumentando el error inicial a 4 rad. Una vez finalizado este primer tramo, ambos escenarios utilizan la misma secuencia de referencias.

El primer escenario se utiliza para cuantificar las diferencias respecto de la referencia en punto flotante bajo un transitorio de menor amplitud, mientras que el segundo permite evaluar el comportamiento del formato ante una variación inicial más amplia y revisar el cumplimiento de las restricciones.

La Figura 5.2 presenta la respuesta en punto flotante para el primer escenario. Se muestran la posición angular, la velocidad angular y la actuación, junto con sus referencias y restricciones correspondientes. Esta respuesta se utiliza como referencia para calcular las métricas de error durante la exploración.

Evaluación del formato $Q_{4,7}$

Primer escenario. Manteniendo la configuración numérica definida en la Sección 5.1.3, se exploran distintas longitudes de palabra y distribuciones entre bits enteros y fraccionales. En el primer escenario, el formato $Q_{4,7}$, con una longitud total de 11 bits, satisface el criterio principal definido para el error absoluto medio de la actuación. Como se observa en la Figura 5.3, las respuestas en punto fijo y punto flotante mantienen una evolución próxima durante la mayor parte de la simulación.

Para el formato $Q_{4,7}$ se obtiene un error absoluto medio de $0,0081$ V, inferior al umbral de $0,3$ V. El error absoluto máximo alcanza $0,5233$ V y se concentra durante el transitorio inicial, intervalo en el que también se observa una superación de la restricción de actuación en la respuesta de referencia en punto flotante.

Dado que en este escenario aparecen superaciones puntuales de las restricciones, se amplía la evaluación para comprobar el comportamiento del formato frente a un transitorio de mayor amplitud.

Segundo escenario. La Figura 5.4 presenta la respuesta obtenida con el formato $Q_{4,7}$ bajo las condiciones del segundo escenario. Durante el transitorio inicial, la actuación calculada en punto fijo se separa de la referencia en punto flotante y presenta una superación más amplia de sus límites. Esta diferencia se propaga hacia la velocidad angular, cuya respuesta cuantizada también excede la restricción definida, mientras que la implementación en punto flotante se mantiene dentro de ella. Como resultado, la trayectoria de la posición en punto fijo alcanza la referencia a diferencia de la respuesta flotante, pero lo hace a partir de una evolución de la actuación y de la velocidad distinta de la utilizada como referencia.

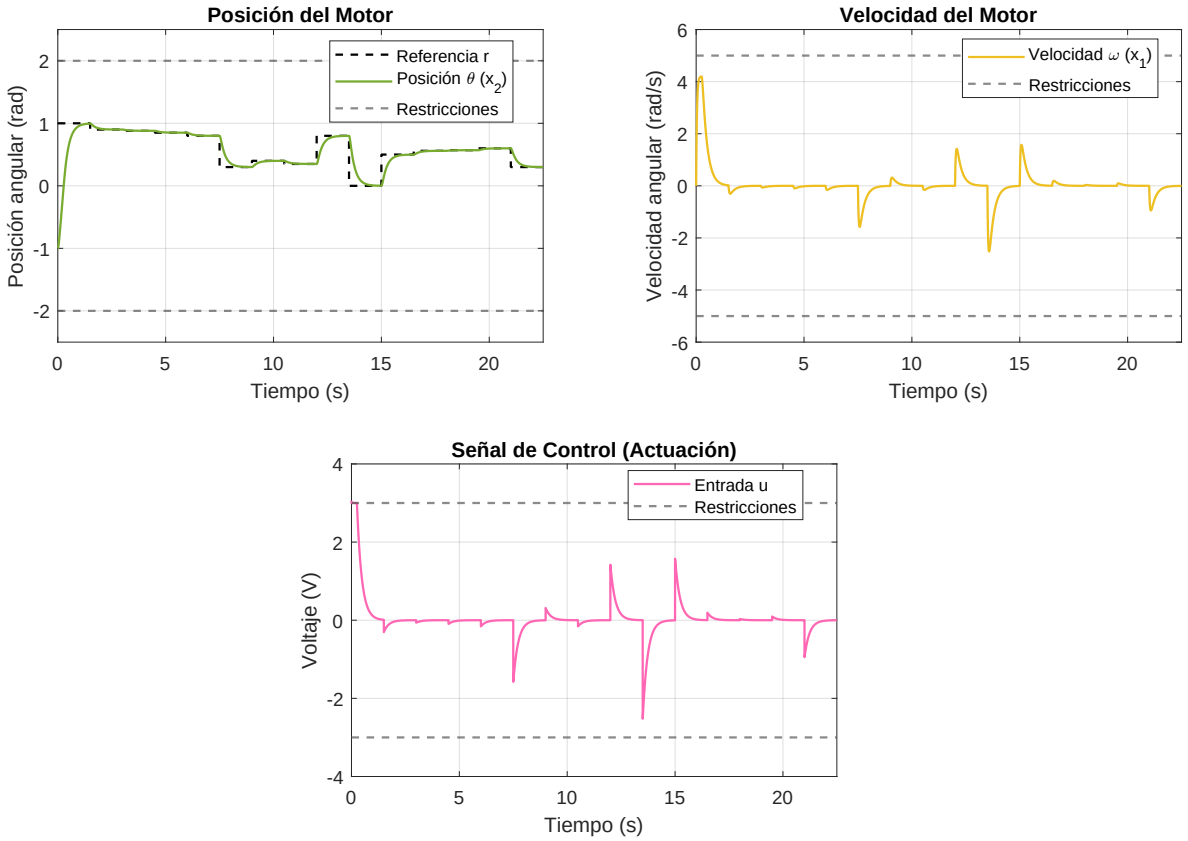


Figura 5.2: Simulación del controlador MPC en lazo cerrado utilizando representación en punto flotante.

Aunque $Q_{4,7}$ satisface el criterio asociado al error absoluto medio en el primer escenario, frente al desplazamiento inicial más amplio aumenta la separación respecto de la referencia y se acentúan las superaciones de las restricciones de actuación y velocidad. Por esta razón, este formato no se selecciona para continuar con la implementación.

Selección del formato $Q_{6,8}$

Ajuste del rango representable. Las pruebas realizadas durante la exploración mostraron que aumentar únicamente la cantidad de bits fraccionales no mejoraba suficientemente la respuesta en el segundo escenario. Por este motivo, se amplió el rango representable aumentando la cantidad de bits enteros hasta $m = 6$ y, a partir de este valor, se ajustó la resolución fraccional. Este proceso condujo al formato $Q_{6,8}$, con una longitud total de palabra de $W = 14$ bits.

Evaluación en el segundo escenario. La Figura 5.5 muestra la respuesta de este formato para el segundo escenario. En comparación con $Q_{4,7}$, el aumento del rango representable reduce la separación respecto de la implementación en punto flotante durante el transitorio inicial. La actuación y la velocidad mantienen una evolución más cercana a la referencia, y las superaciones observadas se reducen respecto del formato $Q_{4,7}$.

Aun con esta mejora, la trayectoria cuantizada no coincide con la obtenida en punto flotante. En particular, durante el primer transitorio la implementación en punto fijo alcanza la referencia, mientras que la respuesta en punto flotante permanece por debajo de ella durante el resto del intervalo evaluado. Esto hace que el controlador cuantizado presente, para este caso, un seguimiento aparentemente más rápido. Sin embargo, este comportamiento no debe interpretarse como una ventaja general de la cuantización, ya que se relaciona con variaciones de mayor magnitud en la actuación, las cuales modifican con mayor rapidez la evolución de los estados.

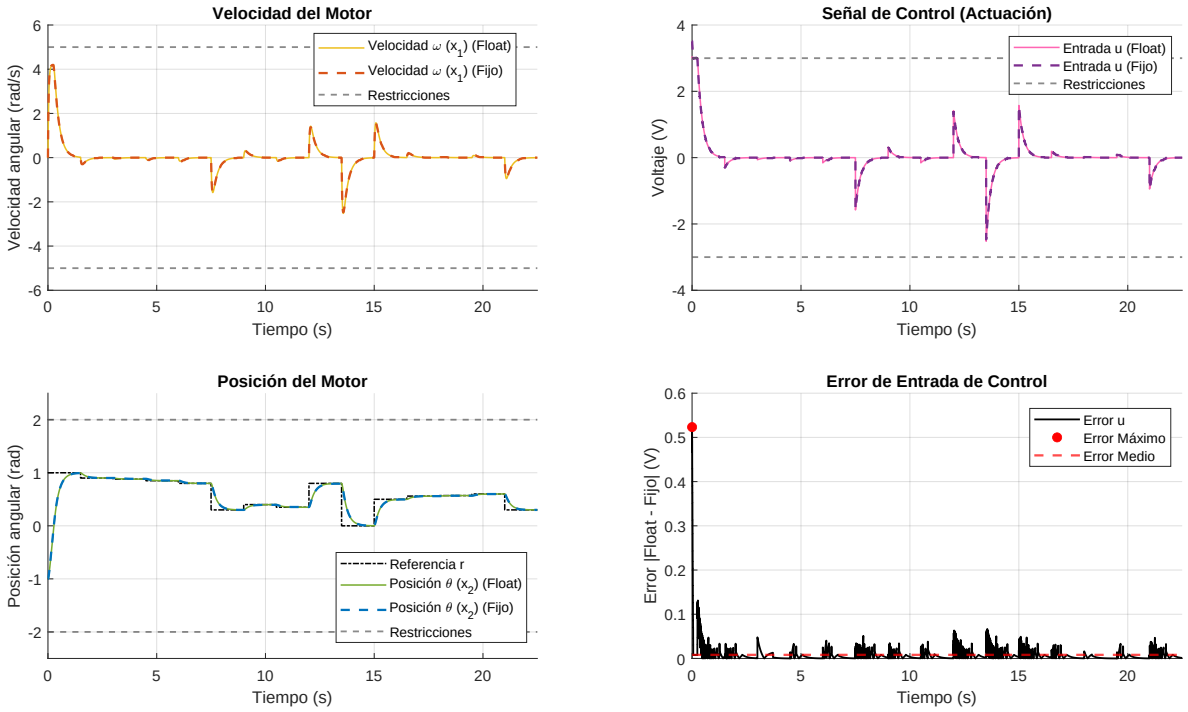


Figura 5.3: Comparación de la respuesta del motor en punto flotante y punto fijo con formato $Q_{4,7}$ para el primer escenario de simulación.

La diferencia entre ambas trayectorias confirma que el error instantáneo de la actuación debe interpretarse junto con la respuesta del sistema, de acuerdo con el criterio general establecido en la Sección 5.1.4. En este caso, la simulación en lazo cerrado permite revisar esta relación antes del traslado a hardware.

Criterio de selección. Los dos escenarios cumplen funciones complementarias dentro de la exploración. El primero permite cuantificar la cercanía entre las implementaciones bajo una respuesta en la que sus trayectorias permanecen próximas, mientras que el segundo permite comprobar el comportamiento del formato frente a una variación inicial más amplia y revisar su efecto sobre las restricciones. Por lo tanto, la selección no se basa exclusivamente en una de estas pruebas, sino en la combinación de las métricas obtenidas en el primer escenario y de la respuesta a nivel de sistema observada en el segundo.

Al volver al primer escenario, el formato $Q_{6,8}$ presenta un error absoluto medio de la actuación de $0,0052$ V y un error absoluto máximo de $0,073$ V respecto del controlador en punto flotante. Ambas métricas se mantienen por debajo del umbral de $0,3$ V definido para la comparación. Además, en el segundo escenario este formato reduce las diferencias observadas con $Q_{4,7}$ y mantiene una evolución más cercana a la referencia en punto flotante.

A partir de estos resultados se selecciona el formato $Q_{6,8}$ para continuar con la implementación en Vitis HLS. En la etapa siguiente, esta representación se traslada a C/C++ con el propósito de obtener una respuesta cercana a la observada en MATLAB, tanto en las métricas de error como en el comportamiento del sistema en lazo cerrado.

5.2.3. Implementación y validación en Vitis HLS

La representación $Q_{6,8}$ seleccionada en MATLAB se traslada a C/C++ mediante tipos `ap_fixed` con la misma longitud de palabra, posición del punto binario y reglas de cuantización y desbordamiento. Como se discutió en el Capítulo 4, el orden de evaluación y el almacenamiento de resultados intermedios

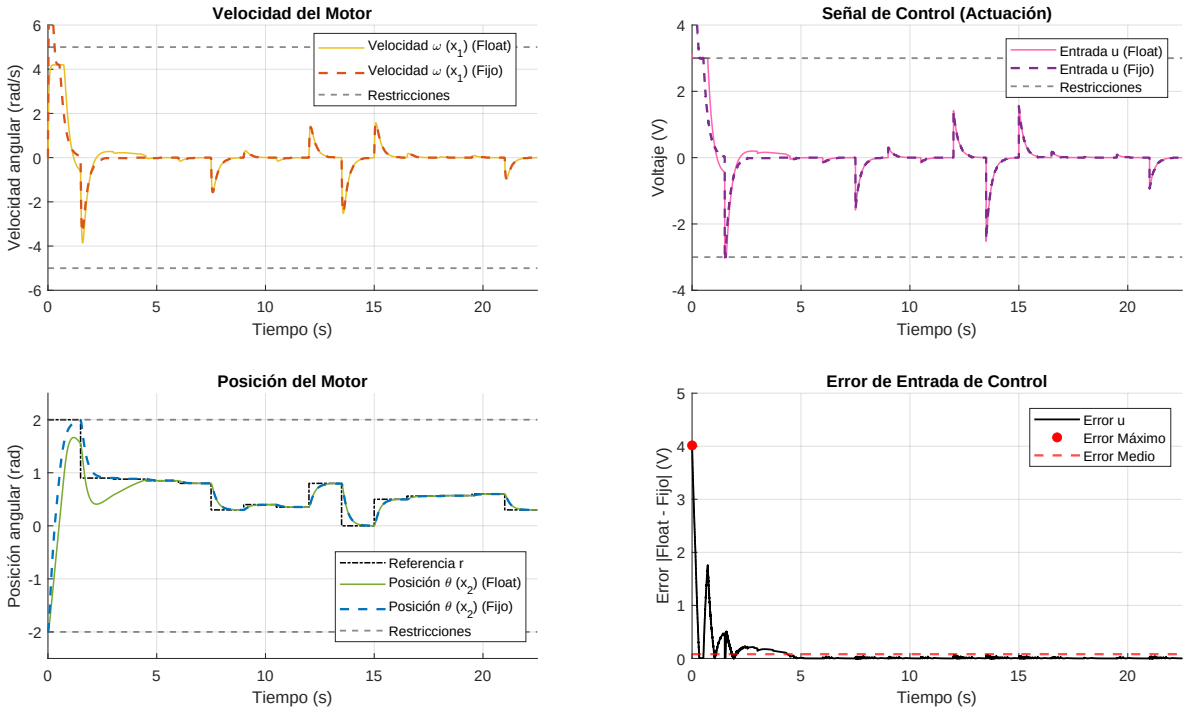


Figura 5.4: Comparación de la respuesta del motor con formato $Q_{4,7}$ para el segundo escenario de simulación.

pueden producir diferencias entre herramientas, por lo que la comprobación considera tanto la correspondencia con el modelo fijo de MATLAB como el comportamiento del lazo respecto de la referencia en punto flotante.

La implementación se orienta a la plataforma Nexys A7, que incorpora una FPGA Xilinx Artix-7 100T. Se establece un período de reloj objetivo de 10 ns, equivalente a 100 MHz. La primera síntesis utiliza las opciones por defecto de Vitis HLS y no incorpora directivas manuales de optimización, con el fin de caracterizar inicialmente el efecto de la representación numérica.

Comprobación numérica y verificación RTL

El formato $Q_{6,8}$ se evalúa primero mediante simulación en C y posteriormente mediante cosimulación RTL. Para el escenario evaluado, ambas entregan los mismos resultados. Por esta razón, la Figura 5.6 presenta únicamente la simulación en C, comparada con las versiones en punto flotante y punto fijo de MATLAB, mientras que la cosimulación RTL se utiliza para verificar la correspondencia entre la implementación C/C++ y el RTL generado.

Durante gran parte de la simulación, el error entre la implementación en C/C++ de Vitis HLS y el modelo en punto fijo de MATLAB es nulo o de baja magnitud. Las diferencias principales se concentran en los transitorios, como permite apreciar el zoom de la actuación y de su error. En estos intervalos, los cambios de la señal pueden producirse en muestras distintas o adoptar temporalmente valores diferentes en cada implementación, lo que genera intervalos de error comparables a los obtenidos respecto de la referencia en punto flotante. Sin embargo, una vez finalizado el transitorio, las actuaciones vuelven a coincidir o a mantenerse muy próximas. En conjunto, el sistema conserva el seguimiento de la referencia y el error respecto de la implementación en punto flotante no supera el umbral definido. Estos resultados muestran que el comportamiento en lazo cerrado observado durante la exploración en MATLAB se conserva después del traslado a C/C++.

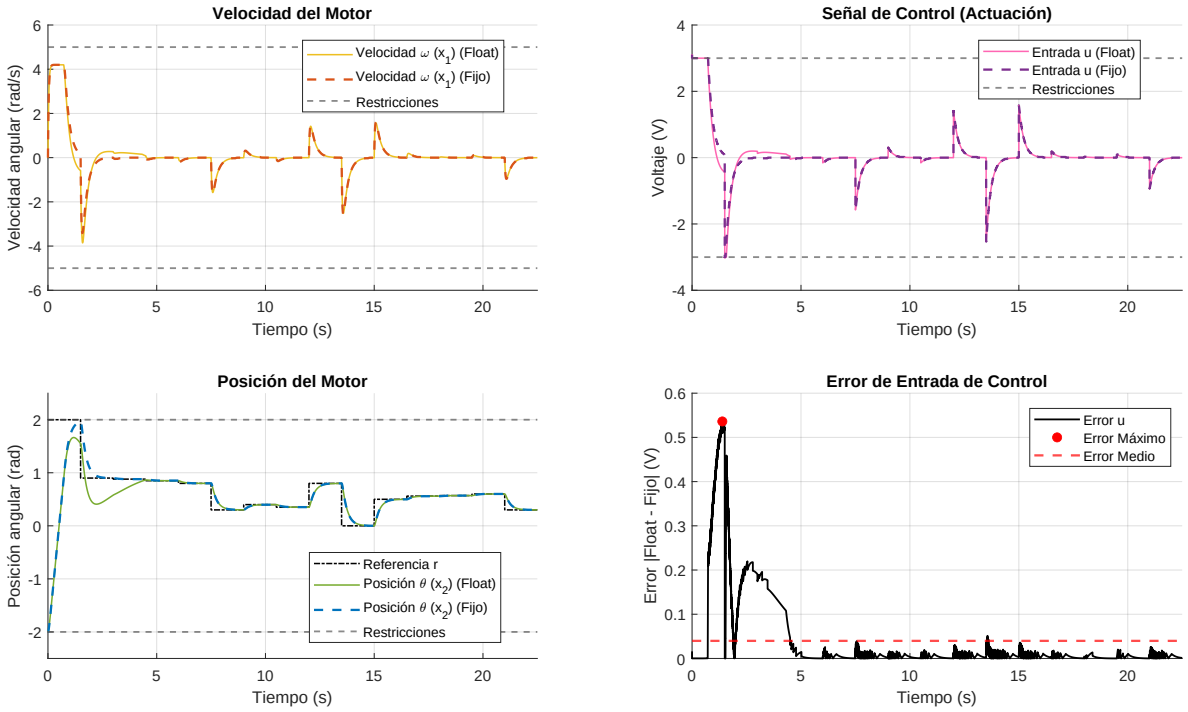


Figura 5.5: Comparación de la respuesta del motor con formato $Q_{6,8}$ para el segundo escenario de simulación.

Tabla 5.1: Resultados posteriores a la implementación en Vitis HLS para distintas representaciones numéricas del controlador MPC, sin directivas manuales de optimización.

Formato	**LUTs**	**FFs**	**DSPs**	**BRAMs**	**Ciclos**	f_c **(kHz)**	$\overline{e}_u$ **(V)**	$e_{u_{\text{máx}}}$ **(V)**
Flotante	7775	13820	120	9	3621	27,617	0,00000153	0,00001216
$Q_{6,26}$ (32 bits)	3715	5799	94	9	2823	35,423	0,00000153	0,00001168
$Q_{6,8}$ (14 bits)	1847	1851	31	0	2650	37,736	0,00684163	0,0500777

Síntesis base sin directivas manuales de optimización

Una vez verificada la respuesta numérica, se realiza una síntesis base para analizar el efecto de la representación sobre el uso de recursos y el desempeño temporal. Las opciones de Vitis HLS se mantienen por defecto y todavía no se incorporan directivas manuales de optimización.

La comparación considera tres versiones del controlador, punto flotante de precisión simple, $Q_{6,26}$ y $Q_{6,8}$. El formato $Q_{6,26}$ conserva los seis bits enteros del formato seleccionado y completa una longitud total de 32 bits con 26 bits fraccionales. De esta forma actúa como una referencia de punto fijo de alta resolución, útil para distinguir el efecto de cambiar la representación numérica desde punto flotante del efecto adicional producido al reducir la longitud de palabra hasta 14 bits. Las tres versiones se sintetizan e implementan con el mismo dispositivo y período de reloj objetivos. La Tabla 5.1 resume el uso de recursos, la cantidad de ciclos necesaria para calcular una actuación y las métricas de error respecto de la referencia en punto flotante.

La cantidad de ciclos corresponde a un cálculo completo de la función del controlador, que incluye las 10 iteraciones de ADMM y entrega una actuación para un instante de muestreo del lazo cerrado. La frecuencia de cálculo se obtiene a partir de este valor y del período de reloj mediante la expresión definida en la ecuación (5.6).

Los resultados de la Tabla 5.1 muestran una reducción progresiva de los recursos a medida que disminuye la longitud de palabra. El cambio desde punto flotante hacia $Q_{6,26}$ reduce de manera

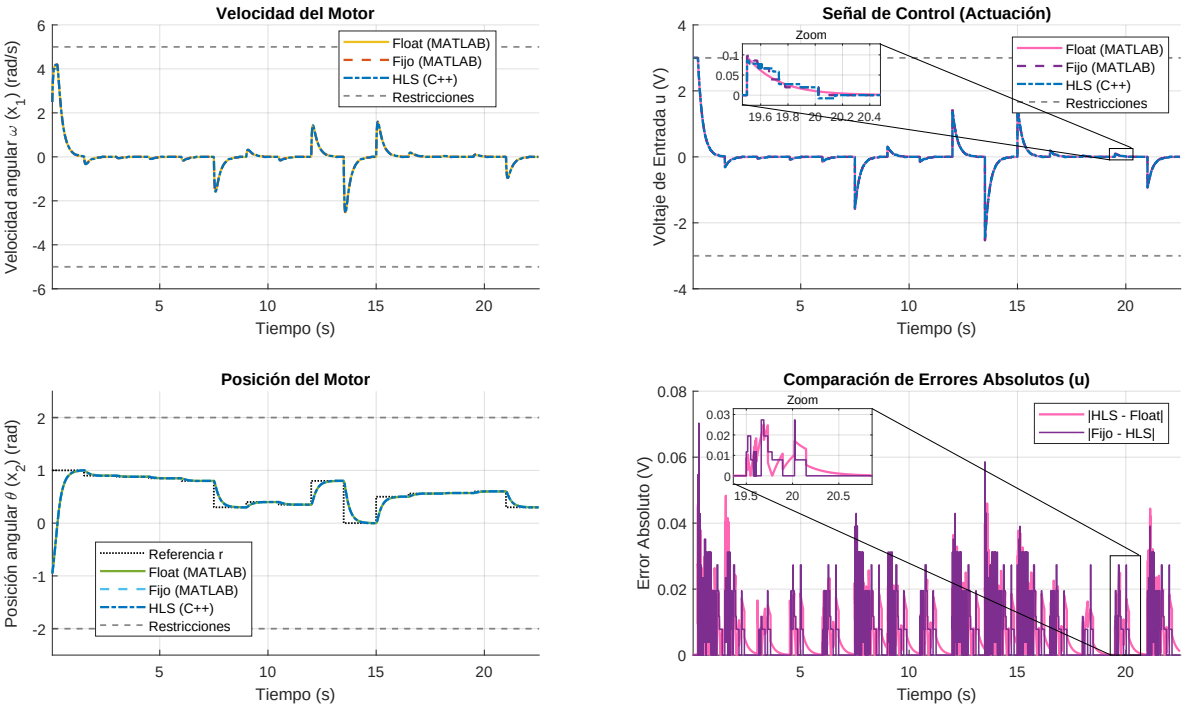


Figura 5.6: Comparación en lazo cerrado entre las implementaciones en punto flotante y punto fijo de MATLAB y la simulación en C de Vitis HLS para el formato $Q_{6,8}$.

importante el uso de LUTs y FFs, además de disminuir la cantidad de bloques DSP. Esta tendencia se acentúa con $Q_{6,8}$, que presenta la menor ocupación entre las representaciones evaluadas.

También se observa un cambio en la implementación de las estructuras de almacenamiento. Con las opciones por defecto utilizadas en esta síntesis, las versiones en punto flotante y $Q_{6,26}$ emplean bloques BRAM, mientras que $Q_{6,8}$ se mapea sin este recurso. Este resultado describe la decisión automática tomada por Vitis HLS para cada ancho de palabra y no una propiedad de los formatos. El diseñador puede modificar el mapeo mediante directivas de almacenamiento, por lo que la asignación final también depende de la profundidad de los arreglos, sus patrones de acceso y las restricciones impuestas a la herramienta.

En cuanto a la precisión, $Q_{6,26}$ mantiene una respuesta prácticamente equivalente a la referencia en punto flotante, mientras que $Q_{6,8}$ introduce un error mayor como consecuencia de la reducción de bits fraccionales. Aun así, las dos métricas consideradas permanecen dentro del umbral definido para la actuación y son cercanas a las obtenidas durante la exploración en MATLAB, lo que respalda la selección del formato de 14 bits.

La reducción de la longitud de palabra también disminuye la cantidad de ciclos requeridos para calcular una actuación y, en consecuencia, aumenta la tasa máxima de cálculo. En las tres implementaciones, la frecuencia de cálculo supera con holgura la frecuencia de muestreo de 1 kHz del motor, incluso sin incorporar directivas manuales de optimización.

Exploración con directivas de optimización

Aunque la síntesis base ya satisface el período de muestreo del motor, se evalúa el rendimiento alcanzable al aumentar el paralelismo y el costo en recursos asociado. Para ello se adopta una estrategia de optimización mediante pragmas basada en la propuesta de [25]. En las multiplicaciones matriz-vector se aplica partición completa de los arreglos involucrados en la dimensión utilizada para el cálculo y `PIPELINE` con un intervalo de iniciación objetivo de un ciclo (`II=1`). En las operaciones elemento a elemento de suma, resta y proyección se utiliza partición cíclica de los vectores con factor 16, `UNROLL`

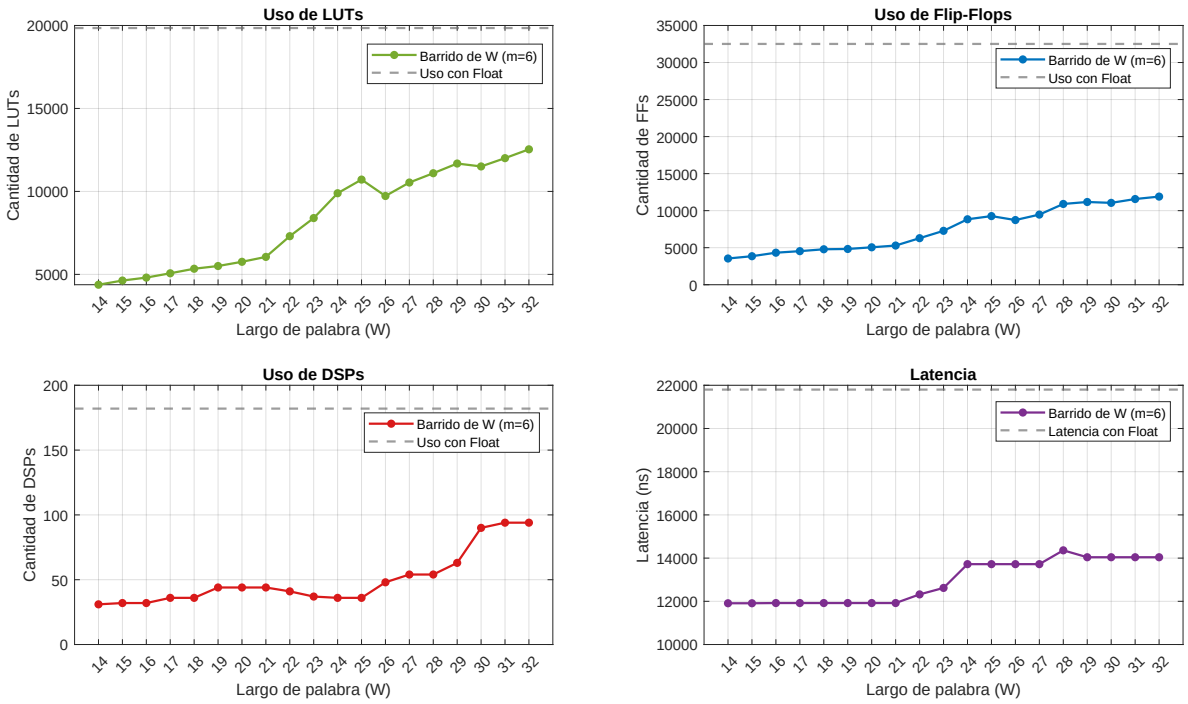


Figura 5.7: Uso de recursos y latencia del controlador con directivas de optimización para distintas longitudes de palabra, manteniendo $m = 6$.

con el mismo factor y `PIPELINE` con `II=1`.

Manteniendo estas directivas, se realiza un barrido de la longitud total de palabra entre 14 y 32 bits, con una cantidad fija de bits enteros $m = 6$. De esta forma, se evalúa el efecto de la longitud de palabra sobre la precisión numérica, el tiempo de cálculo y el uso de recursos bajo una misma estrategia de optimización.

La Figura 5.7 presenta el uso de LUTs, FFs y DSPs, junto con la latencia obtenida para cada longitud de palabra. En comparación con las implementaciones base, las directivas disminuyen la cantidad de ciclos necesaria para ejecutar el controlador, a cambio de un aumento en los recursos utilizados. Este resultado es coherente con el mayor paralelismo introducido mediante la partición de arreglos, la segmentación y el desenrollado de los bucles.

Para complementar este análisis, la Figura 5.8 muestra la evolución del error absoluto medio y máximo de la actuación respecto de la referencia en punto flotante. La incorporación de esta información permite observar conjuntamente el costo de aumentar la longitud de palabra y la mejora numérica obtenida.

Los resultados muestran una tendencia general al aumento del uso de recursos y de la latencia a medida que se incrementa la longitud total de palabra. Sin embargo, esta evolución no es estrictamente proporcional para todos los recursos, ya que Vitis HLS puede modificar la implementación de las operaciones según el ancho de los operandos y las posibilidades de mapeo disponibles en la FPGA. Por esta razón, algunos cambios en el uso de DSPs se acompañan de variaciones en LUTs y FFs, sin que pueda establecerse una relación directa entre recursos a partir de este barrido.

En paralelo, la Figura 5.8 muestra que el aumento de la cantidad de bits fraccionales reduce progresivamente el error de la actuación. La mayor mejora se concentra en las primeras extensiones de la longitud de palabra, mientras que, al continuar aumentando n, la reducción adicional del error se vuelve menos significativa. De esta manera, el barrido expone el compromiso entre precisión y costo de implementación, ya que una mayor longitud de palabra mejora la cercanía respecto de la referencia en punto flotante, pero incrementa los recursos y la latencia.

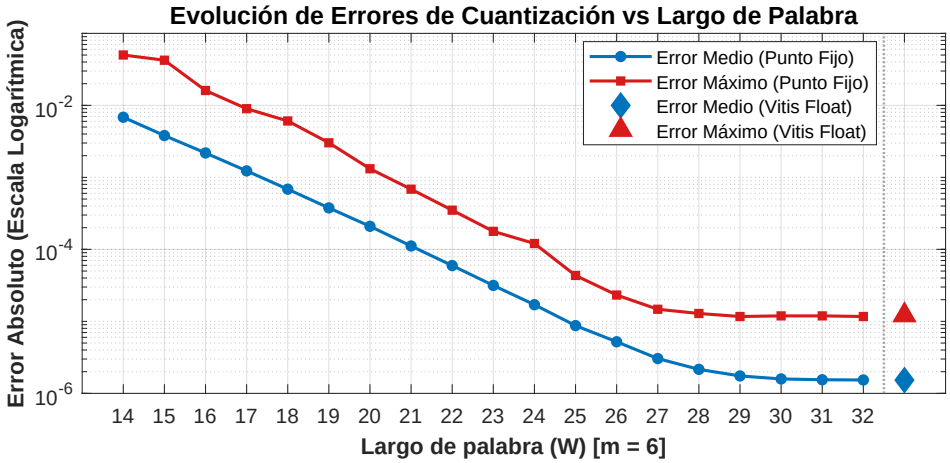


Figura 5.8: Error absoluto de la actuación respecto de la referencia en punto flotante para distintas longitudes de palabra, manteniendo $m = 6$.

Dentro de las representaciones evaluadas, el formato $Q_{6,8}$ corresponde a la alternativa de menor longitud de palabra que satisface los criterios definidos previamente. Los formatos de mayor tamaño reducen el error, pero lo hacen a costa de aumentar el uso de recursos sin que esta mejora sea necesaria para cumplir los umbrales establecidos en este caso de estudio.

El análisis anterior permite estudiar el efecto de aumentar la longitud total de palabra, pero modifica simultáneamente la cantidad de bits fraccionales. Por lo tanto, sus resultados no permiten determinar si el uso de recursos depende únicamente de W o si también varía al desplazar la posición del punto binario.

Para estudiar este aspecto se realiza un segundo barrido, ahora manteniendo fija la longitud total en $W = 32$ bits y variando la distribución entre las partes entera y fraccional. La cantidad de bits fraccionales se modifica entre $n = 8$ y $n = 26$, mientras que los bits enteros se ajustan para conservar la longitud total. La Figura 5.9 presenta los recursos y la latencia obtenidos para estos formatos.

Los resultados de la Figura 5.9 muestran que el costo de implementación no depende únicamente de la longitud total de palabra. Aunque todos los formatos utilizan $W = 32$ bits, el desplazamiento del punto binario produce variaciones tanto en los recursos como en la latencia. Estas diferencias aparecen de forma escalonada, con intervalos en los que las métricas se mantienen relativamente constantes y transiciones en las que aumentan de manera apreciable.

El efecto es especialmente visible en el uso de bloques DSP, cuya cantidad aumenta al asignar más bits a la parte fraccional. Las LUTs y los FFs presentan una evolución similar por niveles, mientras que la latencia pasa por distintos valores prácticamente constantes antes de aumentar nuevamente. Este comportamiento indica que Vitis HLS modifica la forma en que implementa y segmenta las operaciones según la distribución de bits, aun cuando el ancho total de los operandos permanezca fijo.

Por lo tanto, la posición del punto binario constituye una variable relevante durante la exploración de representaciones, ya que dos formatos con la misma longitud total pueden generar arquitecturas con distinto uso de recursos y diferente latencia. En este caso, los resultados sugieren que asignar una mayor cantidad de bits a la parte fraccional puede aumentar el costo de implementación. Sin embargo, esta tendencia depende de las operaciones del controlador y de las decisiones de mapeo realizadas por la herramienta, por lo que no debe interpretarse como una propiedad general de todos los diseños en punto fijo.

Como extensión del análisis, se comparan las implementaciones optimizadas mediante pragmas de los controladores en punto flotante de precisión simple, $Q_{6,26}$ y $Q_{6,8}$.

La comparación se realiza inicialmente en la FPGA Artix-7 integrada en la plataforma Nexys A7, utilizando un período de reloj objetivo de 10 ns. El formato $Q_{6,8}$ se implementa posteriormente en una FPGA Zynq UltraScale+ de la plataforma ZCU104. Esta implementación se evalúa primero con

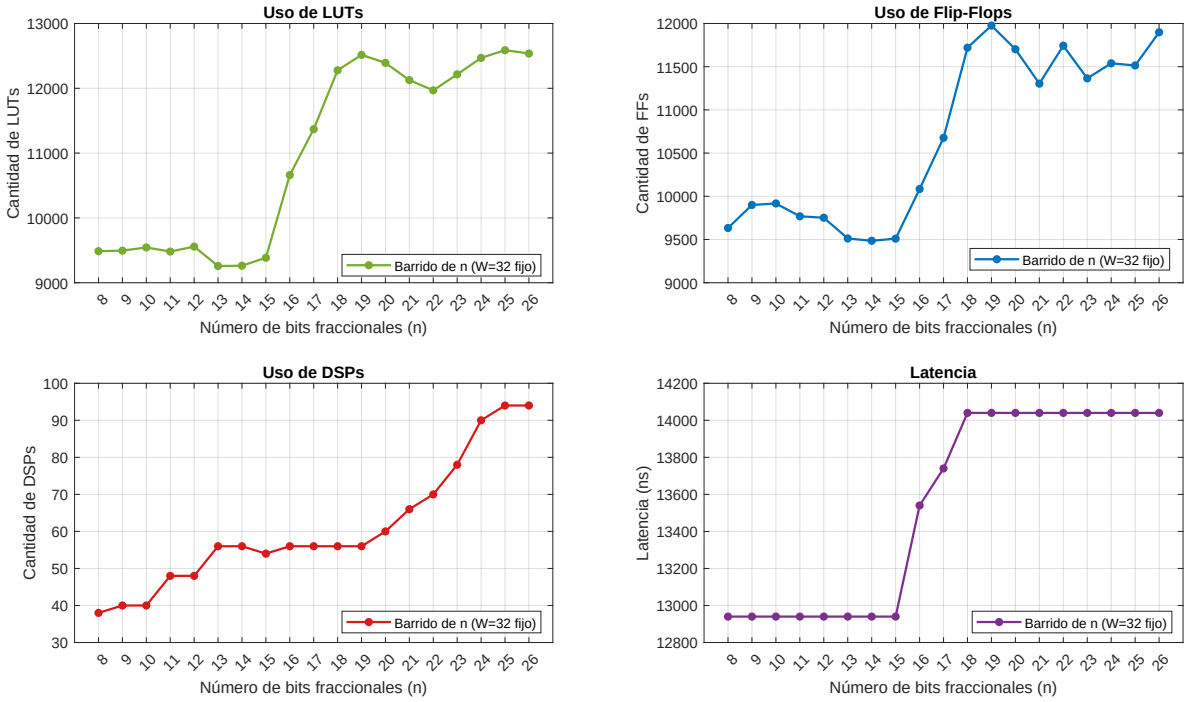


Figura 5.9: Uso de recursos y latencia para distintas distribuciones de bits enteros y fraccionales, manteniendo fija la longitud total en $W = 32$ bits.

Tabla 5.2: Resultados de las implementaciones del controlador MPC con directivas de optimización para distintas representaciones numéricas y dispositivos objetivo.

Formato	**LUTs**	**FFs**	**DSPs**	**Ciclos**	WNS (ns)	f_c **(kHz)**	**FPGA**	T_{obj} **(ns)**
Flotante	19846	32508	182	2180	0,989	45,871	Artix-7	10
$Q_{6,26}$ (32b)	12537	11898	94	1404	1,043	71,225	Artix-7	10
$Q_{6,8}$ (14b)	4378	3540	31	1191	0,672	83,96	Artix-7	10
$Q_{6,8}$ (14b)	3785	2892	31	926	3,591	107,991	UltraScale+	10
$Q_{6,8}$ (14b)	3829	3053	31	996	1,490	167,34	UltraScale+	6
$Q_{6,8}$ (14b)	3835	3012	31	1017	1,235	196,66	UltraScale+	5
$Q_{6,8}$ (14b)	3814	3073	31	1119	0,487	223,41	UltraScale+	4

el mismo período objetivo de 10 ns, para mantener condiciones comparables entre ambos dispositivos, y luego con períodos de reloj menores, con el fin de explorar el margen de frecuencia de operación alcanzable en la Zynq UltraScale+.

La comparación entre las implementaciones $Q_{6,8}$ permite observar el efecto del cambio de familia tecnológica manteniendo el formato numérico y, en la primera prueba, el mismo período objetivo. Las diferencias obtenidas en recursos y ciclos corresponden, por tanto, a la arquitectura generada para cada dispositivo y a las características de sus bloques internos, aunque los valores absolutos de LUTs, FFs y DSPs no son completamente equivalentes entre ambas familias.

Las implementaciones en la Zynq UltraScale+ con períodos objetivo de 5 ns y 4 ns aprovechan el mayor margen temporal de la plataforma. Estas pruebas permiten evaluar cuánto puede incrementarse la tasa de cálculo sin modificar el algoritmo, el formato $Q_{6,8}$ ni las directivas aplicadas, y muestran el beneficio temporal obtenido al trasladar la arquitectura a un dispositivo con mayores capacidades.

5.2.4. Validación en hardware mediante FPGA-in-the-Loop

Como comprobación final, el diseño RTL correspondiente al formato $Q_{6,8}$ se valida sobre una Nexys A7 mediante FPGA-in-the-Loop. Para esta prueba se utiliza el mismo escenario en lazo cerrado y la misma referencia en punto flotante empleada durante las etapas anteriores, lo que permite comparar la actuación calculada por la FPGA con la obtenida en la simulación en C/C++.

Durante la prueba, la respuesta obtenida mediante FIL reproduce el comportamiento observado en la simulación en C/C++, tanto en el seguimiento de la referencia como en la evolución de la actuación. No se observaron diferencias que modificaran cualitativamente la respuesta del sistema en lazo cerrado. Esta correspondencia es consistente con el uso de los tipos `ap_fixed`, cuya simulación en C permite reproducir los efectos de cuantización y desbordamiento considerados posteriormente en la descripción RTL sintetizada [19].

Dado que la respuesta observada mediante FIL no presenta diferencias apreciables respecto de las comprobaciones anteriores, no se incluye una figura adicional. Los archivos necesarios para reproducir esta validación se encuentran disponibles en el repositorio del proyecto, junto con las instrucciones de ejecución. Estos incluyen el modelo de Simulink, el *bitstream* utilizado en la Nexys A7 y los archivos RTL generados por Vitis HLS, por lo que es posible repetir la prueba sobre la misma plataforma o volver a generar el bloque FIL para otra FPGA compatible.

5.3. Sistema de recursos energéticos distribuidos

El segundo caso de estudio corresponde al control de un convertidor DC/AC conectado a un filtro LC, utilizado para la integración de recursos energéticos distribuidos. En este sistema, el controlador regula los voltajes del condensador del filtro y restringe las corrientes y actuaciones asociadas al convertidor.

El caso se basa en el controlador desarrollado en [25], donde se estudió su implementación sobre una plataforma heterogénea CPU-FPGA y el efecto de distintas directivas HLS sobre la latencia y el uso de recursos. En este trabajo se conserva el controlador definido en dicho antecedente y se aplica el flujo propuesto para seleccionar una representación en punto fijo, trasladarla a Vitis HLS y evaluar su efecto sobre la respuesta en lazo cerrado.

5.3.1. Modelo, controlador y escenario de evaluación

El modelo de la planta y los parámetros del controlador se conservan sin modificaciones respecto a la implementación de referencia.

Modelo de la planta

El sistema se representa en un marco de referencia síncrono dq mediante la transformada de Park. En este marco, las variables sinusoidales trifásicas se representan mediante componentes d y q, que permanecen constantes en régimen estacionario cuando el marco gira a la frecuencia fundamental.

El modelo utiliza como estados las corrientes del inductor y los voltajes del condensador del filtro en ambos ejes:

$$\mathbf{x}_k = \begin{bmatrix} i_{d,k} \\ i_{q,k} \\ v_{d,k} \\ v_{q,k} \end{bmatrix},$$

mientras que el vector de actuaciones contiene las dos componentes de voltaje generadas por el convertidor DC/AC que actúan como entrada al filtro:

$$\mathbf{u}_k = \begin{bmatrix} u_{d,k} \\ u_{q,k} \end{bmatrix}.$$

En el modelo utilizado, el vector de salida coincide con el vector de estados, por lo que $\mathbf{C} = \mathbf{I}_4$. Sin embargo, las referencias se definen únicamente para los voltajes del condensador. Para seleccionar estas variables se utiliza la matriz:

$$\mathbf{C}_r = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

De esta forma, el vector de variables asociadas al seguimiento de referencia queda dado por:

$$\mathbf{y}_{r,k} = \mathbf{C}_r \mathbf{x}_k = \begin{bmatrix} v_{d,k} \\ v_{q,k} \end{bmatrix}.$$

Además de las actuaciones, el modelo considera el efecto de la corriente demandada por la carga mediante un vector de perturbaciones medibles $\mathbf{d}_k$. En tiempo continuo, la dinámica del sistema se expresa como:

$$\dot{\mathbf{x}} = \mathbf{A}_c \mathbf{x} + \mathbf{B}_c \mathbf{u} + \mathbf{B}_{p,c} \mathbf{d},$$

donde

$$\mathbf{A}_c = \begin{bmatrix} -\dfrac{R_f}{L_f} & \omega_b & -\dfrac{1}{L_f} & 0 \\ -\omega_b & -\dfrac{R_f}{L_f} & 0 & -\dfrac{1}{L_f} \\ \dfrac{1}{C_f} & 0 & 0 & \omega_b \\ 0 & \dfrac{1}{C_f} & -\omega_b & 0 \end{bmatrix}, \qquad \mathbf{B}_c = \begin{bmatrix} \dfrac{1}{L_f} & 0 \\ 0 & \dfrac{1}{L_f} \\ 0 & 0 \\ 0 & 0 \end{bmatrix},$$

y la matriz que describe el efecto de las perturbaciones sobre los estados es:

$$\mathbf{B}_{p,c} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -\dfrac{1}{C_f} & 0 \\ 0 & -\dfrac{1}{C_f} \end{bmatrix}.$$

En estas expresiones, R_f, L_f y C_f representan la resistencia, inductancia y capacitancia del filtro LC, respectivamente. Para este caso se utilizan $R_f = 0,065\ \Omega$, $L_f = 3$ mH y $C_f = 15\ \mu$F, con una frecuencia de red de 50 Hz ($\omega_b = 100\pi$ rad/s).

Para la carga resistiva considerada, el vector de perturbaciones se calcula en cada instante de muestreo como:

$$\mathbf{d}_k = \begin{bmatrix} v_{d,k}/R_L(k) \\ v_{q,k}/R_L(k) \end{bmatrix},$$

por lo que sus componentes corresponden a las corrientes demandadas por la carga en los ejes d y q. De esta manera, las variaciones de R_L modifican $\mathbf{d}_k$, mientras que las matrices que describen el filtro permanecen constantes.

Las matrices continuas $\mathbf{A}_c$, $\mathbf{B}_c$ y $\mathbf{B}_{p,c}$ se discretizan mediante un retenedor de orden cero (ZOH) con un período de muestreo de $T_s = 200\ \mu$s, obteniendo $\mathbf{A}$, $\mathbf{B}$ y $\mathbf{B}_p$, respectivamente. El modelo discreto utilizado en la simulación queda entonces dado por:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{B}_p \mathbf{d}_k.$$

Características del controlador predictivo

En cada instante de muestreo, el controlador recibe los cuatro estados de la planta, las dos referencias asociadas a los voltajes v_d y v_q y las dos componentes del vector de perturbaciones medibles $\mathbf{d}_k$. A partir de estas variables se formula el problema QP y se calculan las actuaciones u_d y u_q.

Para el sistema DER se utiliza un horizonte de predicción $N = 2$. Con cuatro estados ($N_{\text{SYS}} = 4$) y dos actuaciones ($M_{\text{SYS}} = 2$), la formulación densa genera la siguiente cantidad de variables de decisión:

$$N_{\text{QP}} = N \cdot M_{\text{SYS}} = 2 \cdot 2 = 4.$$

A diferencia del motor DC, las restricciones sobre las corrientes y las actuaciones no se expresan como límites independientes para cada componente. En el plano dq, los límites físicos corresponden a restricciones sobre la magnitud de los vectores:

$$\sqrt{i_d^2 + i_q^2} \leq I_{\text{nom}}, \qquad I_{\text{nom}} = 8 \text{ A},$$

$$\sqrt{u_d^2 + u_q^2} \leq U_{\text{máx}}, \qquad U_{\text{máx}} = \frac{V_{\text{dc}}}{\sqrt{3}} \approx 115,47 \text{ V},$$

donde $V_{\text{dc}} = 200$ V.

Estas restricciones definen regiones circulares y, por lo tanto, no pueden incorporarse directamente como desigualdades lineales en el problema QP. Para obtener una formulación compatible con el controlador, cada circunferencia se aproxima mediante un polígono regular de 10 lados. De esta forma, cada magnitud restringida da lugar a 10 desigualdades lineales por instante del horizonte.

Al considerar las restricciones sobre las corrientes y las actuaciones, el problema incorpora 20 desigualdades por cada instante del horizonte. En consecuencia, la cantidad total de restricciones queda dada por:

$$M_{\text{QP}} = 20N = 40.$$

Aunque el número de variables de decisión coincide con el utilizado en el motor DC, la matriz de restricciones es de mayor tamaño debido a la aproximación politópica. Como resultado, aumenta la cantidad de operaciones asociadas a las restricciones durante cada iteración de ADMM.

5.3.2. Exploración de formatos de punto fijo en MATLAB

La simulación se realiza durante $0,5$ s, con condición inicial $\mathbf{x}_0 = \mathbf{0}$. Las referencias de voltaje se mantienen constantes en $v_d^* = 100$ V y $v_q^* = 0$ V.

Para evaluar la respuesta del controlador frente a cambios en las condiciones de operación, la resistencia de carga se modifica durante la simulación, según se muestra en la Figura 5.10. El valor inicial de $47,2\ \Omega$ disminuye a $10\ \Omega$ en $t = 0,1$ s, aumenta a $30\ \Omega$ en $t = 0,3$ s y cambia finalmente a $20\ \Omega$ en $t = 0,4$ s. Estas variaciones modifican la corriente demandada por la carga.

La Figura 5.11 presenta la respuesta temporal del sistema DER bajo control MPC en punto flotante. En el gráfico superior se muestran los voltajes del condensador en los ejes d y q, junto con sus respectivas referencias.

Los cambios en la resistencia de carga generan transitorios en ambos voltajes. En particular, cuando la resistencia disminuye a $10\ \Omega$ en $t = 0,1$ s, aumenta la corriente requerida por la carga hasta alcanzar el límite nominal. Durante este intervalo, la restricción de corriente limita las trayectorias admisibles del controlador y el voltaje v_d permanece por debajo de su referencia, alrededor de 80 V. Al aumentar la resistencia a $30\ \Omega$ en $t = 0,3$ s, disminuye la corriente demandada y el voltaje retorna a un valor cercano a su referencia. El cambio aplicado en $t = 0,4$ s produce un nuevo transitorio de menor magnitud.

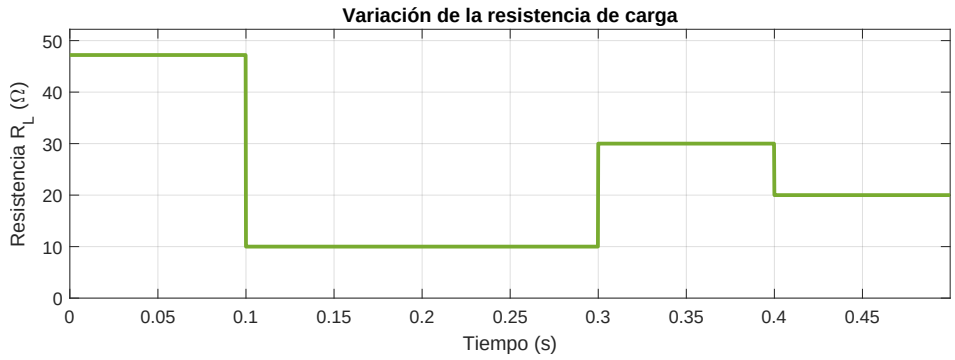


Figura 5.10: Variación de la resistencia de carga aplicada durante la simulación del sistema DER. Los cambios introducidos en $t = 0,1$ s, $t = 0,3$ s y $t = 0,4$ s permiten evaluar la respuesta del controlador en distintos puntos de operación.

El gráfico central presenta las corrientes del inductor del filtro. La componente i_d modifica su nivel de acuerdo con la carga conectada, alcanzando sus valores más altos durante el intervalo en que la resistencia es menor. La componente i_q, en cambio, permanece cercana a cero durante la mayor parte de la simulación. Finalmente, el gráfico inferior muestra las actuaciones u_d y u_q calculadas por el controlador, las cuales se ajustan ante cada cambio de carga para regular los voltajes del condensador.

La Figura 5.12 muestra las trayectorias de las actuaciones y de las corrientes del filtro en los planos dq. En el gráfico izquierdo, la trayectoria formada por u_d y u_q se compara con el límite circular asociado a la tensión máxima disponible y con las rectas que definen la aproximación politópica utilizada por el controlador. La actuación se concentra principalmente sobre el eje d y permanece dentro del límite circular durante la simulación.

En el gráfico derecho se representa la trayectoria de las corrientes i_d e i_q respecto del límite nominal I_{nom}. La corriente se desarrolla principalmente sobre el eje d, mientras que la componente i_q conserva valores reducidos. Durante los transitorios asociados a los cambios de carga, la trayectoria supera temporalmente el límite nominal, especialmente después de la disminución de la resistencia en $t = 0,1$ s. Este sobrepaso también está presente en la implementación de referencia y se relaciona con la solución aproximada obtenida al utilizar un número fijo de iteraciones de ADMM. En [25] se muestra que el efecto del pico de corriente puede reducirse al aumentar la cantidad de iteraciones. En este trabajo se conservan las 10 iteraciones utilizadas en el controlador de referencia.

La respuesta obtenida en la simulación de punto flotante se utiliza como referencia para explorar los formatos de punto fijo. En este caso, las métricas de la Sección 5.1.4 se calculan por separado para u_d y u_q, y se interpretan junto con la regulación de los voltajes, la evolución de las corrientes y el cumplimiento de las restricciones de operación.

A diferencia del motor DC, en este caso no se establece un umbral escalar de error para las actuaciones. La selección se realiza a partir de la cercanía de las trayectorias en lazo cerrado, la regulación de los voltajes y la ausencia de nuevas violaciones de restricciones respecto de la referencia en punto flotante.

Inicialmente se asignan 16 bits a la parte entera, cantidad suficiente para cubrir los rangos observados durante la simulación sin introducir desbordamientos. A partir de este valor, se ajusta la cantidad de bits fraccionales para obtener una resolución suficiente para el controlador.

La exploración condujo al formato $\text{Q}_{16,11}$, correspondiente a una longitud total de palabra de 27 bits. La Figura 5.13 compara la respuesta temporal del sistema en punto flotante y punto fijo para el formato seleccionado. En los voltajes del condensador, las corrientes del filtro y las actuaciones, se observa una superposición cercana entre ambas representaciones, tanto durante los intervalos de régimen permanente como frente a los cambios de carga. Las principales diferencias se concentran en los transitorios generados al inicio de la simulación y en los instantes $t = 0,1$ s, $t = 0,3$ s y $t = 0,4$ s.

La Figura 5.14 presenta las trayectorias de las actuaciones y de las corrientes del filtro en los planos

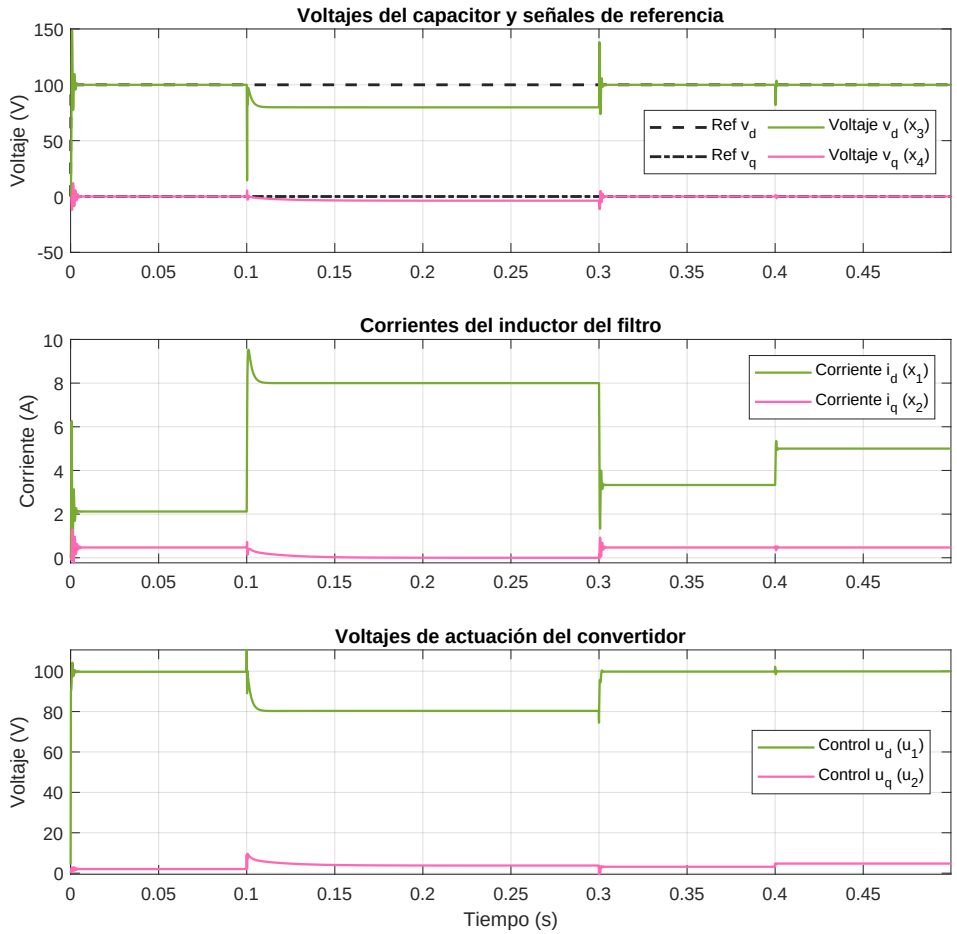


Figura 5.11: Respuesta temporal del sistema DER en lazo cerrado utilizando el controlador MPC en punto flotante. Se muestran los voltajes del condensador y sus referencias, las corrientes del inductor del filtro y las actuaciones calculadas por el controlador.

dq. En el gráfico izquierdo, ambas trayectorias permanecen dentro del límite de actuación y siguen recorridos cercanos, aunque el detalle ampliado permite observar pequeñas diferencias durante los transitorios. En el gráfico derecho, las corrientes superan temporalmente el límite nominal en ambas representaciones. El zoom muestra que la trayectoria en punto fijo reproduce de manera cercana el comportamiento del modelo en punto flotante incluso en la zona donde se alcanza la mayor corriente.

Como complemento a esta comparación, para u_d se obtiene un error absoluto máximo de $5,2752$ V y medio de $0,0889$ V, mientras que para u_q estos valores son $2,0191$ V y $0,0808$ V, respectivamente. La diferencia entre los errores máximos y medios indica que las mayores discrepancias se concentran en instantes puntuales.

Los resultados muestran que $Q_{16,11}$ conserva el comportamiento general del controlador en lazo cerrado y reproduce las principales características de la referencia en punto flotante. Por esta razón, el formato seleccionado se utiliza como base para el traslado del controlador a Vitis HLS y su posterior implementación en hardware.

5.3.3. Traslado de la representación numérica a Vitis HLS

Una vez seleccionado el formato $Q_{16,11}$ en MATLAB, el controlador se traslada a C/C++ para su síntesis mediante Vitis HLS, utilizando tipos de punto fijo y modos de cuantización y desbordamiento equivalentes a los definidos durante la exploración numérica.

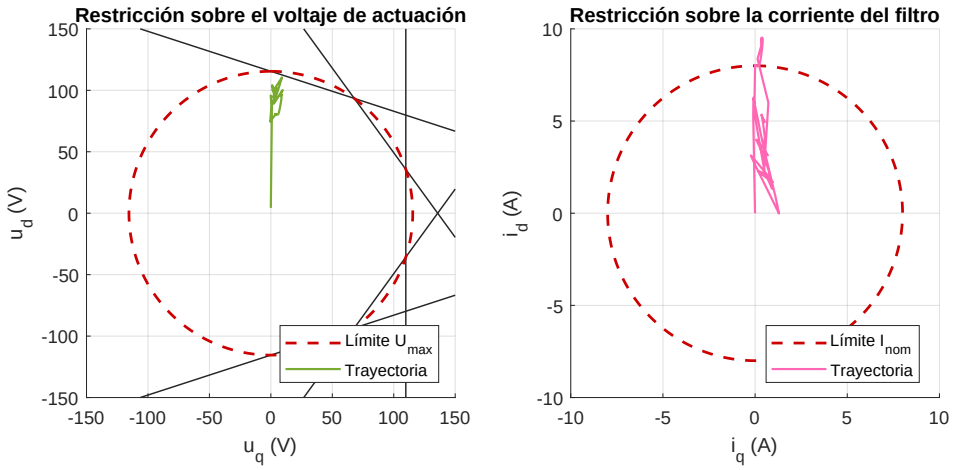


Figura 5.12: Trayectorias de las actuaciones y de las corrientes del filtro en los planos dq para la simulación en punto flotante. El gráfico izquierdo presenta la trayectoria de actuación, el límite circular y su aproximación politópica, mientras que el gráfico derecho muestra la corriente del filtro respecto del límite nominal.

Tabla 5.3: Errores absolutos de las actuaciones u_d y u_q para las implementaciones evaluadas.

Comparación	Señal	Error máximo [V]	Error medio [V]
MATLAB fijo respecto de flotante	u_d	5,2752	0,0889
MATLAB fijo respecto de flotante	u_q	2,0191	0,0808
Vitis HLS respecto de flotante	u_d	5,2884	0,0901
Vitis HLS respecto de flotante	u_q	2,0176	0,0814
Vitis HLS respecto de MATLAB fijo	u_d	0,4917	0,0025
Vitis HLS respecto de MATLAB fijo	u_q	0,0112	0,0024

La verificación inicial considera los estados y las actuaciones obtenidos con tres implementaciones, el modelo de referencia en punto flotante, la versión en punto fijo desarrollada en MATLAB y la implementación en punto fijo de Vitis HLS. Las respuestas de los estados presentaron una superposición similar a la observada en la Subsección 5.3.2, por lo que el análisis se concentra en las actuaciones u_d y u_q, donde las diferencias pueden evaluarse directamente a la salida del controlador.

La Figura 5.16 compara las actuaciones calculadas por las tres implementaciones. Los gráficos superiores muestran una evolución prácticamente coincidente en la escala completa, mientras que las ampliaciones permiten distinguir las diferencias que aparecen durante los transitorios. En los gráficos inferiores se presenta el error absoluto de Vitis HLS respecto de la referencia en punto flotante y de la implementación en punto fijo de MATLAB.

La Tabla 5.3 resume el error absoluto máximo y el error absoluto medio para las comparaciones evaluadas. El error de MATLAB en punto fijo respecto de la referencia en punto flotante representa el efecto introducido por la cuantización seleccionada, mientras que la comparación de Vitis HLS respecto de esa misma referencia reúne dicho efecto y las diferencias asociadas al traslado a C/C++. Por otra parte, la comparación directa entre las versiones en punto fijo permite evaluar la cercanía entre las trayectorias obtenidas en ambos entornos. Como cada implementación evoluciona en su propio lazo cerrado, las diferencias observadas incluyen tanto las discrepancias numéricas introducidas durante el traslado como su propagación hacia los estados de muestras posteriores.

Los errores de Vitis HLS respecto de la referencia en punto flotante se mantienen cercanos a los obtenidos con MATLAB en punto fijo. La comparación directa entre ambas implementaciones en punto fijo presenta errores medios de $0,0025$ V para u_d y $0,0024$ V para u_q. En u_d se observa una diferencia

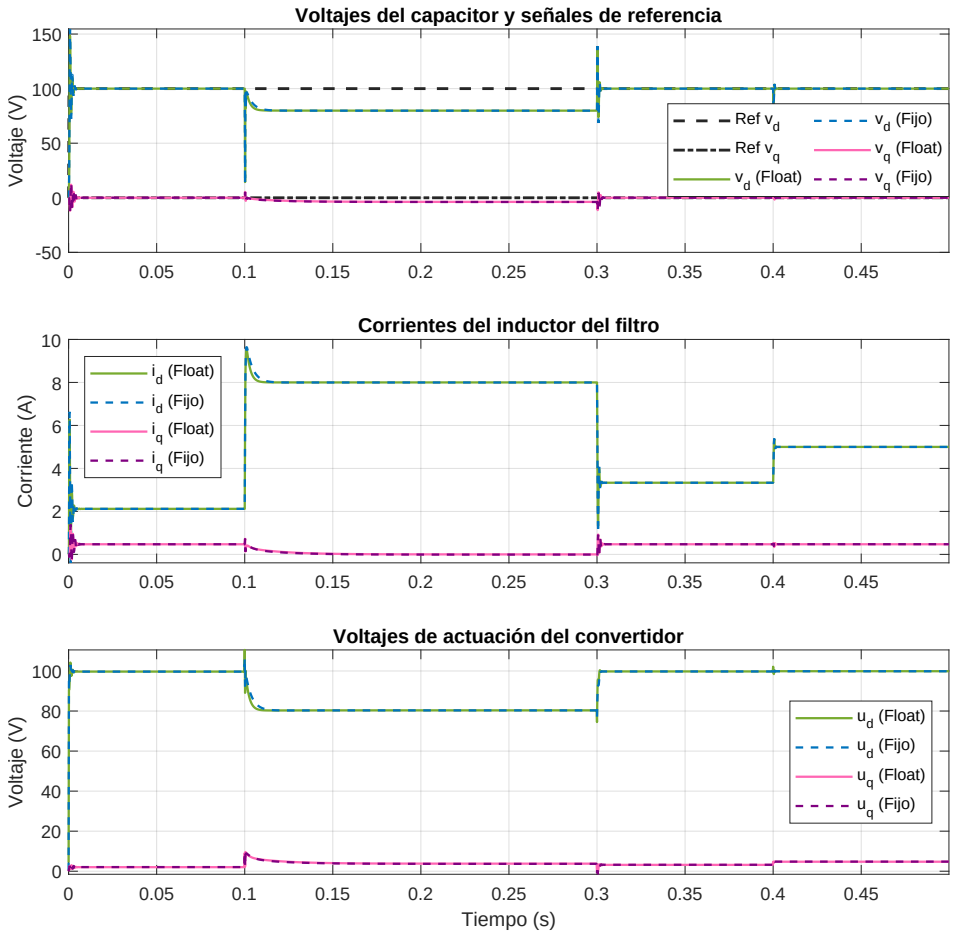


Figura 5.13: Comparación de la respuesta temporal del sistema DER en punto flotante y punto fijo con formato $Q_{16,11}$. Se muestran los voltajes del condensador y sus referencias, las corrientes del inductor del filtro y las actuaciones calculadas por el controlador.

máxima puntual de $0,4917$ V, concentrada durante los transitorios asociados a los cambios de carga. En conjunto, estos resultados indican que el traslado a Vitis HLS conserva de manera cercana el comportamiento numérico definido en MATLAB.

Una vez verificada esta correspondencia, se evalúa el efecto de la representación sobre la arquitectura generada. Para ello, se sintetizan tres versiones del controlador utilizando las mismas directivas de optimización descritas para el caso del motor DC, una implementación en punto flotante, un formato $Q_{16,16}$ y el formato seleccionado $Q_{16,11}$. En todos los casos se utilizó una FPGA Artix-7 100T como dispositivo objetivo y un período de reloj de 10 ns.

El formato $Q_{16,16}$ conserva los 16 bits enteros utilizados durante la exploración y completa una longitud total de 32 bits. Esta alternativa se incluye como referencia de punto fijo de mayor resolución, con el propósito de separar el efecto del cambio desde punto flotante del efecto adicional de reducir la longitud de palabra hasta los 27 bits de $Q_{16,11}$.

La Tabla 5.4 presenta el uso de LUTs, FFs y DSPs, junto con la cantidad de ciclos, la latencia y la frecuencia de cálculo estimada para cada alternativa. Como ninguna de las versiones utiliza bloques BRAM, este recurso se omite de la comparación. Los resultados en punto fijo corresponden a la etapa posterior a la implementación, mientras que para la versión en punto flotante solo se dispone de la estimación posterior a la síntesis, debido a que su demanda de DSP impidió completar la implementación en el dispositivo seleccionado.

Los resultados muestran que las representaciones en punto fijo permiten implementar el controlador

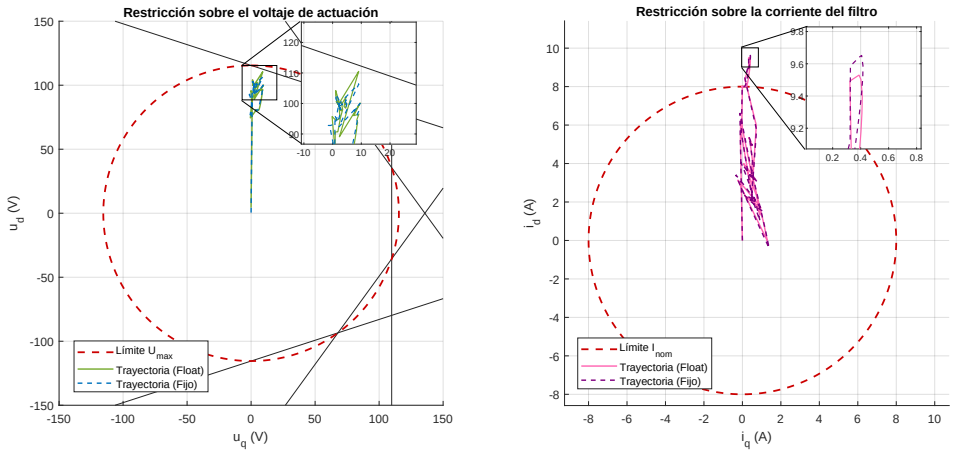


Figura 5.14: Comparación de las trayectorias de actuación y corriente para las implementaciones en punto flotante y punto fijo con formato $Q_{16,11}$. El gráfico izquierdo muestra el límite de actuación y su aproximación politópica, mientras que el gráfico derecho presenta la trayectoria de corriente respecto del límite nominal.

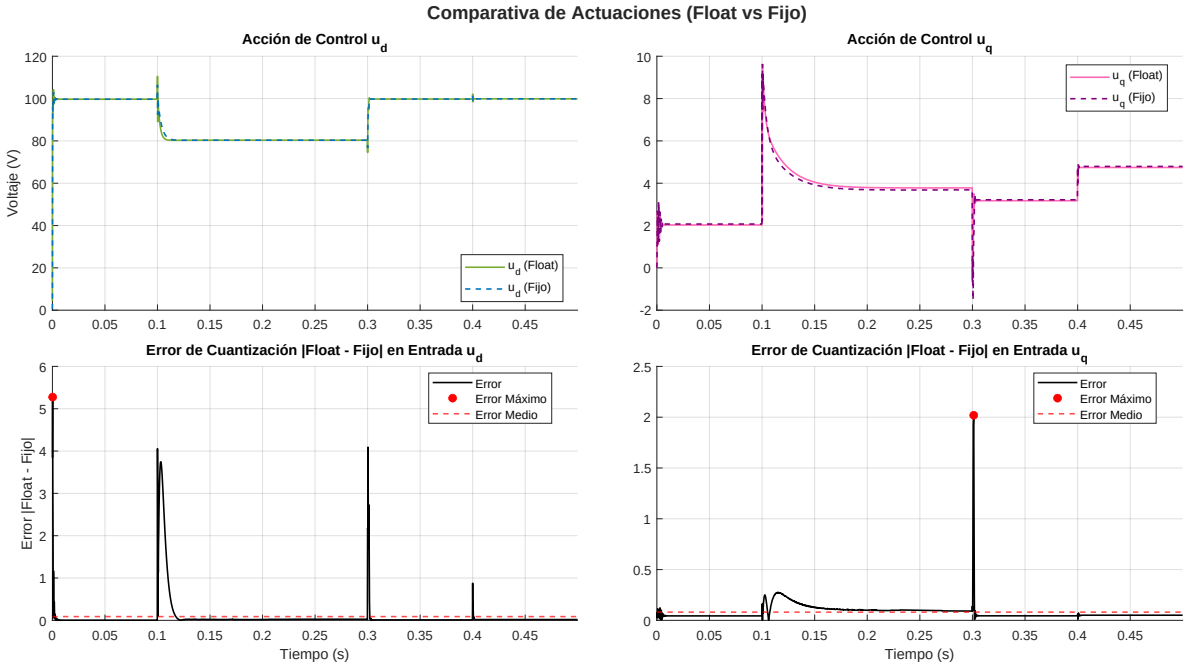


Figura 5.15: Comparación de las actuaciones u_d y u_q en punto flotante y punto fijo con formato $Q_{16,11}$, junto con sus errores absolutos. En cada caso se indican el error absoluto máximo y el error absoluto medio respecto de la referencia en punto flotante.

en la Artix-7 100T, mientras que la versión en punto flotante excede la cantidad de bloques DSP disponibles con la configuración de síntesis evaluada.

Entre las alternativas en punto fijo, el formato $Q_{16,11}$ reduce principalmente el uso de FF y DSP respecto de $Q_{16,16}$, mientras que la cantidad de LUT se mantiene prácticamente sin cambios. También presenta una menor latencia y una frecuencia de cálculo ligeramente superior. En ambos casos, el tiempo de ejecución se mantiene por debajo del período de muestreo de 200 μs utilizado por el sistema DER.

Como referencia adicional, en [25] la alternativa en punto flotante de menor latencia reportada

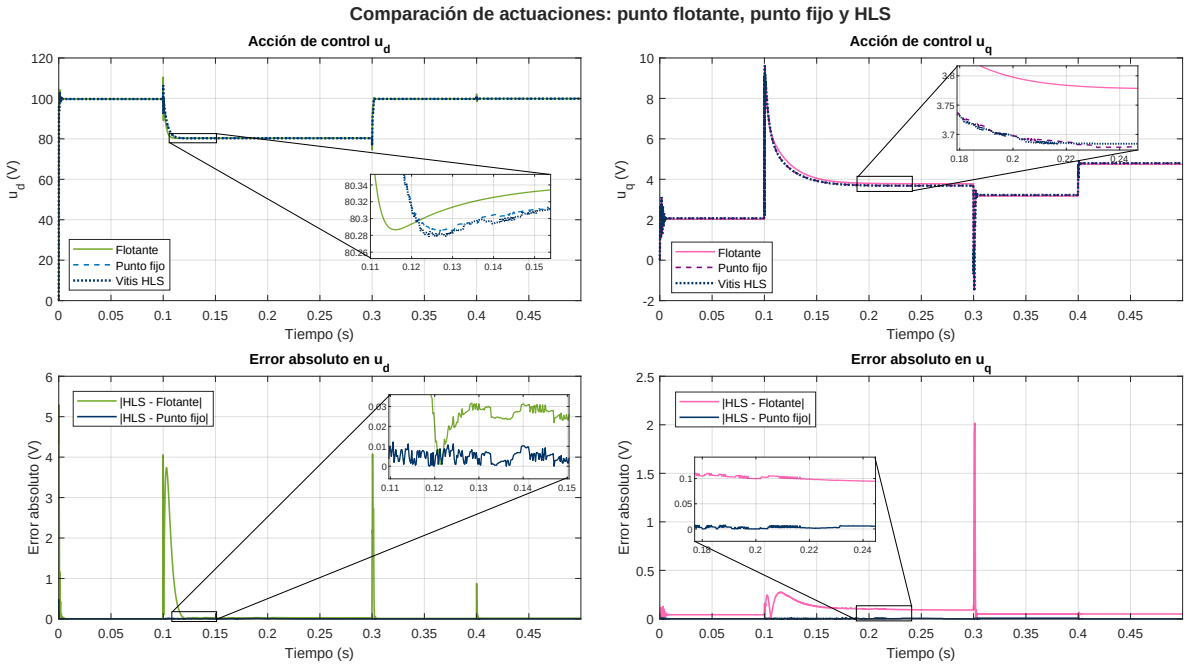


Figura 5.16: Comparación de las actuaciones u_d y u_q obtenidas con el modelo de referencia en punto flotante, la implementación en punto fijo de MATLAB y la implementación en punto fijo de Vitis HLS para el formato $Q_{16,11}$. Los gráficos inferiores presentan el error absoluto de Vitis HLS respecto de la referencia en punto flotante y respecto de la implementación en punto fijo de MATLAB.

Tabla 5.4: Uso de recursos y desempeño de las implementaciones del controlador DER. Las representaciones en punto fijo corresponden a resultados posteriores a la implementación, mientras que los valores en punto flotante corresponden a la síntesis. Entre paréntesis se indica el porcentaje respecto de los recursos disponibles en la FPGA Artix-7 100T.

Representación	LUTs	FFs	DSPs	Ciclos	Latencia [μs]	f_c [kHz]
Flotante	28701 (45,3 %)	41397 (32,6 %)	262 (109,2 %)	2550	25,50	39,22
$Q_{16,16}$	14763 (23,3 %)	16074 (12,7 %)	104 (43,3 %)	1924	19,24	51,98
$Q_{16,11}$	14859 (23,4 %)	13344 (10,5 %)	66 (27,5 %)	1890	18,90	52,91

para el controlador DER sobre una plataforma ZCU104 requiere $18,5$ μs por ejecución, equivalente a una frecuencia de cálculo aproximada de $54,05$ kHz. Al implementar el formato $Q_{16,11}$ sobre la misma FPGA y mantener un período objetivo de 10 ns, el controlador requiere 1289 ciclos, correspondientes a una latencia de $12,89$ μs y una frecuencia de cálculo estimada de $77,58$ kHz. Esta comparación muestra que, sobre el mismo dispositivo, la representación en punto fijo reduce la latencia respecto de la alternativa en punto flotante reportada previamente.

Capítulo 6

CONCLUSIONES Y TRABAJO FUTURO

Esta tesis desarrolló y evaluó una metodología para seleccionar representaciones en punto fijo durante etapas tempranas del diseño de controladores MPC y trasladarlas posteriormente a una implementación en FPGA mediante HLS. A partir de los casos de estudio evaluados, se analizan las principales conclusiones sobre la correspondencia numérica entre herramientas, la selección de la representación y su efecto sobre el comportamiento en lazo cerrado, el uso de recursos y la latencia de la arquitectura resultante. Finalmente, se plantean líneas de trabajo futuro orientadas a ampliar y automatizar el flujo propuesto.

6.1. Conclusiones

El flujo propuesto relaciona la exploración numérica realizada en MATLAB con las distintas etapas de implementación en C/C++, RTL y hardware, manteniendo la trazabilidad de las diferencias numéricas introducidas durante el proceso. En el caso del producto punto, se obtuvo coincidencia bit a bit entre MATLAB y Vitis HLS al reproducir tanto la representación numérica como la estructura de cálculo, incluido el orden de evaluación y los puntos de almacenamiento de los resultados intermedios. Este resultado muestra que, cuando se busca equivalencia numérica entre herramientas, no basta con reproducir el formato de las variables, sino que también deben conservarse la configuración de las operaciones y la estructura de cálculo.

Cuando no se requiere coincidencia bit a bit entre MATLAB y C/C++, mantener la misma representación numérica sigue siendo útil para la exploración temprana. En los casos evaluados, descomponer las expresiones de modo que cada asignación contuviera una sola operación y almacenar su resultado en el formato correspondiente permitió que la evaluación realizada en MATLAB siguiera siendo válida al trasladar las representaciones candidatas a Vitis HLS. Las representaciones que satisfacían en MATLAB los criterios definidos para el controlador también los satisfacían en Vitis HLS, mientras que aquellas que no los cumplían podían descartarse antes de la síntesis. Las diferencias numéricas restantes se identificaron y cuantificaron durante las etapas de verificación. De esta forma, el modelo de alto nivel permite reducir tempranamente el conjunto de representaciones que deben llevarse a las etapas de síntesis e implementación en FPGA.

Los resultados obtenidos en lazo cerrado mostraron que, para los casos evaluados, cubrir los rangos observados durante la simulación y satisfacer una métrica de error en un único escenario no fue suficiente para seleccionar una representación numérica. Una representación que presentó un comportamiento aceptable bajo una determinada condición de operación produjo diferencias mayores al evaluar un transitorio más exigente. En lazo cerrado, las diferencias en la actuación modifican la

evolución de la planta y, por tanto, los estados que recibe el controlador en instantes posteriores, por lo que sus efectos pueden propagarse a lo largo de la simulación. Por esta razón, la evaluación no debe limitarse al error respecto de la referencia en punto flotante, sino que debe considerar también el seguimiento de referencia, la evolución de los estados y de las actuaciones, y el cumplimiento de las restricciones bajo distintas condiciones representativas de operación. Asimismo, las eventuales violaciones de restricciones deben analizarse en comparación con la implementación de referencia. Si una violación ya está presente en punto flotante, su existencia no puede atribuirse a la reducción de precisión, aunque sí debe verificarse si la representación en punto fijo modifica su magnitud o duración.

Desde el punto de vista de la implementación, los casos evaluados mostraron que el uso de aritmética de punto fijo puede reducir el uso de recursos y la latencia de cálculo. En el caso del motor DC, esta reducción se observa incluso al comparar una representación en punto fijo de 32 bits con punto flotante de precisión simple, también de 32 bits, lo que muestra que las diferencias en la arquitectura resultante no dependen únicamente de la longitud de palabra. Al reducir posteriormente el formato de acuerdo con los requerimientos de rango y resolución de cada caso, se obtienen reducciones adicionales en el uso de recursos y en la cantidad de ciclos de cálculo, manteniendo la respuesta en lazo cerrado dentro de los criterios definidos. En conjunto, estos resultados muestran que la representación numérica influye tanto en la precisión del controlador como en el costo y desempeño temporal de la arquitectura resultante.

La exploración de formatos mostró también que la longitud total de palabra no determina por sí sola el costo de la arquitectura. Al mantener fija una longitud de 32 bits y modificar la distribución entre bits enteros y fraccionales, se observaron variaciones en el uso de LUT, FF y DSP, así como en la latencia de cálculo. En el barrido realizado, una mayor cantidad de bits fraccionales tendió a asociarse con un mayor uso de recursos y una mayor latencia, con cambios escalonados entre distintas configuraciones. Estos resultados muestran que la posición del punto binario puede influir en la arquitectura generada por Vitis HLS, aun cuando la longitud total de palabra se mantenga constante. La tendencia observada depende de las operaciones del controlador y del mapeo realizado por la herramienta, por lo que se requieren más casos de estudio antes de generalizar este comportamiento.

En conjunto, los resultados respaldan la hipótesis planteada en esta tesis. Fue posible seleccionar representaciones en punto fijo durante etapas tempranas del diseño, trasladarlas al entorno HLS y verificar las diferencias numéricas introducidas durante este proceso, manteniendo la respuesta en lazo cerrado dentro de los criterios establecidos. El uso de punto fijo permitió además reducir el uso de recursos y la latencia respecto de las implementaciones en punto flotante evaluadas. En el caso con mayor demanda de hardware, esta reducción permitió implementar el controlador en una FPGA con menor disponibilidad de recursos que la requerida por la alternativa en punto flotante. Estos resultados muestran que una representación numérica seleccionada y trasladada de manera consistente entre las distintas etapas del flujo puede reducir el costo de hardware y el tiempo de cálculo sin degradar la respuesta del controlador más allá de los criterios definidos.

6.2. Trabajo futuro

El flujo desarrollado deja abiertas posibilidades para ampliar tanto el espacio de diseño considerado como el nivel de automatización y validación experimental.

Una primera línea de trabajo corresponde al desarrollo de un *framework* que integre y automatice las principales etapas del flujo propuesto. Este entorno podría incorporar la recopilación de rangos durante las simulaciones, la generación y evaluación de formatos candidatos, el traslado de las configuraciones seleccionadas hacia los tipos de punto fijo utilizados por herramientas HLS y la comparación sistemática de resultados entre implementaciones. Una infraestructura de este tipo permitiría acelerar la exploración del espacio de diseño, evaluar un mayor número de formatos y facilitar la incorporación de nuevos casos de estudio, proporcionando además una base para estudiar estrategias más sistemáticas de selección de representaciones numéricas.

Una segunda extensión corresponde al uso de formatos de punto fijo heterogéneos dentro de un mismo controlador. En los casos de estudio se adoptó una representación homogénea, donde las variables internas utilizan un mismo formato $Q_{m,n}$. Esta decisión permitió aislar el efecto de la representación numérica y simplificar su traslado entre herramientas, pero no explota las diferencias

de rango presentes entre las variables y operaciones del algoritmo. Asignar formatos específicos a distintos grupos de variables o resultados intermedios podría reducir adicionalmente el uso de recursos o aumentar la precisión sin incrementar de igual manera el costo de toda la arquitectura. Esta extensión requeriría, a su vez, estrategias de exploración que permitan manejar el aumento del número de formatos posibles.

Finalmente, la validación experimental puede extenderse hacia condiciones más próximas a una aplicación física. Las pruebas mediante FPGA-in-the-Loop realizadas en este trabajo permiten ejecutar el controlador sintetizado sobre la FPGA mientras la planta permanece representada en el entorno de simulación. Como continuación, el controlador podría evaluarse mediante Hardware-in-the-Loop, utilizando un modelo de la planta ejecutado en tiempo real e incorporando los retardos e interfaces propios de una plataforma embebida. Una etapa posterior podría integrar el controlador directamente con una planta física, incluyendo efectos que fueron excluidos del análisis realizado en esta tesis, como ruido de medición, cuantización de conversores, estimación de estados y retardos de adquisición y actuación. Esta validación permitiría estudiar si las representaciones seleccionadas durante las etapas tempranas mantienen el desempeño observado cuando el controlador interactúa con las condiciones propias de una implementación real.

BIBLIOGRAFÍA

[1] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017. [Online]. Available: https://books.google.cl/books?id=MrJctAEACAAJ

[2] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109814005160

[3] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior-point methods to model predictive control," *Journal of Optimization Theory and Applications*, vol. 99, no. 3, pp. 723–757, 1998. [Online]. Available: https://doi.org/10.1023/A:1021711402723

[4] S. Vazquez, J. I. Leon, L. G. Franquelo, J. Rodriguez, H. A. Young, A. Marquez, and P. Zanchetta, "Model predictive control: A review of its applications in power electronics," *IEEE Industrial Electronics Magazine*, vol. 8, no. 1, pp. 16–31, 2014.

[5] S. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733–764, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0967066102001867

[6] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded online optimization for model predictive control at megahertz rates," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3238–3251, 2014.

[7] 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.

[8] H. Peyrl, A. Zanarini, T. Besselmann, J. Liu, and M.-A. Boéchat, "Parallel implementations of the fast gradient method for high-speed MPC," *Control Engineering Practice*, vol. 33, pp. 22–34, 2014.

[9] P. Zhang, J. Zambreno, and P. H. Jones, "An embedded scalable linear model predictive hardware-based controller using ADMM," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2017, pp. 176–183.

[10] 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 *IEEE International Conference on Automation/XXV Congress of the Chilean Association of Automatic Control (ICA-ACCA)*, 2022, pp. 1–6.

[11] M.-A. Boéchat, J. Liu, H. Peyrl, A. Zanarini, and T. Besselmann, "An architecture for solving quadratic programs with the fast gradient method on a field programmable gate array," in *21st Mediterranean Conference on Control and Automation*, 2013, pp. 1557–1562.

[12] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are we there yet? a study on the state of high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 898–911, 2019.

[13] 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)*, pp. 1704–1711.

[14] S. Lucia, D. Navarro, 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.

[15] 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.

[16] D. Menard, N. Herve, O. Sentieys, and H.-N. Nguyen, "High-level synthesis under fixed-point accuracy constraint," *Journal of Electrical and Computer Engineering*, no. 1, p. 906350, 2012.

[17] A. Finnerty and H. Ratigner, "Reduce power and cost by converting from floating point to fixed point, xilinx white paper, wp491 (v1.0)," 2017.

[18] *fimath: Set fixed-point math settings*, The MathWorks, Inc., 2023, MATLAB R2023b Documentation. [Online]. Available: https://www.mathworks.com/help/fixedpoint/ref/embedded.fimath.html

[19] AMD, *Vitis High-Level Synthesis User Guide*, Advanced Micro Devices, Inc., Jul. 2024, version 2024.1, sección "C++ Arbitrary Precision Fixed-Point Types". [Online]. Available: https://docs.amd.com/r/2024.1-English/ug1399-vitis-hls/C-Arbitrary-Precision-Fixed-Point-Types

[20] I. Winandy, A. Dion, F. Manni, P.-L. Garoche, D. B. Khalifa, and M. Martel, "Automated fixed-point precision optimization for FPGA synthesis," *IEEE Open Journal of Circuits and Systems*, vol. 6, pp. 192–204, 2025.

[21] S. Gerkšič and B. Pregelj, "Finite-word-length FPGA implementation of model predictive control for ITER resistive wall mode control," *Fusion Engineering and Design*, vol. 169, p. 112480, 2021.

[22] P. J. Mosterman, "Model-based design of embedded systems," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*. IEEE, 2007, pp. 3–3.

[23] T. Erkkinen and M. Conrad, "Verification, validation, and test with model-based design," in *Commercial Vehicle Engineering Congress & Exhibition*, no. 2008-01-2709. SAE International, 2008.

[24] M. Purraji, E. Zamiri, and A. de Castro, "Easy and straightforward FPGA implementation of model predictive control using HDL coder," *Electronics*, vol. 14, no. 3, 2025. [Online]. Available: https://www.mdpi.com/2079-9292/14/3/419

[25] A. N. Cortés Neira, "Implementación de aceleradores de cómputo en lógica reconfigurable para aplicaciones de control predictivo por modelo utilizando síntesis de alto nivel," Tesis de Magíster, Universidad Técnica Federico Santa María, Valparaíso, Chile, mar 2024.

[26] F. Donoso, N. Salvador, C. A. Rojas, J. Marín, and G. Carvajal, "Design of a pi-controller asic for a multilevel converter using high-level synthesis and librelane," in *2026 IEEE 17th Latin America Symposium on Circuits and System (LASCAS)*, 2026, pp. 1–5.

[27] 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.

[28] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan, "FPGA implementation of an interior point solver for linear model predictive control," in *2010 International Conference on Field-Programmable Technology*, 2010, pp. 316–319.

[29] MathWorks, "FPGA-in-the-Loop Simulation," HDL Verifier Documentation. [Online]. Available: https://www.mathworks.com/help/hdlverifier/ug/fpga-in-the-loop-fil-simulation.html

[30] H. A. Shukla, B. Khusainov, E. C. Kerrigan, and C. N. Jones, "Software and hardware code generation for predictive control using splitting methods," in *Proceedings of the 20th IFAC World Congress*, 2017, pp. 14 386–14 391.

[31] P. Patrinos, A. Guiggiani, and A. Bemporad, "Fixed-point dual gradient projection for embedded model predictive control," in *2013 European Control Conference (ECC)*, 2013, pp. 3602–3607.

[32] P. Krupa, O. Inverso, M. Tribastone, and A. Bemporad, "Certification of the proximal gradient method under fixed-point arithmetic for box-constrained QP problems," *Automatica*, vol. 159, p. 111411, 2024.

[33] S. Longo, E. C. Kerrigan, and G. A. Constantinides, "Constrained LQR for low-precision data representation," *Automatica*, vol. 50, no. 1, pp. 162–168, 2014.

[34] Y. Wang, M. Soltani, and D. M. A. Hussain, "Attitude stabilization of marine satellite tracking antenna using model predictive control," *IFAC Journal of Systems and Control*, vol. 17, p. 100173, 2021.

[35] H. Peyrl, H. J. Ferreau, and D. Kouzoupis, "A hybrid hardware implementation for nonlinear model predictive control," *IFAC-PapersOnLine*, vol. 48, no. 23, pp. 87–93, 2015.

[36] The MathWorks, Inc., "Data types and scaling in digital hardware," MATLAB Help Center, Fixed-Point Designer. [Online]. Available: https://www.mathworks.com/help/fixedpoint/ug/data-types-and-scaling-in-digital-hardware.html

[37] Texas Instruments, *TMS320C62x DSP Library Programmer's Reference*, Texas Instruments, April 2002, apéndice A: Fractional Q Formats. [Online]. Available: https://www.ti.com/lit/ug/spru402b/spru402b.pdf

[38] MathWorks, "fi — construct fixed-point numeric object," MATLAB Help Center (Fixed-Point Designer Documentation), 2025. [Online]. Available: https://www.mathworks.com/help/fixedpoint/ref/embedded.fi.html

[39] Texas Instruments, "How to read and interpret digital temperature sensor output data," Texas Instruments, Application Report SBAA588A, 1 2025. [Online]. Available: https://www.ti.com/lit/an/sbaa588a/sbaa588a.pdf

[40] S. L. Harris and D. Harris, *Digital Design and Computer Architecture*, 3rd ed. Cambridge, MA: Morgan Kaufmann, 2022, ch. 5, p. 256, RISC-V Edition.

[41] J.-L. Lions *et al.*, "Ariane 5 flight 501 failure: Report by the inquiry board," European Space Agency (ESA), Paris, France, Tech. Rep., 1996, sección 2.1: Sequence of events, p. 7. [Online]. Available: https://esamultimedia.esa.int/docs/esa-x-1819eng.pdf

[42] J. J. Vásquez Cádiz, "Implementación en FPGA de redes neuronales artificiales aplicadas a lazos de control predictivo por modelo," Tesis de Magíster, Universidad Técnica Federico Santa María, Valparaíso, Chile, aug 2025.

[43] The MathWorks, Inc., "MATLAB Language Features Supported for Automated Fixed-Point Conversion," https://www.mathworks.com/help/coder/ug/matlab-language-features-supported-for-automated-fixed-point-conversion.html, 2026, accessed: 2026-06-10.

Apéndice A

APLICACIÓN COMPLEMENTARIA DEL FLUJO DE DISEÑO A UN CONTROLADOR PI

Este apéndice presenta una aplicación complementaria del flujo utilizado en esta tesis para trasladar algoritmos de control desde una referencia de alto nivel hacia una implementación con aritmética de punto fijo. El caso corresponde al diseño de un controlador PI para un convertidor de capacitor flotante de tres niveles (*Three-Level Flying Capacitor Converter*, 3L-FCC), desarrollado como parte del trabajo presentado en [26]. Aunque este desarrollo no utiliza MPC, sigue una secuencia similar a la propuesta en esta tesis: parte de una referencia funcional, explora representaciones de punto fijo mediante simulación, traslada la representación seleccionada a HLS y verifica su comportamiento antes de su implementación en hardware.

El objetivo final de esta aplicación fue extender el diseño hacia una implementación ASIC mediante un flujo basado en herramientas *open source*. Por este motivo, además de utilizar Vitis HLS como referencia, el controlador se trasladó a Bambu HLS. Para representar las variables en punto fijo se utilizó la biblioteca abierta *Algorithmic C Datatypes* (`ac_types`), desarrollada por Siemens. Su tipo `ac_fixed` presenta una interfaz de plantilla similar a `ap_fixed` de Vitis HLS, pero ambas bibliotecas no asignan necesariamente la misma precisión a los resultados intermedios de las operaciones. Por ello, la migración requirió revisar explícitamente estas diferencias e incorporar conversiones de tipo para evitar truncamientos no deseados. Este aspecto refuerza la necesidad de considerar tanto el formato como la configuración y la precisión de los resultados intermedios al trasladar una representación numérica entre herramientas.

El trabajo incorpora finalmente una etapa de diseño físico mediante LibreLane y el PDK SKY130. Esta extensión permite relacionar las decisiones tomadas durante la exploración numérica con métricas de implementación como el área y el consumo de potencia.

A.1. Sistema de referencia

El caso de estudio considera el control de un convertidor 3L-FCC mediante dos controladores PI. La implementación original del sistema se encontraba descrita en PLECS, con los controladores implementados mediante bloques C-script. Para establecer una referencia común para las etapas posteriores, el comportamiento de los controladores se reprodujo en MATLAB y se verificó su consistencia con el modelo original.

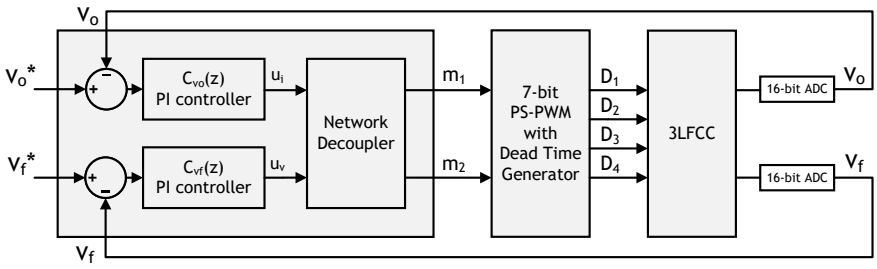


Figura A.1: Arquitectura general del sistema de control utilizado para el convertidor 3L-FCC.

La Figura A.1 muestra la arquitectura general del sistema. El controlador regula el voltaje de salida y el voltaje del capacitor flotante a partir de dos lazos PI. Sus salidas alimentan posteriormente el modulador utilizado para generar las señales de conmutación del convertidor.

Las interfaces digitales imponen anchos de palabra definidos para las señales externas. Las entradas de los controladores corresponden a enteros sin signo de 16 bits, mientras que sus salidas utilizan enteros sin signo de 7 bits para representar los ciclos de trabajo enviados al modulador. Estos formatos se mantuvieron constantes durante la exploración, por lo que las modificaciones se limitaron a las operaciones internas del controlador.

A.2. Exploración de la representación en punto fijo

A partir de la referencia funcional se realizó una exploración de representaciones de punto fijo en MATLAB. Los controladores candidatos se evaluaron en lazo cerrado mediante Simulink y *PLECS Blockset*, comparando su respuesta con la implementación original en punto flotante. En esta exploración se mantuvo una distribución uniforme de bits para las variables internas y se modificó principalmente la cantidad de bits fraccionales.

De los formatos evaluados se seleccionaron tres configuraciones representativas:

- $Q_{8,24}$, como configuración de alta precisión.
- $Q_{8,19}$, como configuración intermedia.
- $Q_{8,14}$, como la menor precisión evaluada que mantuvo un comportamiento estable del sistema en lazo cerrado.

La Figura A.2 compara la respuesta obtenida con la referencia en punto flotante y con la configuración $Q_{8,14}$. A pesar de la reducción de precisión, el controlador conserva el seguimiento de la referencia. La exploración en software permitió así seleccionar configuraciones candidatas antes de trasladar el algoritmo al entorno HLS.

A.3. Traslado y verificación en HLS

Las tres configuraciones seleccionadas se trasladaron inicialmente a Vitis HLS mediante el tipo `ap_fixed`. La verificación comparó las salidas del modelo de alto nivel con las obtenidas a partir de la descripción sintetizada para un mismo conjunto de estímulos. Los resultados mostraron una correspondencia cercana entre ambas implementaciones.

En esta etapa se utilizaron trayectorias de entrada registradas previamente a partir del modelo de alto nivel. Debido a la acción integral del controlador PI, pequeñas diferencias numéricas entre implementaciones se acumularon durante esta comparación en lazo abierto. Esta limitación motivó complementar la verificación mediante una prueba posterior con la planta física.

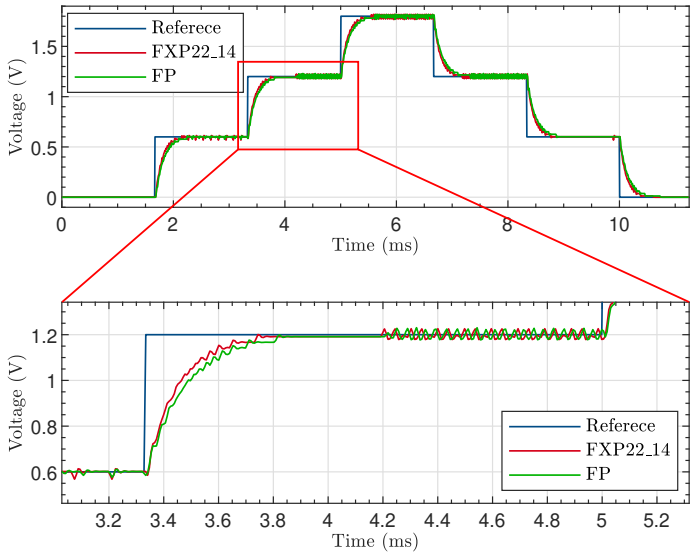


Figura A.2: Respuesta simulada en lazo cerrado del convertidor con los controladores en punto flotante y $Q_{8,14}$

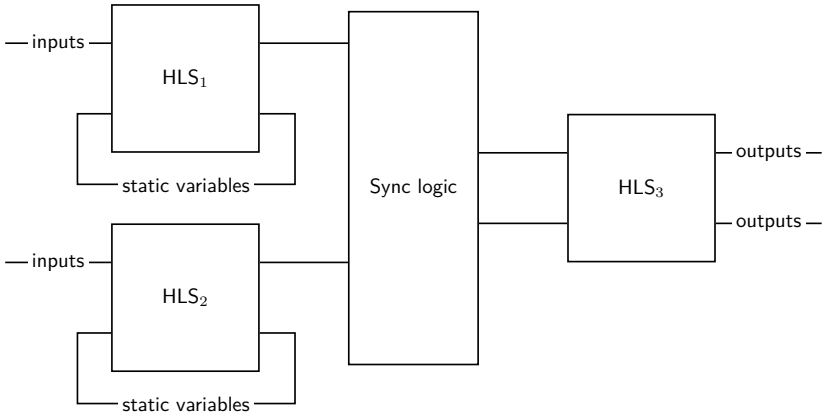


Figura A.3: Diagrama de regularización necesaria para implementar el controlador PI en Bambu HLS.

Como el objetivo final requería un flujo abierto orientado a ASIC, la configuración $Q_{8,14}$ se trasladó posteriormente a Bambu HLS. Para esta implementación se utilizó la biblioteca `ac_types`, cuyo tipo de punto fijo puede expresarse como

$$\texttt{ac_fixed}\langle \texttt{W,I,S,C,O} \rangle, \tag{A.1}$$

con parámetros análogos a los utilizados por `ap_fixed` para definir el ancho de palabra, los bits enteros y los modos de cuantización y desbordamiento.

Sin embargo, la similitud entre ambas interfaces no implica un comportamiento aritmético idéntico. En particular, las bibliotecas aplican reglas propias para determinar la precisión de los resultados intermedios. Durante la migración a Bambu fue necesario incorporar conversiones explícitas en algunas expresiones para controlar estos resultados y evitar truncamientos que producían diferencias respecto de la implementación de referencia.

También fue necesario adaptar el tratamiento de las variables con estado. Los dos controladores PI se separaron en bloques independientes y sus estados internos se conservaron mediante registros y conexiones de realimentación explícitas, como se muestra en la Figura A.3. De esta forma se mantuvo la acción integral entre ejecuciones sucesivas del controlador.

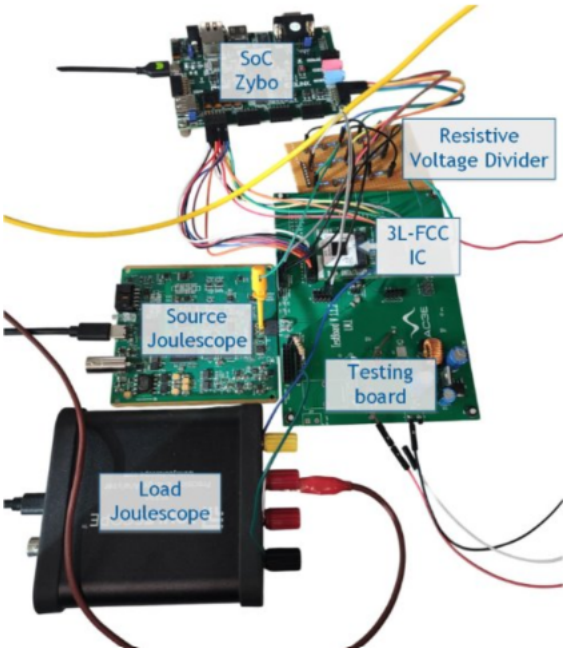


Figura A.4: Montaje utilizado para la validación experimental.

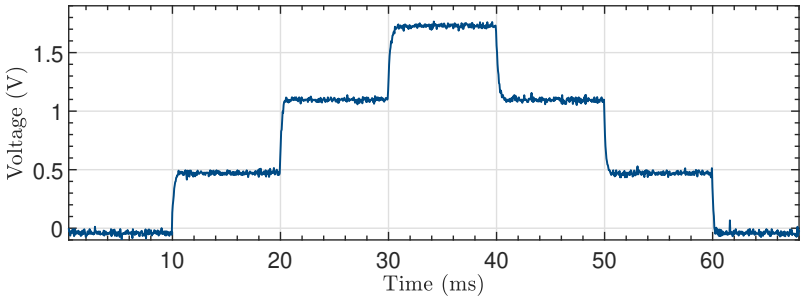


Figura A.5: Voltaje de salida medido para escalones de referencia de 0,6 V.

A.4. Validación experimental en FPGA

Antes de avanzar hacia el diseño físico del ASIC, la implementación generada con Bambu se prototipó en FPGA y se conectó directamente al convertidor 3L-FCC utilizado como planta. Esta prueba corresponde a una validación experimental en lazo cerrado con la planta física, por lo que va más allá de una verificación FPGA-in-the-Loop o Hardware-in-the-Loop basada en un modelo simulado de la planta.

La Figura A.4 muestra el montaje experimental y la Figura A.5 la respuesta medida para la configuración $Q_{8,14}$. El controlador implementado en FPGA regula el convertidor frente a escalones de referencia de 0,6 V dentro de un rango de operación de 0 a 1,8 V. Los resultados muestran que la representación seleccionada durante la exploración en software mantiene el comportamiento del lazo al ejecutar el controlador sobre hardware y conectarlo directamente con la planta real.

A.5. Extensión hacia una implementación ASIC

Después de verificar la implementación en FPGA, la descripción RTL generada con Bambu se procesó mediante LibreLane utilizando el PDK SKY130. El flujo incluyó síntesis lógica, *floorplanning*,

ubicación de celdas, síntesis del árbol de reloj, enrutamiento y verificación de reglas de diseño hasta obtener los archivos correspondientes al diseño físico.

Las descripciones obtenidas con Vitis HLS para las tres configuraciones también se procesaron mediante LibreLane con fines comparativos. Los resultados muestran una reducción progresiva del área al disminuir la longitud de palabra. En particular, el área de celdas reportada para las implementaciones generadas con Vitis disminuyó desde aproximadamente 151 365 μm^2 para $Q_{8,24}$ hasta 64 671 μm^2 para $Q_{8,14}$.

La implementación de $Q_{8,14}$ generada mediante Bambu también completó el flujo físico y alcanzó un área de 93 711 μm^2. Los valores de potencia fueron obtenidos mediante estimaciones sin vectores de actividad y, por lo tanto, se utilizaron principalmente para comparar las implementaciones bajo condiciones comunes. Los resultados permiten observar que las decisiones tomadas durante la selección de la representación numérica se reflejan posteriormente en las características de la arquitectura obtenida.

A.6. Relación con el flujo utilizado en esta tesis

Este caso complementario aplica la misma secuencia general utilizada en los capítulos anteriores: establecer una referencia funcional, explorar la representación de punto fijo a un nivel de abstracción alto, trasladar la representación seleccionada a un entorno HLS y verificar su comportamiento antes de avanzar hacia la implementación final.

En este caso, el flujo se aplicó a un controlador PI y se extendió posteriormente hacia un diseño orientado a ASIC. La utilización de dos bibliotecas de punto fijo también permitió observar que una representación nominalmente similar puede producir diferencias entre entornos debido al tratamiento de los resultados intermedios. Esta observación coincide con la necesidad, abordada en el Capítulo 4, de establecer equivalencias entre las reglas aritméticas de las herramientas utilizadas y verificar numéricamente el resultado del traslado.

Por otra parte, la etapa experimental complementa los casos de estudio del Capítulo 5. Mientras que en esta tesis la verificación final utiliza FPGA-in-the-Loop con una planta simulada, en esta aplicación el controlador implementado en FPGA se conectó directamente al convertidor físico. El caso permite así extender el mismo principio de validación progresiva desde la simulación de alto nivel hasta una implementación experimental y, posteriormente, hacia el diseño físico de un ASIC.

Apéndice B

PUBLICACIONES GENERADAS

A la fecha de entrega de este documento, se reportan las siguientes publicaciones derivadas de resultados parciales de la investigación desarrollada en el marco de este proyecto de tesis.

Conferencias Internacionales

1. F. Donoso, N. Salvador, C. A. Rojas, J. Marín and G. Carvajal, **"Design of a PI-Controller ASIC for a Multilevel Converter Using High-Level Synthesis and LibreLane,"** 2026 IEEE 17th Latin America Symposium on Circuits and System (LASCAS), Arequipa, Peru, 2026, pp. 1-5, doi: 10.1109/LASCAS67804.2026.11457108.

2. F. Donoso, N. Salvador, J. Marin, C. A. Rojas and G. Carvajal, **"A Case Study on the Migration of a High-Level PI Controller to ASIC-Compatible HDL Representation,"** 2025 IFIP/IEEE 33rd International Conference on Very Large Scale Integration (VLSI-SoC), Puerto Varas, Chile, 2025, pp. 1-2, doi: 10.1109/VLSI-SoC64688.2025.11421717.

3. N. Salvador, F. Donoso, J. Marín, V. Grimblatt and C. A. Rojas, **"Towards Full Integration of a Three-Level Flying Capacitor Converter Control in a Mixed-Signal ASIC,"** 2025 IFIP/IEEE 33rd International Conference on Very Large Scale Integration (VLSI-SoC), Puerto Varas, Chile, 2025, pp. 1-2, doi: 10.1109/VLSI-SoC64688.2025.11421731.