



GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Implementación de un control lineal cuadrático
en FPGA. Análisis comparativo de
rendimiento y recursos.

Autor: Alicia López García

Director: Fermín Zabalegui Sanz

Madrid, Julio 2026

Declaración de originalidad

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título **Implementación de un control lineal cuadrático en FPGA. Análisis comparativo de rendimiento y recursos.** en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2025/2026 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Uso de Inteligencia Artificial¹

Declaro bajo mi responsabilidad que (indicar la opción correcta):

- ☐ No he utilizado Inteligencia Artificial en la elaboración del presente documento.
- ☒ He utilizado Inteligencia Artificial en la elaboración del presente documento y/o del Anexo B siempre en las condiciones permitidas por la Universidad Pontificia Comillas, es decir, aplicando el Nivel 2 de la [Escala de Evaluación de Perkins et al. \(2024\)](#): *“La IA puede utilizarse para actividades previas a la tarea, como la lluvia de ideas, la descripción y la investigación inicial. Este nivel se centra en el uso de la IA para la planificación, las síntesis y la generación de ideas, pero las evaluaciones deben hacer hincapié en la capacidad de desarrollar y refinar estas ideas de forma independiente”*. En concreto, la Inteligencia Artificial ha sido empleada para:

Se ha utilizado inteligencia artificial generativa como apoyo para generar ideas iniciales, mejorar la redacción, sintetizar contenido y organizar la literatura consultada en la memoria, así como para la generación de testbenches y la depuración de código.

¹Esta declaración se refiere al uso de la Inteligencia Artificial generativa para realizar los documentos del Proyecto (Anexo B y Memoria). No aplica a Proyectos donde, por su naturaleza, deban emplear inteligencia artificial como parte de los mismos (aplicación de técnicas de aprendizaje automático, redes neuronales, análisis de datos...).


Fdo: Alicia López García
Fecha: 10/07/2026

Autorización para la entrega del Proyecto

El Director del Proyecto	El co-Director del Proyecto (si aplica)
ZABALEGUI SANZ 2026.07.13 FERMIN - 50202045Z 13:30:47 +02'00' (firma aquí)	 (firma aquí)
Fdo: Fermín Zabalegui Sanz	Fdo:
Fecha:	



GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Implementación de un control lineal cuadrático
en FPGA. Análisis comparativo de
rendimiento y recursos.

Autor: Alicia López García

Director: Fermín Zabalegui Sanz

Madrid, Julio 2026

Agradecimientos

Gracias a mi tutor Fermín por su paciencia a pesar de las dificultades técnicas que han surgido en el laboratorio.

A mis amigos que me han acompañado toda esta etapa.

A mi familia por su apoyo incondicional y a mis padres en especial por ser los ingenieros industriales que me inspiraron a estudiar esta carrera. Gracias por esta oportunidad.

IMPLEMENTACIÓN DE UN CONTROL LINEAL CUADRÁTICO EN FPGA. ANÁLISIS COMPARATIVO DE RENDIMIENTO Y RECURSOS.

Autor: López García, Alicia.

Director: Zabalegui Sanz, Fermín.

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

Resumen

Este trabajo aborda la implementación de un controlador lineal cuadrático en una FPGA Artix-7 (Basys3) (Digilent Inc., 2016), utilizando como planta un péndulo invertido construido en el laboratorio (Rodríguez Criado et al., 2025). Se desarrolla el modelo físico del sistema, el diseño del controlador, y su codificación en VHDL con aritmética en punto fijo. El diseño se valida mediante simulación en MATLAB/Simulink, simulación funcional en Vivado y pruebas sobre la planta física, comparándose además con una implementación equivalente en microcontrolador. Los resultados muestran un consumo energético reducido y una latencia de lazo de 360 ns, despreciable frente al periodo de muestreo de 1 ms, confirmando la viabilidad de esta plataforma para control en tiempo real.

Palabras clave: Control LQR; Péndulo invertido sobre carro; FPGA; VHDL; Aritmética en punto fijo; Microcontrolador; Control digital en tiempo real.

Resumen ejecutivo

1 Introducción

Las técnicas de control óptimo, como el LQR, se han popularizado en aplicaciones como la robótica o los sistemas aeroespaciales. Tradicionalmente se implementan en microcontroladores cuya ejecución secuencial limita la frecuencia de actuación y la latencia del lazo (Lin & Morton, 2017). La FPGA es una alternativa atractiva por su procesamiento en paralelo y latencia determinista (Kilts, 2006). Los trabajos que comparan cualitativamente ambas plataformas siguen siendo escasos, algunos limitados a una única plataforma (Karhan et al., 2024) o sin verificar el diseño sobre una planta real (Zhang et al., 2017). Este es el hueco que se pretende cubrir, tomando como sistema de validación un péndulo invertido sobre carro, banco de pruebas por excelencia del LQR por su inestabilidad, no linealidad y subactuación.

2 Objetivos

El objetivo principal del trabajo es la implementación del LQR discreto sobre FPGA y la evaluación comparativa de su rendimiento y consumo con una implementación equivalente en microcontrolador.

3 Descripción del sistema

El sistema consiste en un carro de masa M que se desplaza sobre un carril lineal mediante un motor NEMA17 con driver DRV8825 y transmisión por correa GT2, con un péndulo de masa m articulado a un encoder rotativo en cuadratura de 600 P/R. Se trata de un sistema no lineal de cuarto orden cuyo modelo se obtiene del planteamiento de las ecuaciones de Euler-Lagrange; se linealiza en torno al punto de equilibrio inestable para obtener una representación en espacio de estados.

El controlador para el sistema continuo se diseña mediante la función `lqr` de MATLAB y se discretiza para un periodo de muestreo de 1 ms. Las ganancias y las operaciones para obtener la fuerza de control u en cada ciclo de muestreo se codifican en VHDL utilizando aritmética en punto fijo Q8.8, teniendo el sistema la entidad de nivel superior (*top level*) mostrada en la Figura 1.

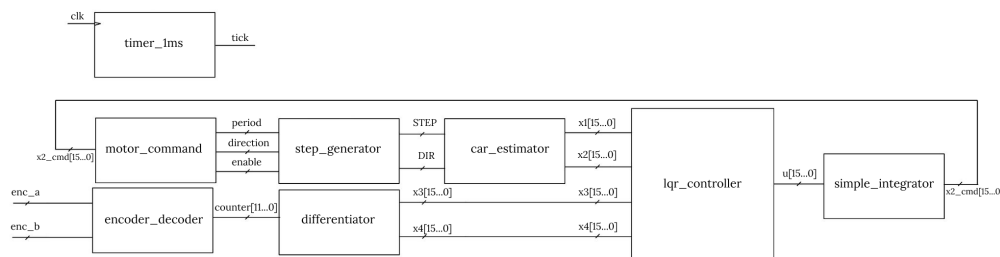


Figura 1: Entidad de nivel superior (*top level*) del sistema de control.

Como se puede ver en la Figura 1, las señales del encoder se procesan para estimar el estado del carro, que alimenta el cálculo de la ley de control LQR; la señal resultante, calculada como una fuerza, se integra para traducirla en el comando de velocidad que requiere el motor paso a paso real.

4 Resultados

El diseño se validó en tres etapas: simulación del lazo cerrado en MATLAB/Simulink, simulación funcional del código VHDL en Vivado y pruebas sobre la planta física. La implementación cumple los requisitos de temporización a 100 MHz con margen positivo ($WNS = +0,148$ ns) y presenta un uso reducido de recursos del Artix-7: 1 936 LUT (9,3 %), 2 310 registros (5,6 %) y 9 bloques DSP (10 %), sin emplear BRAM. La latencia de cómputo del lazo es de 360 ns, despreciable frente al periodo de muestreo de 1 ms, y el consumo medido sobre la placa es de 193 mW, frente a los 121 mW estimados por Vivado.

En la validación experimental, la secuencia automática de arranque (*homing*, centrado y espera de reposo) funciona correctamente y, tras la activación del controlador, el sistema estabiliza el péndulo en menos de un segundo, manteniendo el equilibrio durante breves segundos. Se identificaron tres limitaciones que impiden la estabilización sostenida: el desbordamiento del contador del encoder al aplicar el desplazamiento de referencia angular, la pérdida ocasional de pasos del motor a alta velocidad y una limitación de modelado, ya que el LQR se diseñó para una planta cuya entrada de control es una fuerza, mientras que el motor paso a paso regula velocidad, por lo que la planta controlada en hardware difiere ligeramente de la modelada. Adicionalmente, se verificó con éxito un experimento de *swing-up* por bombeo de energía.

La comparativa frente al microcontrolador (dsPIC33FJ32MC202) muestra que la latencia de la FPGA es unas 100 veces menor que la estimada para una implementación PID en coma flotante sin optimizar (40-50 μ s), pero solo de 3 a 8 veces menor que la de una implementación LQR equivalente en punto fijo (1-3 μ s). En consumo, ambas plataformas resultan del mismo orden de magnitud (~ 193 frente a ~ 178 mW estimados).

5 Conclusiones

Los resultados confirman la viabilidad de la FPGA como plataforma de control en tiempo real: el LQR discreto en punto fijo Q8.8 se ejecuta con latencia determinista muy inferior al periodo de muestreo y ocupa una fracción reducida del dispositivo, dejando margen para extensiones como el filtro de Kalman (LQG), que queda como línea de trabajo futuro. Para este problema, ambas plataformas cumplen el requisito de tiempo real con holgura, por lo que el cuello de botella del sistema es la dinámica del motor paso a paso y no la capacidad de cómputo.

La aportación principal del trabajo es una comparativa cuantitativa FPGA-microcontrolador

sobre una misma planta física, escasa en la literatura. De ella se concluye que la ventaja genuina de la FPGA no reside en la velocidad bruta ni en el consumo energético (comparables en condiciones equivalentes), sino en su determinismo temporal y su capacidad de ejecución en paralelo, a costa de una mayor complejidad de desarrollo frente a la programación en C del microcontrolador.

6 Referencias

- Digilent Inc. (2016). *Basys 3 FPGA Board Reference Manual*. Digilent Inc. Consultado el 13 de junio de 2026, desde <https://digilent.com/reference/programmable-logic/basys-3/reference-manual>
- Karhan, H. K., Yaren, T., & Kizir, S. (2024). Modelling and Control of a Reaction Wheel Pendulum Using a Linear Quadratic Regulator. *Sigma Journal of Engineering and Natural Sciences*, 42(6), 1907-1915. <https://doi.org/10.14744/sigma.2024.00157>
- Kilts, S. (2006). *Advanced FPGA Design: Architecture, Implementation, and Optimization*. John Wiley & Sons.
- Lin, Y., & Morton, T. D. (2017). A Microcontroller-based DSP Laboratory Curriculum. <https://doi.org/10.18260/1-2--27480>
- Rodríguez Criado, J., López de Azcona, J., & Millán Sánchez, T. (2025). Péndulo invertido [Proyecto de la asignatura Sistemas Electrónicos (Máster en Ingeniería Industrial) no publicado].
- Zhang, P., Mills, A., Zambreno, J., & Jones, P. H. (2017). The Design and Integration of a Software Configurable and Parallelized Coprocessor Architecture for LQR Control. *Journal of Parallel and Distributed Computing*, 106, 121-131. <https://doi.org/10.1016/j.jpdc.2017.01.028>

IMPLEMENTATION OF A LINEAR QUADRATIC REGULATOR ON FPGA. COMPARATIVE ANALYSIS OF PERFORMANCE AND RESOURCES.

Author: Alicia López García

Director: Fermín Zabalegui Sanz

Collaborating Entity: ICAI – Universidad Pontificia Comillas.

Abstract

This work addresses the implementation of a linear quadratic regulator on an Artix-7 FPGA (Basys3) (Digilent Inc., 2016), using as the plant an inverted pendulum built in the laboratory (Rodríguez Criado et al., 2025). The physical model of the system is developed, along with the controller design and its coding in VHDL using fixed-point arithmetic. The design is validated through simulation in MATLAB/Simulink, functional simulation in Vivado, and tests on the physical plant, also comparing it with an equivalent implementation on a microcontroller. The results show reduced energy consumption and a loop latency of 360 ns, negligible compared to the 1 ms sampling period, confirming the viability of this platform for real-time control.

Keywords: LQR control; Cart-pole inverted pendulum; FPGA; VHDL; Fixed-point arithmetic; Microcontroller; Real-time digital control.

Executive Summary

1 Introduction

Optimal control techniques, such as LQR, have become widespread in applications such as robotics or aerospace systems. They are traditionally implemented on microcontrollers, whose sequential execution limits the actuation frequency and loop latency (Lin & Morton, 2017). FPGAs are an attractive alternative due to their parallel processing and deterministic latency (Kilts, 2006). Works that qualitatively compare both platforms remain scarce, some limited to a single platform (Karhan et al., 2024) or without verifying the design on a real plant (Zhang et al., 2017). This is the gap this work aims to fill, using as its validation system a cart-pole inverted pendulum, the quintessential LQR benchmark given its instability, nonlinearity, and underactuation.

2 Objectives

The main objective of this work is the implementation of the discrete LQR on FPGA and the comparative evaluation of its performance and power consumption, also comparing it with an equivalent implementation on a microcontroller.

3 Description of the System

The system consists of a cart of mass M that moves along a linear rail via a NEMA17 motor with a DRV8825 driver and a GT2 belt transmission, with a pendulum of mass m attached to a 600 CPR quadrature rotary encoder. It is a fourth-order nonlinear system whose model is obtained from the Euler-Lagrange equations; it is linearized around the unstable equilibrium point to obtain a state-space representation.

The controller for the continuous system is designed using MATLAB's `lqr` function and discretized for a sampling period of 1 ms. The gains and the operations to obtain the control force u at each sampling cycle are coded in VHDL using fixed-point arithmetic, with the top-level entity of the system shown in Figure 1.

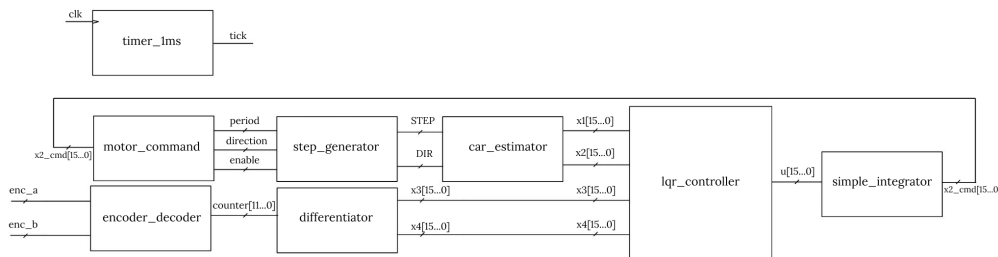


Figure 1: Top-level entity of the control system.

As shown in Figure 1, the encoder signals are processed to estimate the cart's state, which feeds the calculation of the LQR control law; the resulting signal, computed as a force, is integrated to translate it into the velocity command required by the physical stepper motor.

4 Results

The design was validated in three stages: closed-loop simulation in MATLAB/Simulink, functional simulation of the VHDL code in Vivado, and tests on the physical plant. The implementation meets timing requirements at 100 MHz with positive margin ($WNS = +0.148$ ns) and shows low resource usage on the Artix-7: 1,936 LUTs (9.3 %), 2,310 registers (5.6 %), and 9 DSP blocks (10 %), with no BRAM usage. The computational latency of the loop is 360 ns, negligible compared to the 1 ms sampling period, and the measured on-board power consumption is 193 mW, versus the 121 mW estimated by Vivado.

In the experimental validation, the automatic start-up sequence (homing, centering, and settling wait) works correctly, and once the controller is activated, the system stabilizes the pendulum in under one second, maintaining balance for a few seconds. Three limitations were identified that prevent sustained stabilization: overflow of the encoder counter when applying the angular reference offset, occasional motor step loss at high speed, and a modeling limitation, since the LQR was designed for a plant whose control input is a force, while the stepper motor regulates velocity, so the plant controlled in hardware differs slightly from the one modeled. Additionally, an energy-pumping swing-up experiment was successfully verified.

The comparison against the microcontroller (dsPIC33FJ32MC202) shows that the FPGA's latency is roughly 100 times lower than that estimated for an unoptimized floating-point PID implementation (40-50 μ s), but only 3 to 8 times lower than that of an equivalent fixed-point LQR implementation (1-3 μ s). In terms of power, both platforms are of the same order of magnitude (~ 193 vs. ~ 178 mW estimated).

5 Conclusions

The results confirm the viability of the FPGA as a real-time control platform: the discrete LQR in Q8.8 fixed-point arithmetic runs with deterministic latency far below the sampling period and occupies a small fraction of the device, leaving margin for extensions such as a Kalman filter (LQG), left as future work. For this problem, both platforms meet the real-time requirement with ample margin, so the system's bottleneck is the stepper motor's dynamics rather than either platform's computing capacity.

The work's main contribution is a quantitative FPGA-microcontroller comparison on the same physical plant, which remains scarce in the literature. It shows that the FPGA's genuine advantage

lies not in raw speed nor in energy consumption (comparable under equivalent conditions) but in its temporal determinism and parallel execution capability, at the cost of greater development complexity compared to programming the microcontroller in C.

6 References

- Digilent Inc. (2016). *Basys 3 fpga board reference manual*. Digilent Inc. Retrieved June 13, 2026, from <https://digilent.com/reference/programmable-logic/basys-3/reference-manual>
- Karhan, H. K., Yaren, T., & Kizir, S. (2024). Modelling and control of a reaction wheel pendulum using a linear quadratic regulator. *Sigma Journal of Engineering and Natural Sciences*, 42(6), 1907–1915. <https://doi.org/10.14744/sigma.2024.00157>
- Kilts, S. (2006). *Advanced fpga design: Architecture, implementation, and optimization*. John Wiley & Sons.
- Lin, Y., & Morton, T. D. (2017). A microcontroller-based dsp laboratory curriculum. <https://doi.org/10.18260/1-2--27480>
- Rodríguez Criado, J., López de Azcona, J., & Millán Sánchez, T. (2025). Péndulo invertido [Proyecto de la asignatura Sistemas Electrónicos (Máster en Ingeniería Industrial) no publicado].
- Zhang, P., Mills, A., Zambreno, J., & Jones, P. H. (2017). The Design and Integration of a Software Configurable and Parallelized Coprocessor Architecture for LQR Control. *Journal of Parallel and Distributed Computing*, 106, 121–131. <https://doi.org/10.1016/j.jpdc.2017.01.028>

Índice

Capítulo 1	Introducción	4
1.1	Motivación	6
1.2	Objetivos	8
1.3	Estructura del trabajo	10
Capítulo 2	Estado del arte	11
2.1	Control óptimo y regulador LQR	11
2.1.1	Aeronáutica y aeroespacial	11
2.1.2	Robótica y sistemas autónomos	12
2.1.3	Automoción	12
2.2	El péndulo invertido	13
2.2.1	Robots bípedos	13
2.2.2	Vehículos autoequilibrados (<i>Segway</i>)	13
2.2.3	Despegue de vehículos aeroespaciales	14
2.3	Implementaciones en microcontroladores	14
2.4	Implementaciones en FPGA	15
2.5	Limitaciones actuales y hueco en la literatura	17
Capítulo 3	Metodología	19
3.1	Planteamiento del problema y requisitos del sistema	19
3.1.1	Objetivo de control	19
3.1.2	Requisitos funcionales	20
3.1.3	Requisitos de implementación FPGA	20
3.1.4	Métricas de evaluación	21
3.2	Descripción del sistema físico	21
3.2.1	Estructura mecánica	22
3.2.2	Actuador: motor paso a paso NEMA17 con driver DRV8825	23
3.2.3	Sensor: encoder óptico de cuadratura	23
3.2.4	Plataforma FPGA: Basys3 (Artix-7)	24
3.2.5	Interfaz eléctrica y conexionado Pmod	25
3.3	Diseño del controlador LQR	27
3.3.1	Fundamentos teóricos del regulador LQR	27

3.3.2	Modelo del péndulo invertido sobre carro	30
3.3.3	Elección de las matrices de peso Q y R	34
3.3.4	Ganancias del LQR continuo	35
3.3.5	Verificación de estabilidad en tiempo continuo	35
3.3.6	Simulación en tiempo continuo	35
3.4	Discretización del controlador	36
3.4.1	Método de retención de orden cero (ZOH)	36
3.4.2	Elección del periodo de muestreo	36
3.4.3	Ganancias del LQR discreto	37
3.4.4	Verificación de estabilidad discreta	37
3.4.5	Simulación en tiempo discreto	38
3.5	Aritmética en punto fijo	38
3.5.1	Representación en punto fijo: formato $Qm.n$	39
3.5.2	Formatos elegidos y análisis de rango de variables	40
3.5.3	Error de cuantización de las ganancias y constantes	40
3.5.4	Análisis de los productos intermedios	41
3.6	Arquitectura hardware en FPGA	43
3.6.1	Visión general del sistema	43
3.6.2	Máquina de estados del sistema	45
3.7	Implementación en VHDL	47
3.7.1	Módulo <code>timer_1ms</code>	47
3.7.2	Módulo <code>encoder_decoder</code>	47
3.7.3	Módulo <code>stillness_detector</code>	47
3.7.4	Módulo <code>differentiator</code>	48
3.7.5	Módulo <code>simple_integrator</code>	48
3.7.6	Módulo <code>mult_q8_8</code>	48
3.7.7	Módulo <code>sumador_sat</code>	49
3.7.8	Módulo <code>lqr_controller</code>	49
3.7.9	Módulo <code>motor_command</code>	49
3.7.10	Módulo <code>step_generator</code>	50
3.7.11	Módulo <code>cart_estimator</code>	51
3.7.12	Módulo <code>lqr_system_fsm</code>	51
3.7.13	Módulo <code>lqr_system_top</code>	52

3.8	Implementación en microcontrolador	52
3.8.1	Lectura del encoder	53
3.8.2	Temporizador del lazo de control	53
3.8.3	Generación de pulsos STEP/DIR	53
3.8.4	Cálculo de la ley de control	53
Capítulo 4	Resultados	54
4.1	Validación mediante simulación	54
4.1.1	Simulación MATLAB del sistema en lazo cerrado	54
4.1.2	Simulación funcional VHDL (testbench)	59
4.1.3	<code>tb_lqr_system_fsm</code>	59
4.1.4	<code>lqr_controller</code>	60
4.1.5	<code>motor_command</code>	61
4.2	Resultados de implementación en FPGA	63
4.2.1	Síntesis y uso de recursos (Vivado)	63
4.2.2	Análisis energético	64
4.2.3	Latencia del lazo de control	65
4.2.4	Validación experimental en hardware	66
4.3	Resultados de implementación en microcontrolador	67
4.3.1	Uso de recursos	67
4.3.2	Análisis energético	68
4.3.3	Latencia del lazo de control	68
4.3.4	Validación experimental en hardware	69
4.4	Comparativa FPGA vs. microcontrolador	69
4.4.1	Latencia y frecuencia de actualización	69
4.4.2	Recursos	69
4.4.3	Complejidad de desarrollo	69
4.4.4	Análisis comparativo global	70
4.5	Proceso de depuración y principales dificultades	70
4.5.1	Corrección del pipeline en <code>sumador_sat</code>	71
4.5.2	Sustitución de <code>double_integrator</code> por <code>cart_estimator</code> y <code>simple_integrator</code>	71
4.5.3	Limitación de frecuencia del motor	71
4.5.4	Problema de arranque del motor	72
4.5.5	Ajuste de parámetros rampa de aceleración motor	73

4.5.6	Experimento de control de subida (<i>swing-up</i>)	74
Capítulo 5	Conclusiones y Trabajo Futuro	76
5.1	Conclusiones	76
5.1.1	Cumplimiento de objetivos e implementación en FPGA	76
5.1.2	Conclusiones sobre la comparativa FPGA vs. microcontrolador	76
5.1.3	Lecciones aprendidas del proceso de desarrollo	77
5.2	Coste normalizado	78
5.3	Trabajo Futuro	79
5.3.1	Extensión a control LQG (filtro de Kalman)	79
5.3.2	Expansión del contador del encoder a 16 bits	79
5.3.3	Optimización de las matrices Q y R y simulación del sistema no lineal	79
5.3.4	Portabilidad a FPGA de menor coste	80
Bibliografía		81
Anexo A	Alineación con los ODS	86
Anexo B	Código MATLAB	88
Anexo C	Simulaciones Simulink	91
Anexo D	Código VHDL	93
Anexo E	Simulaciones Vivado	111

Índice de figuras

1	Reloj de agua de Ctesibio, s. III a.C. (Ozer, 2011).	4
2	Regulador centrífugo de Watt, 1788 (Bissell, 2009).	4
3	Comparación de la ejecución secuencial en microcontrolador frente a la ejecución paralela en FPGA para un regulador LQR de tres estados.	7
4	Posición objetivo del control: péndulo en equilibrio invertido ($\theta = 0$).	20
5	Motor paso a paso NEMA17.	23
6	Driver DRV8825.	23
7	Encoder óptico de cuadratura.	24
8	Placa de desarrollo Basys3.	24
9	Diagrama de conexonado del sistema.	26
10	Sistema péndulo invertido sobre carro completamente montado en el laboratorio.	27
11	Circuito del sistema completamente montado en el laboratorio.	27
12	Esquema del sistema péndulo invertido sobre carro (University of Michigan, 2024).	30
13	Fuerzas actuando sobre el carro y el péndulo (University of Michigan, 2024).	31
14	Diagrama de bloques del sistema completo (<code>lqr_system_top</code>).	43
15	Diagrama de bloques del controlador LQR (<code>lqr_controller</code>).	44
16	Máquina de estados del sistema (<code>lqr_system_fsm</code>).	45
17	Máquina de estados de la rampa de velocidad (<code>motor_command</code>).	50
18	Efecto de aumentar el peso de cada variable de estado ($\theta_0 = 15^\circ$).	55
19	Barrido de pesos Q con prioridad sobre x_3 ($\theta_0 = 15^\circ$).	56
20	Barrido de ángulos iniciales con Q_1 .	57
21	Respuesta continua del sistema en lazo cerrado ($\theta_0 = 10^\circ$).	58
22	Respuesta discreta del sistema en lazo cerrado ($\theta_0 = 10^\circ$).	58
23	Fuerza de mando u sistema continuo ($\theta_0 = 10^\circ$).	58
24	Fuerza de mando u sistema discreto ($\theta_0 = 10^\circ$).	58
25	Simulación funcional del módulo <code>lqr_system_fsm</code> .	60
26	Simulación funcional del módulo <code>lqr_controller</code> .	61
27	Simulación de <code>motor_command</code> : saturación de <code>period</code> a <code>PER_MIN= 6250</code> con $x_{2_cmd} = 205$ y reinicio de rampa al cambiar la referencia de velocidad.	62

28	Simulación de <code>motor_command</code> : convergencia de <code>period</code> al cociente $T = 25100$ ciclos con $x_2_cmd = 51$, transición por <code>LQR_IDLE</code> y arranque de nueva rampa con $x_2_cmd = 128$	62
29	Rampa teórica de velocidad lineal del carro para las combinaciones exitosas del ensayo (microstepping 1/8).	74
30	Icono del ODS 9 (Naciones Unidas, s.f.).	86
31	Icono del ODS 7 (Naciones Unidas, s.f.).	87
32	Icono del ODS 8 (Naciones Unidas, s.f.).	87
33	Icono del ODS 4 (Naciones Unidas, s.f.).	87
34	Esquema de Simulink sistema continuo.	91
35	Esquema de Simulink sistema discreto.	91
36	Simulación sistema continuo I.	91
37	Simulación sistema discreto I.	91
38	Simulación sistema continuo II.	92
39	Simulación sistema discreto II.	92
40	Simulación sistema continuo III.	92
41	Simulación sistema discreto III.	92
42	Simulación funcional del módulo <code>encoder_decoder</code>	111
43	Simulación funcional del módulo <code>stillness_detector</code>	112
44	Simulación funcional del módulo <code>differentiator</code>	112
45	Simulación funcional del módulo <code>cart_estimator I</code>	113
46	Simulación funcional del módulo <code>cart_estimator II</code>	113
47	Simulación funcional del módulo <code>simple_integrator</code>	113
48	Simulación funcional del módulo <code>step_generator</code>	114

Índice de tablas

1	Comparativa entre microcontrolador y FPGA para implementación de controladores LQR.	17
2	Parámetros físicos del sistema (confirmados experimentalmente).	22
3	Recursos disponibles en la Artix-7 XC7A35T (Digilent Inc., 2016).	24
4	Asignación de pines de la Basys3 (XC7A35T-1CPG236C).	25
5	Características del formato Q8.8 signed de 16 bits.	40
6	Rango físico de los estados para verificar su representabilidad en formato Q8.8.	40
7	Errores de cuantización de las ganancias LQR en formato Q8.8.	41
8	Otras constantes en formato Q8.8 y su error de cuantización.	41
9	Rango máximo de los productos intermedios LQR para verificar su representabilidad en formato Q8.8.	42
10	Características del formato Q0.16 de 16 bits y error relativo tras la conversión.	42
11	Constantes sub-unitarias en formato Q0.16 y su error de cuantización.	43
12	Resultados del barrido de Q ($\theta_0 = 15^\circ$).	56
13	Rango de velocidad del carro para las configuraciones finalistas ($\theta_0 = 15^\circ$).	57
14	Utilización de recursos tras síntesis (<code>synth_1</code>), sin <code>div_gen_0</code>	63
15	Utilización de recursos del IP Core <code>div_gen_0</code> (síntesis independiente).	63
16	Utilización de recursos de la Basys3 (Artix-7 XC7A35T).	64
17	Consumo estimado de la implementación (Power Report, <code>impl_1</code>).	65
18	Desglose de latencia del lazo de control.	66
19	Resumen del análisis de temporización (<code>impl_1</code>).	66
20	Comparativa de recursos entre FPGA y microcontrolador.	69
21	Comparativa global entre FPGA y microcontrolador.	70
22	Ensayo de ajuste de la rampa de aceleración <code>motor_command</code>	73
23	Ensayo del experimento de <i>swing-up</i> por bombeo de energía.	75
24	Componentes principales.	78
25	Mano de obra.	78
26	Presupuesto general.	78

Capítulo 1 Introducción

A lo largo de la historia, el ser humano siempre ha buscado formas de regular y controlar sistemas físicos. Los primeros intentos de control se remontan a la Edad Antigua, con ejemplos como el reloj de agua de Ctesibio en el siglo III a.C., que con un mecanismo de retroalimentación lograba mantener el nivel de agua constante para obtener un flujo de salida constante (Bissell, 2009). Sin embargo, fue durante la Revolución Industrial cuando el control automático empezó a tomar forma con la invención del regulador centrífugo de James Watt en 1788, que controlaba la velocidad de las máquinas de vapor, manteniéndolas constantes a pesar de las variaciones en la carga (Suzuki, 2013). La consolidación del control automático de sistemas como disciplina formal llegó en el siglo XX como respuesta a la demanda de la industria, la aviación y más tarde la carrera espacial (Åström & Kumar, 2014). En las últimas décadas, el desarrollo de sistemas de control avanzados ha cobrado una relevancia aún mayor en ámbitos como la robótica, la automoción o los sistemas aeroespaciales (Burns, 2001).

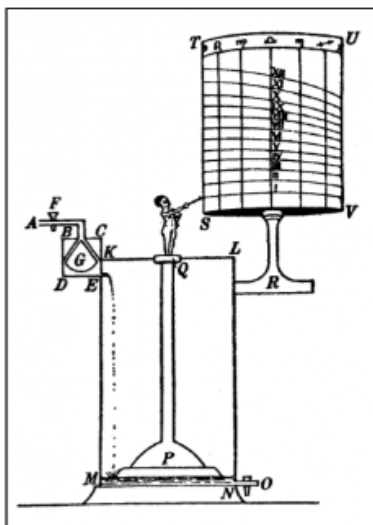


Figura 1: Reloj de agua de Ctesibio, s. III a.C. (Ozer, 2011).

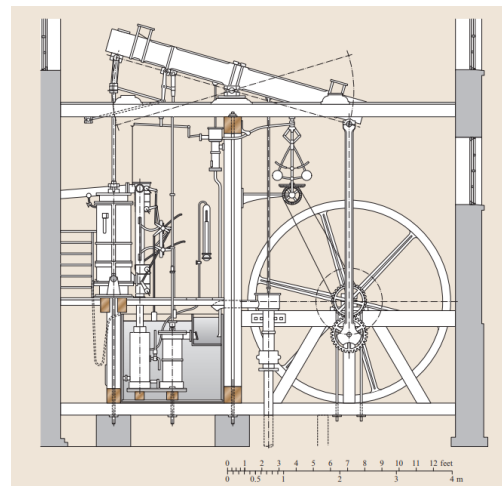


Figura 2: Regulador centrífugo de Watt, 1788 (Bissell, 2009).

El control automático de sistemas dinámicos cumple el objetivo de regular el comportamiento de un sistema físico manteniendo las variables del mismo en los valores deseados, rechazando perturbaciones externas y corrigiendo las incertidumbres del propio sistema (Ogata, 2010). En estas aplicaciones, los sistemas pueden ser rápidos, no lineales y operar en entornos

cambiantes, lo que hace que el diseño de controladores robustos y eficientes sea un desafío constante. Los enfoques clásicos de control, como el control proporcional-integral-derivativo (PID) o el diseño por lugar de raíces han sido ampliamente utilizados debido a su simplicidad y eficacia en una amplia gama de aplicaciones (Nise, 2020). Sin embargo, estos enfoques pueden no ser suficientes para sistemas complejos que requieren un rendimiento óptimo y una respuesta rápida ante perturbaciones (Saraf et al., 2020).

En este contexto, el control óptimo ofrece un escenario más sistemático para abordar estos desafíos, permitiendo diseñar el controlador desde un criterio de rendimiento concreto que puede formalizar el compromiso entre la rapidez de la respuesta y el esfuerzo realizado por el actuador del sistema. Entre las técnicas de control óptimo, el regulador lineal cuadrático (Linear Quadratic Regulator, LQR) se ha consolidado como una de las más usadas en ingeniería dada su capacidad de diseño y simplicidad a la hora de proporcionar soluciones con garantías de estabilidad y un equilibrio directo entre el error del sistema y el esfuerzo de control (Alavinasab et al., 2006).

Históricamente, los primeros controladores automáticos se implementaron mediante circuitos analógicos, amplificadores operacionales y redes RC para lograr la respuesta deseada mediante operaciones de control en tiempo continuo. A pesar de ser rápidos, eran diseños rígidos: cualquier pequeña modificación suponía un rediseño de hardware (Bennett, 1993). En los 70 la llegada de los microprocesadores y posteriormente de los procesadores digitales de señal (DSP) supuso un cambio de paradigma, permitiendo la implementación en software, lo cual dio gran flexibilidad y facilidad de modificación (Isermann, 1989). Los microcontroladores se convirtieron de esta manera en la plataforma más usada para control digital (Mohd Isa, 2008). No obstante, su arquitectura secuencial impone una limitación fundamental: el procesador ejecuta las instrucciones una a una, lo que puede ser un cuello de botella en aplicaciones de alta velocidad y baja latencia. Como alternativa, las FPGA (*Field Programmable Gate Arrays*), surgidas a mediados de los años 80, han evolucionado hasta convertirse en una plataforma viable para implementar algoritmos de control gracias a su capacidad de procesamiento paralelo, latencia determinista y flexibilidad de reconfiguración (Zhao, 2024).

En esta intersección entre el control óptimo y la implementación hardware es donde se sitúa este Trabajo Fin de Grado, cuyo objetivo es implementar un controlador LQR discreto en una FPGA y realizar un análisis comparativo exhaustivo de rendimiento (latencia, frecuencia de actualización), consumo de recursos hardware y eficiencia energética con el fin de compararla también con implementación equivalente en microcontrolador. Como sistema

de validación se emplea un péndulo invertido sobre carro, uno de los bancos de pruebas más populares en la literatura de control por combinar inestabilidad intrínseca, dinámica no lineal y subactuación en un sistema de baja dimensión, cuyas propiedades lo hacen representativo de problemas reales como el control de cohetes o robots bípedos. Adicionalmente, el trabajo explora la viabilidad de extender el diseño hacia el control Lineal Cuadrático Gaussiano (LQG), que incorporaría un filtro de Kalman para estimar los estados del sistema cuando las mediciones de los sensores estén contaminadas con ruido.

1.1. Motivación

En el mundo actual áreas como la automatización industrial, la robótica o los sistemas aeroespaciales imponen requisitos de control cada vez más exigentes: tiempos de respuesta reducidos, alta precisión, y comportamiento determinista ante perturbaciones (Brogårdh, 2009). En este contexto, la elección de la plataforma de implementación del controlador no es una decisión trivial, sino un factor que afecta directamente al rendimiento del sistema completo.

Durante décadas, los microcontroladores y procesadores digitales de señal (DSP) han sido la plataforma principal de implementación de algoritmos de control digital, gracias a su bajo coste, facilidad de programación y amplia disponibilidad (Lin & Morton, 2017). Sin embargo, su arquitectura secuencial impone una gran limitación: el procesador ejecuta las instrucciones una a una, de modo que el tiempo de cómputo del algoritmo de control depende directamente del número de operaciones realizadas. Técnicas como el uso de periféricos DMA o interrupciones prioritarias permiten aliviar parcialmente este problema, pero no eliminan el cuello de botella de fondo.

Las FPGA presentan una alternativa radicalmente distinta. Al implementar el algoritmo de control como un circuito hardware dedicado, con operaciones ejecutadas en paralelo y en pipeline, el retardo queda determinado únicamente por la profundidad del camino crítico, independientemente del número de operaciones concurrentes. Esto permite alcanzar latencias del orden de decenas de nanosegundos con carácter estrictamente determinista, una propiedad muy importante para el análisis formal de estabilidad del sistema en lazo cerrado (Kilts, 2007).

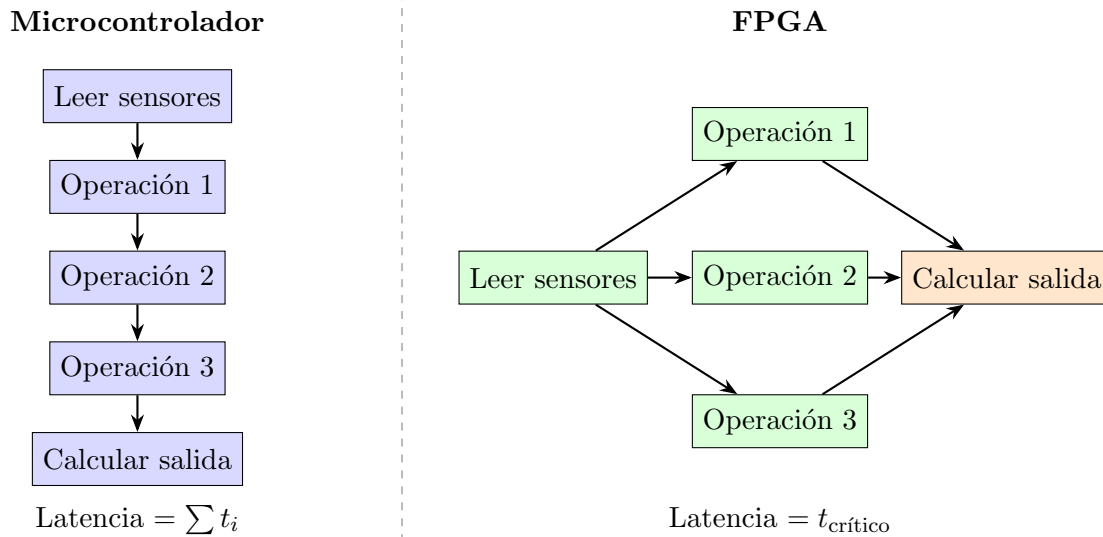


Figura 3: Comparación de la ejecución secuencial en microcontrolador frente a la ejecución paralela en FPGA para un regulador LQR de tres estados.

Esta ventaja resulta especialmente relevante en aplicaciones como la electrónica de potencia, donde aumentar la frecuencia de conmutación de los semiconductores permite reducir el tamaño de los componentes pasivos, pero impone mayores exigencias dinámicas al sistema de control. En este contexto, la implementación en FPGA se presenta como una opción viable no solo para los convertidores actuales, sino también para futuras aplicaciones con semiconductores de banda ancha (SiC, GaN), donde las frecuencias de conmutación pueden aumentar de forma significativa (Kahl & Dieckerhoff, 2017). Todo ello con carácter estrictamente determinista, una propiedad especialmente valiosa para el análisis formal de estabilidad del sistema en lazo cerrado.

Sin embargo, esta mejora de rendimiento tiene un coste: mayor complejidad de diseño, necesidad de herramientas de síntesis especializadas y un uso de recursos hardware que debe gestionarse cuidadosamente. Precisamente por ello, cuantificar de forma rigurosa la ganancia en latencia y frecuencia de actualización, y determinar a costa de cuántos recursos y consumo de potencia se obtiene, es una contribución de valor para la comunidad de control embebido. La mayor parte de los estudios existentes se limitan a validar el controlador en simulación o a reportar métricas de recursos sin contrastarlas con una implementación equivalente en microcontrolador sobre un sistema físico real, ese es el vacío que este trabajo pretende cubrir.

Adicionalmente, se plantea explorar la viabilidad de extender el diseño hacia el control

Lineal Cuadrático Gaussiano (LQG), lo que añadiría realismo al proyecto. En la práctica, las mediciones de los sensores están siempre contaminadas por ruido, lo que puede degradar seriamente el rendimiento de un regulador LQR puro que asume conocimiento exacto del estado. La incorporación de un filtro de Kalman como estimador de estado permitiría abordar este problema de forma óptima; su implementación completa, sin embargo, queda fuera del alcance de este trabajo y se plantea como línea de trabajo futuro.

1.2. Objetivos

El objetivo principal de este Trabajo Fin de Grado es la implementación de un controlador LQR discreto sobre una FPGA y la evaluación comparativa de su rendimiento frente a una implementación equivalente en un microcontrolador. Para alcanzar dicho objetivo, se han definido los siguientes objetivos específicos:

- **Diseñar y simular el controlador LQR para la planta seleccionada**

Se modelará matemáticamente la planta del sistema que se quiere controlar, planteando las ecuaciones de movimiento del péndulo invertido y la linealización de las mismas entorno al punto de equilibrio inestable. A partir de este modelo, se calcularán las ganancias óptimas del controlador LQR discreto. Se validará entonces el diseño mediante simulaciones, comprobando que cumple las expectativas de estabilidad y rendimiento antes de pasar a la implementación hardware.

- **Implementar el controlador LQR discreto en FPGA**

Tras esta validación en simulación se implementará en FPGA el regulador en su versión discreta, pasando del diseño al hardware. Se tendrán en cuenta las capacidades de procesamiento paralelo de la FPGA para garantizar un control en tiempo real con baja latencia y comportamiento determinista.

- **Validar el controlador en sistemas físicos representativos**

Se probará el control implementado en el sistema dinámico real de péndulo invertido. Esto permitirá comprobar el comportamiento real del controlador incluyendo perturbaciones, no linealidades y variaciones del sistema no contempladas en el modelo ideal linealizado.

- **Realizar un análisis de rendimiento temporal y consumo de recursos hardware en la implementación FPGA**

Se evaluará el rendimiento temporal del controlador en FPGA, midiendo la latencia de ejecución y la frecuencia de actualización de las señales de control. Este análisis permitirá comparar la implementación en FPGA con soluciones basadas en microcontroladores, así como verificar el cumplimiento de los requisitos del sistema. Además, se analizará el uso de recursos en la FPGA, como lógica programable, bloques de memoria y multiplicadores, con el fin de determinar la eficiencia del diseño. Este objetivo permitirá identificar posibles limitaciones del hardware y plantear mejoras futuras.

- **Análisis de la eficiencia energética**

Se analizará el consumo de potencia eléctrica para determinar la eficiencia del control implementado en la FPGA. Con esta información se podrá determinar si verdaderamente la implementación en FPGA frente al microcontrolador supone una optimización energética.

- **Comparar el rendimiento de la implementación en FPGA frente a microcontrolador**

A partir de las métricas de rendimiento temporal, recursos hardware y consumo de potencia obtenidas, se realizará un análisis comparativo sistemático entre la implementación en FPGA y una equivalente en microcontrolador, identificando en qué escenarios cada plataforma presenta ventajas claras y cuáles son sus limitaciones respectivas.

- **Explorar la viabilidad de extender el diseño hacia el control Lineal Cuadrático Gaussiano (LQG)**

Se analizará la viabilidad de incorporar un filtro de Kalman para estimar los estados del sistema cuando las mediciones de los sensores estén contaminadas con ruido, evaluando los recursos hardware y la complejidad adicional que requeriría dicha extensión, sin llegar a implementarla dentro del alcance de este trabajo.

1.3. Estructura del trabajo

Este trabajo está organizado en cinco capítulos, estructurados de la siguiente manera:

- **Capítulo 1. Introducción:** presenta el contexto y la motivación del trabajo, define los objetivos y describe la alineación del proyecto con los Objetivos de Desarrollo Sostenible.
- **Capítulo 2. Estado del arte:** revisa el control óptimo y el regulador LQR, el péndulo invertido como banco de pruebas estándar, y las implementaciones existentes tanto en microcontroladores como en FPGA, identificando las limitaciones de los enfoques actuales.
- **Capítulo 3. Metodología:** describe el modelado matemático del sistema físico, el diseño del regulador LQR discreto, su adaptación a aritmética de punto fijo y la implementación completa sobre la placa Basys 3 (Artix-7).
- **Capítulo 4. Resultados:** presenta los resultados obtenidos, incluyendo el diseño y validación del controlador en simulación tanto en MATLAB como en Simulink, así como los resultados de la simulación del código implementado en VHDL, y su posterior rendimiento en el sistema físico analizando las métricas de latencia, frecuencia de actualización, consumo de recursos hardware y eficiencia energética y comparando los resultados con una implementación equivalente en microcontrolador, revisa por último, los desafíos presentados a lo largo de la implementación y las soluciones elegidas.
- **Capítulo 5. Conclusiones y trabajo futuro:** recoge las conclusiones del trabajo, evalúa el grado de cumplimiento de los objetivos y propone líneas de desarrollo futuro.

Capítulo 2 Estado del arte

2.1. Control óptimo y regulador LQR

El control óptimo surge como respuesta a la necesidad de diseñar controles que no solo garanticen la estabilidad de un sistema, sino que también optimicen su respuesta respecto a un criterio de rendimiento cuantificable de su comportamiento.

Los fundamentos del control óptimo fueron establecidos en la década de 1950 por el ruso Lev Pontryagin, quien introdujo el Principio del Máximo, una condición necesaria de optimalidad que permite determinar la ley de control que minimiza un funcional de coste incluso cuando la señal de control está acotada, a diferencia de los métodos variacionales clásicos (Gamkrelidze, 2006). Más tarde en los años 60, Richard Bellman desarrolló la teoría de la programación dinámica, que reformula el problema de optimización mediante el principio de optimalidad, descomponiendo el problema óptimo global en una secuencia de decisiones óptimas locales resueltas recursivamente hacia atrás en el tiempo (Bellman, 1966). En esta misma década, Rudolf Kalman formuló el regulador lineal cuadrático (LQR) (Kálmán, 1960), que consiste en minimizar una función de coste cuadrático J de la forma

$$J = \int_0^\infty (\mathbf{x}^\top Q \mathbf{x} + \mathbf{u}^\top R \mathbf{u}) dt, \quad (2.1)$$

donde las matrices Q y R penalizan, respectivamente, la desviación de los estados $\mathbf{x}$ respecto al equilibrio y el esfuerzo de control $\mathbf{u}$. La minimización de J se reduce a resolver la ecuación algebraica de Riccati, cuya solución proporciona el vector de ganancias K con el que se calcula la ley de control por retroalimentación de estados, $\mathbf{u} = -K\mathbf{x}$. Este regulador se convirtió en un pilar fundamental del control óptimo.

Desde entonces, las técnicas de control óptimo y concretamente el regulador LQR han sido ampliamente aplicadas en una variedad de campos. Algunos ejemplos de aplicaciones se indican a continuación:

2.1.1. Aeronáutica y aeroespacial

Se ha utilizado para el control de actitud de satélites (Yang, 2014). Se partió de un modelo de cuaterniones controlable, sobre el que se diseñó un LQR basado en la linealización del sistema. Los resultados demostraron robustez frente a incertidumbres y perturbaciones

externas para el sistema no lineal global, los autores atribuyeron esto a que el diseño LQR era equivalente a una asignación robusta de polos haciéndolo poco sensible a posibles errores de modelado.

También se ha aplicado a problemas de seguimiento completo de estado en aeronaves (*full state tracking*) (Dul et al., 2020). Consiste en linealizar de forma recursiva el sistema en torno al estado actual mediante jacobianos y aplicar el LQR al error de seguimiento en lugar de al estado. Los resultados evidenciaron que la estabilidad se mantenía aunque con una leve pérdida de precisión ante mediciones ruidosas y perturbaciones atmosféricas como turbulencias o viento constante. Esto demostró que el LQR es robusto para sistemas de alta dimensión en los que los métodos clásicos como el control PID resultan poco prácticos.

2.1.2. Robótica y sistemas autónomos

En este ámbito, el regulador LQR se ha aplicado a cuadricópteros (*quadrotors*), comparando distintas formulaciones de la ecuación de Riccati (en régimen permanente y en su versión variable en el tiempo), evaluando su rendimiento en simulación y en la plataforma real (Bouabdallah, 2007). Las respuestas en simulación fueron satisfactorias, sin embargo, el desempeño en hardware mostró ciertos problemas: en primer lugar grandes variaciones de comportamiento con leves modificaciones en Q y R . Por otro lado, errores de estado estacionario por problemas de asimetrías físicas sin modelar, constatando la distancia que existe entre el diseño teórico y la validación experimental.

2.1.3. Automoción

En el área de la automoción se encuentran numerosas aplicaciones del control óptimo, un ejemplo es el control lateral (*lane keeping*). Se han usado modelos simplificados de coche (modelo de bicicleta linealizado), controlando el error respecto al carril en vez de la orientación absoluta del vehículo (Snider, 2009). El estudio compara el regulador lineal cuadrático frente a métodos geométricos y cinemáticos más simples, concluyendo en que el LQR básico no supone una gran mejora frente a estos métodos ya que el modelo linealizado no incluye la dinámica de la trayectoria. Se incorporó un término *feedforward* dependiente de la curvatura de la trayectoria que eliminó el error en régimen permanente en curvas a velocidades moderadas, sin embargo, introdujo sobreoscilaciones en cambios rápidos de curvatura. Este último problema fue resuelto añadiendo al control una previsión (*preview*) de la trayectoria futura.

2.2. El péndulo invertido

El péndulo invertido sobre un carro es uno de los sistemas de referencia más utilizados para la validación de algoritmos de control. Su atractivo como banco de pruebas reside en que combina en un sistema de baja dimensión tres propiedades que lo hacen especialmente exigente: inestabilidad intrínseca en el punto de operación, no linealidad de las ecuaciones del movimiento y subactuación, al disponer de un único actuador para controlar cuatro variables de estado (Ogata, 1995). La utilización como benchmark es probablemente su aplicación más difundida y se ha empleado en numerosos trabajos para evaluar diversos controladores como LQR, control robusto, control adaptativo o control por modos deslizantes (Irfan et al., 2024).

Además, la dinámica del péndulo invertido reproduce de manera fiel el comportamiento de muchos sistemas reales de gran relevancia tecnológica, como los que se indican a continuación:

2.2.1. Robots bípedos

En robots bípedos, el modelo se usa para aproximar la dinámica del centro de gravedad, como en el 3D-LIPM (Linear Inverted Pendulum Mode) (Graf & Röfer, 2012), donde la altura del CG se mantiene fija y el punto de apoyo modela el origen variable del péndulo. En este trabajo se corrige la trayectoria del péndulo usando un filtro de Kalman que estima el estado del CG a partir de la cinemática de las articulaciones, para posteriormente ajustar los parámetros del modelo en vez de aplicar una ley de control directa. Se manifiesta así la versatilidad del modelo de péndulo invertido, independientemente del mecanismo de actuación y de la estrategia de control aplicada.

2.2.2. Vehículos autoequilibrados (*Segway*)

En vehículos autoequilibrados la formulación del péndulo invertido es también utilizada, en esta el chasis actúa como péndulo y el eje de las ruedas como carro (Gong et al., 2015). El vector de estado $x = [x \ \dot{x} \ \phi \ \dot{\phi}]^T$ y la linealización en torno al punto de equilibrio son análogos al control que se pretende implementar en este TFG. El estudio implementa y compara diferentes estrategias de control (PID y LQR). Los resultados mostraron menos sobreimpulso y más rapidez en la respuesta del controlador lineal cuadrático. Sin embargo, la validación se limitó a la simulación en Simulink, sin dar el paso a la verificación en hardware real.

2.2.3. Despegue de vehículos aeroespaciales

El comportamiento dinámico de los cohetes durante la fase inicial de despegue puede aproximarse al de un péndulo invertido, debido a su inestabilidad intrínseca en el eje de cabeceo. Esta analogía ha sido utilizada para el diseño de controladores de actitud especialmente mediante técnicas como LQR y control por vectorización de empuje (Ramos & Filho, 2007). Se desarrolla un modelo no lineal que incluye fricción viscosa e inercia distribuida del péndulo y se optimiza la matriz de pesos de Q y R con un algoritmo genético. Una de las conclusiones sacadas es que al variar un parámetro el crecimiento en las ganancias del regulador puede ser excesivo si no se penalizan específicamente las variaciones entre iteraciones sucesivas.

2.3. Implementaciones en microcontroladores

Tradicionalmente, los sistemas de control basados en LQR se han implementado sobre plataformas de procesamiento secuencial como microcontroladores, procesadores digitales de señal (DSP) y sistemas embebidos. Estas soluciones presentan un bajo coste, facilidad de programación y amplia disponibilidad, lo que ha favorecido su uso en contextos docentes e industriales. En un microcontrolador, el algoritmo de control se ejecuta como una tarea periódica: en cada instante de muestreo el procesador lee los sensores, evalúa la ley de control $\mathbf{u}_k = -K_d \mathbf{x}_k$ y escribe la salida al actuador de forma secuencial. El tiempo de cómputo necesario para completar este ciclo determina la latencia mínima del sistema y limita la frecuencia de muestreo máxima alcanzable.

En la literatura se encuentran varios ejemplos con estas características:

Neacsu y Sîrbu, 2020 implementaron un controlador LQR para un *boost-converter* sobre un microcontrolador Microchip dsPIC33, con una frecuencia de muestreo de 250 kHz. En este caso se construyó la función de coste de LQR a partir del modelo de pérdidas del convertidor, de modo que la solución de la ecuación de Riccati proporciona ganancias que minimizan la disipación durante los transitorios de referencia de la tensión. Se logró ejecutar el bucle de control completo en menos de 4 μs , ayudándose también de las rutinas de producto escalar de la librería DSP del fabricante.

En sistemas multivariable también se encuentran implementaciones en estas plataformas, Dhewa et al., 2017 emplearon un microcontrolador ARM Cortex-M4 para estabilizar la orientación (*roll*, *pitch* y *yaw*) de un cuadricóptero en hover utilizando un controlador lineal

cuadrático. Los autores reportan tiempos de recuperación ante perturbaciones del orden de 90–150 ms para los ejes de roll y pitch, con errores en régimen permanente inferiores a 2° respecto a la desviación máxima del eje. Este trabajo demuestra la viabilidad del LQR sobre procesadores de tipo ARM para sistemas multivariable, aunque la frecuencia de actualización queda condicionada por el tiempo de lectura de la IMU y el ciclo de ejecución del bucle principal.

En el ámbito de los péndulos invertidos, trabajo directamente relacionado con el presente TFG, Karhan et al. (2024) desarrollaron e implementaron un controlador LQR sobre un microcontrolador STM32F4 para estabilizar un péndulo de rueda de reacción. El modelo del sistema se obtiene aplicando mecánica lagrangiana y se linealiza en torno al punto de equilibrio inestable, resultando en un espacio de estados de tres variables $(\theta_1, \dot{\theta}_1, \dot{\theta}_2)$. Dado que los términos de fricción no son directamente medibles, se estiman de forma experimental mediante mínimos cuadrados no lineales. El código de control se genera desde MATLAB/Simulink usando el Waijung Blockset y se carga directamente en el STM32F4, operando a una frecuencia de muestreo de 100 Hz ($T_s = 0,01$ s). Los resultados experimentales muestran que el controlador es capaz de estabilizar el sistema incluso ante perturbaciones que lo alejan considerablemente del punto de linealización. Como limitación, los autores identifican que el uso de derivadas filtradas introduce retardos que penalizan la estabilización, y proponen el uso de un estimador de estado como línea de mejora futura.

2.4. Implementaciones en FPGA

Como alternativa, las FPGA han surgido como una plataforma especialmente adecuada para aplicaciones de control en tiempo real. Su capacidad de procesamiento en paralelo y su naturaleza determinista permiten reducir significativamente la latencia y aumentar la frecuencia de actualización del sistema de control. A diferencia de un microcontrolador, donde el algoritmo de control se ejecuta de forma secuencial, una FPGA permite implementar el cálculo de $\mathbf{u}_k = -K_d \mathbf{x}_k$ como un circuito combinacional pipelineado en el que todas las multiplicaciones se realizan en paralelo. El retardo total queda determinado únicamente por la profundidad del camino crítico, independientemente del número de estados del sistema.

Algunos trabajos en los que se han podido apreciar estas ventajas son los siguientes:

Zhang et al., 2017 propusieron un coprocesador LQR configurable por software sobre una FPGA Zynq (XC7Z020), aplicado al péndulo invertido sobre carro. El sistema estaba

basado en un árbol binario de multiplicadores en paralelo que permite reformular la ley de control $\mathbf{u}_k = -K\hat{\mathbf{x}}_k$ como una sola multiplicación matriz-vector, reduciendo así el retardo de control. Ejecutado a 100 MHz, el coprocesador calcula u_k en 530 ns y completa una iteración completa del algoritmo en 600 ns, lo que equivale a una frecuencia de actualización máxima de aproximadamente 1,7 MHz.

También en controladores más avanzados se pueden encontrar ejemplos, Torrestiana González et al., 2026 llevaron a cabo una implementación completa del controlador LQG discreto sobre una FPGA Artix-7 para controlar un rodamiento magnético activo de tres grados de libertad, apoyándose en una frecuencia de muestreo de 2 kHz ($T_s = 0,5$ ms). Toda la aritmética se realiza en punto fijo con formato Q8.24 de 32 bits, y las matrices estáticas del controlador y el estimador se almacenan en memoria BRAM y se procesan mediante un motor de multiplicación-acumulación (MAC) multiplexado en tiempo sobre los bloques DSP48E1 del dispositivo. La ejecución completa del ciclo se completa en menos de 200 μ s, dejando un margen de 300 μ s dentro del período de muestreo.

De forma más general, Khan, 2026 revisa las estrategias de diseño e implementación de controladores en tiempo real sobre FPGA para aplicaciones industriales, comparando esta plataforma frente a DSP y procesadores ARM. Los resultados mostraron que a diferencia de los microcontroladores, cuya ejecución secuencial introduce latencia y jitter que pueden comprometer el rendimiento del lazo de control, las FPGA permiten ejecutar múltiples operaciones de forma concurrente con un comportamiento temporal completamente determinista. El trabajo también analiza las implicaciones del uso de aritmética en punto fijo, señalando que una elección adecuada de la longitud de palabra y los factores de escala resulta esencial para mantener la precisión y la estabilidad del algoritmo de control sin incurrir en un consumo excesivo de recursos hardware.

La Tabla 1 resume las diferencias más relevantes discutidas entre ambas plataformas desde el punto de vista del control en tiempo real, a partir de los trabajos revisados en estas secciones.

Tabla 1: Comparativa entre microcontrolador y FPGA para implementación de controladores LQR.

Característica	Microcontrolador	FPGA
Ejecución	Secuencial	Paralela
Latencia	$\mu s - ms$	$ns - \mu s$
Determinismo	Limitado	Total
Frecuencia de muestreo	Hasta ~ 100 kHz	Hasta $\sim MHz$
Aritmética nativa	Entera / FP32	Punto fijo
Complejidad de diseño	Baja	Alta
Coste	Bajo	Medio/alto

2.5. Limitaciones actuales y hueco en la literatura

A pesar de los avances en la implementación de controladores LQR sobre distintas plataformas, los estudios que realizan una comparativa cuantitativa rigurosa entre FPGA y microcontrolador sobre un mismo sistema de control siguen siendo escasos. Los trabajos existentes suelen centrarse en una única plataforma o, cuando comparan, lo hacen en términos de tiempo de cómputo puro sin extender el análisis al rendimiento del lazo cerrado sobre una planta física real (Zhang et al., 2017).

Del mismo modo, la validación experimental sobre sistemas físicos representativos no siempre acompaña a la implementación hardware. Aunque trabajos recientes como el de Torrestiana González et al., 2026 sí incluyen validación sobre planta real, el sistema utilizado de un rodamiento magnético activo tiene una dinámica muy distinta al péndulo invertido sobre carro, que presenta inestabilidad intrínseca, no linealidades y restricciones de actuación distintas. Por lo que la transferibilidad de esos resultados al sistema subactuado de este TFG no puede asumirse directamente.

Un tercer aspecto no tan explorado es el consumo energético comparativo de ambas plataformas en condiciones de operación reales. Si bien las FPGA ofrecen ventajas claras en latencia y paralelismo, su consumo estático puede ser superior al de un microcontrolador de gama media, por lo que la ventaja energética no es universal y depende del perfil de uso del sistema.

Este trabajo pretende cubrir estos vacíos mediante tres contribuciones concretas: en primer lugar, una implementación completa del controlador LQR discreto en FPGA con validación sobre el sistema físico real del péndulo invertido sobre carro; en segundo lugar, una

comparativa cuantitativa de latencia, frecuencia de actualización y consumo de recursos frente a una implementación equivalente en microcontrolador sobre la misma planta; y en tercer lugar, un análisis del consumo energético de ambas plataformas en condiciones de operación comparables.

Capítulo 3 Metodología

En este capítulo se describirá el proceso seguido para el diseño e implementación del controlador LQR discreto sobre la placa Basys-3 (Digilent Inc., 2016), así como la implementación equivalente sobre microcontrolador, que posteriormente servirá de referencia para el análisis comparativo. El desarrollo se estructura en torno a la metodología habitual en sistemas de control embebido:

- Definición del problema y los requisitos que debe cumplir el sistema de control.
- Modelado del sistema físico: parámetros mecánicos, actuador y sensor.
- Diseño del controlador LQR en tiempo continuo y su posterior discretización.
- Adaptación aritmética al hardware de destino.
- Diseño de la arquitectura hardware en FPGA y descripción de los módulos VHDL.
- Implementación alternativa en microcontrolador como referencia comparativa.

Cada sección documenta las decisiones de diseño tomadas y su justificación técnica.

3.1. Planteamiento del problema y requisitos del sistema

El péndulo invertido sobre carro (*cart-pole system*) es un sistema dinámico no lineal, inestable y de orden 4. Esta inestabilidad lo convierte en un banco de pruebas exigente para evaluar plataformas de implementación. El objetivo del presente trabajo no es solamente estabilizar el péndulo, sino hacerlo sobre hardware real y comparar cualitativamente el rendimiento de una FPGA como plataforma de ejecución del controlador lineal cuadrático.

3.1.1. Objetivo de control

Se busca que el controlador sea capaz de mantener el péndulo en equilibrio en la posición vertical invertida ($\theta = 0, \dot{\theta} = 0$) con el carro en el centro del carril ($x = 0, \dot{x} = 0$), tal y como se ilustra en la Figura 4, desde unas condiciones iniciales dentro de la región de linealidad del modelo (es decir, un ángulo cercano al entorno del punto de equilibrio, el margen elegido es $|\theta| \leq 15^\circ$), sin que el carro rebase los límites físicos del carril.

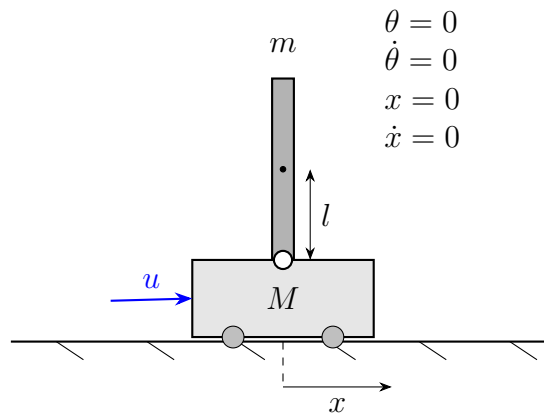


Figura 4: Posición objetivo del control: péndulo en equilibrio invertido ($\theta = 0$).

3.1.2. Requisitos funcionales

1. El sistema debe ser capaz de estabilizar el péndulo para perturbaciones iniciales de hasta $\pm 15^\circ$ respecto a la vertical.
2. El controlador debe ejecutarse con un periodo de muestreo $T_s \leq 5 \text{ ms}$ para garantizar la estabilidad del lazo de control ante la dinámica del sistema.
3. La señal de control calculada debe traducirse en comandos de velocidad para el motor paso a paso antes del siguiente instante de muestreo.
4. El sistema debe disponer de una secuencia automática de puesta en marcha: referenciado del carro (*homing*), posicionamiento central y, a continuación se buscará implementar dos formas de funcionamiento: o bien espera de elevación manual del péndulo hasta la zona de captura, o bien ejecución automática del algoritmo de *swing-up* para elevar el péndulo desde la posición colgante.
5. Si el péndulo supera $\pm 30^\circ$ durante el control activo, el sistema debe detener el motor de forma inmediata, dando por fallido el control.

3.1.3. Requisitos de implementación FPGA

1. La implementación debe caber en los recursos disponibles del Artix-7 XC7A35T y ser compatible con los niveles lógicos 3,3V del Pmod JA de la Basys3.

2. La latencia total del lazo de control (desde la captura del encoder hasta la emisión del primer pulso STEP con la nueva velocidad) debe ser inferior a T_s , garantizando que el actuador recibe la nueva consigna dentro del mismo periodo de muestreo.
3. La implementación debe exponer métricas cuantificables (latencia, recursos consumidos) que permitan una comparación directa con la implementación equivalente en otras plataformas.

3.1.4. Métricas de evaluación

La comparativa se realizará a través de las siguientes métricas:

- **Latencia del lazo:** tiempo desde la llegada del tick de muestreo hasta que la nueva señal de control está disponible en la salida del generador de pulsos. En la FPGA se mide en ciclos de reloj; en el microcontrolador, en microsegundos mediante temporizador de captura.
- **Frecuencia máxima de muestreo alcanzable:** inversa de la latencia mínima del lazo completo en cada plataforma.
- **Consumo de recursos lógicos:** número de LUT, registros, bloques DSP y bloques BRAM utilizados (solo FPGA), obtenido del informe de implementación de Vivado.
- **Error de cuantización:** desviación entre la ganancia real del LQR y la ganancia representable en el formato adoptado.
- **Rendimiento de control:** tiempo de estabilización del péndulo, desplazamiento máximo del carro y margen de perturbación angular tolerable en condiciones de lazo cerrado real.

3.2. Descripción del sistema físico

Se validará el controlador sobre el sistema físico de péndulo invertido sobre carro construido en el laboratorio (Rodríguez Criado et al., 2025) cuya estructura mecánica, actuadores y sensores se describen a continuación.

3.2.1. Estructura mecánica

El carro se desliza sobre un carril lineal con una carrera útil de 320 mm. La transmisión entre el motor y el carro se realiza mediante una correa GT2 con polea de 40 dientes con pasos de 2 mm, lo que proporciona un avance lineal de 80 mm por vuelta del motor. La masa del carro, incluyendo el soporte del péndulo, es $M = 0,272$ kg.

El péndulo es una varilla rígida con un pivote que permite su articulación al encoder, dicha articulación daba una longitud al CG de 8,7 cm (medida desde el pivote). Puesto que la longitud total de la varilla era de 24 cm, se decidió añadir una masa adicional en el extremo libre del péndulo para desplazar el CG al centro de la varilla a 12 cm. Dicha masa fue calculada como se indica a continuación:

$$m_v \cdot d_v = m_a \cdot 2l \quad (3.1)$$

$$m_a = \frac{m_v \cdot d_v}{2l} \quad (3.2)$$

Donde m_v es la masa de la varilla, d_v es la distancia del centro de masa de la varilla al pivote (la mitad de la longitud total de la varilla) y $2l$ es la longitud total de la varilla. Se obtiene así finalmente una masa total de 0,027 kg. El momento de inercia del péndulo respecto al CG se calcula como $I_p = \frac{1}{12}m(2l)^2$ (varilla uniforme). Dado que se añadió una masa adicional en el extremo libre para centrar el CG (Ecuación 3.2), el péndulo deja de ser estrictamente una varilla uniforme, por lo que esta expresión se emplea como una aproximación de ingeniería. En la Tabla 2 se resumen los parámetros físicos del sistema necesarios para el modelado matemático.

Tabla 2: Parámetros físicos del sistema (confirmados experimentalmente).

Símbolo	Descripción	Valor	Unidades
M	Masa del carro	0,272	kg
m	Masa del péndulo	0,027	kg
l	Longitud al CG	0,12	m
I_p	Momento de inercia (péndulo)	$1,296 \times 10^{-4}$	kg·m ²
g	Aceleración gravitatoria	9,81	m/s ²
b	Coef. fricción viscosa	0	N·s/m

3.2.2. Actuador: motor paso a paso NEMA17 con driver DRV8825

El actuador es un motor paso a paso NEMA17 (Creality, 2024) controlado por el driver DRV8825 de Texas Instruments (Texas Instruments, 2014).



Figura 5: Motor paso a paso NEMA17.

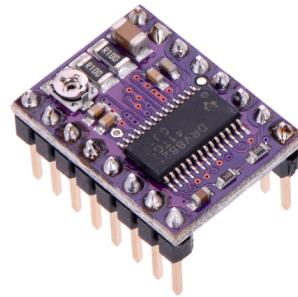


Figura 6: Driver DRV8825.

El motor es de 200 pasos por vuelta y opera en modo de 1/8 micropasos para lograr una resolución más alta, configurado mediante los pines `MODE0=1`, `MODE1=1`, `MODE2=0` siguiendo las indicaciones de la *datasheet*, lo que proporciona 1600 micropasos por vuelta. Con la polea de 40 dientes (80 mm/vuelta), la resolución de posición del carro es:

$$r_{\text{mec}} = \frac{80 \text{ mm}}{1600 \text{ micropasos}} = 0,05 \text{ mm/micropaso} \quad (3.3)$$

El motor se controla a partir de una señal de pulso STEP, que indica al driver que el motor debe avanzar un micropaso, y una señal de dirección DIR, que indica si el avance es positivo o negativo. La frecuencia de pulso STEP determinará la velocidad del motor, y para que el driver pueda leer el paso la señal STEP debe aguantar activa al menos $1,9 \mu\text{s}$.

3.2.3. Sensor: encoder óptico de cuadratura

La posición angular del péndulo se mide con un encoder óptico incremental de cuadratura de 600 pulsos por revolución (p/r) (Industrial Shields, 2024). Consta de dos canales de salida en cuadratura (canales A y B) desfasados 90° entre sí, lo que permite determinar el sentido de rotación y la velocidad de giro. El decodificador de cuadratura detecta los cuatro flancos de cada ciclo (dos por fase), por ello el número de cuentas por revolución es de $4 \times 600 = 2400$,

lo que proporciona una resolución de $360^\circ/2400 = 0,15^\circ$ /cuenta (0,0026 rad). Las salidas del encoder son de colector abierto y se conectan a la FPGA con resistencias de pull-up a 3,3 V.



Figura 7: Encoder óptico de cuadratura.

3.2.4. Plataforma FPGA: Basys3 (Artix-7)

La FPGA utilizada es la Xilinx Artix-7 XC7A35T-1CPG236C, montada en la placa de desarrollo Basys3 de Digilent (Digilent Inc., 2016).

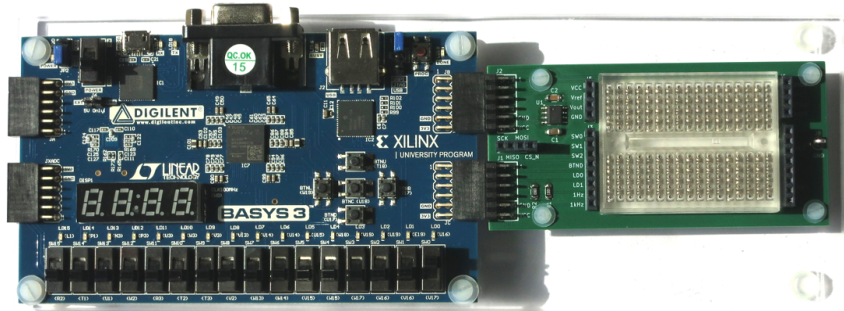


Figura 8: Placa de desarrollo Basys3.

Sus características principales relevantes para este proyecto son:

Tabla 3: Recursos disponibles en la Artix-7 XC7A35T (Digilent Inc., 2016).

Recurso	Disponible	Tipo
Slice LUT	20 800	Lógica / ROM
Flip-flop	41 600	Registro
BRAM (36 Kb)	50	Memoria
DSP48E1	90	Multiplicador-acumulador
Oscilador en placa	hasta 450 MHz	Reloj

Todos los módulos de I/O hacia los periféricos externos (motor, encoder, final de carrera) se enrutan a través del conector Pmod JA. El reloj del sistema es de 100 MHz, lo que proporciona una resolución temporal de 10 ns por ciclo.

3.2.5. Interfaz eléctrica y conexionado Pmod

La Tabla 4 recoge la asignación completa de pines de la Basys3. El conexionado físico entre los periféricos y la FPGA se muestra en la Figura 9, y una fotografía del sistema completo montado se incluye en la Figura 11.

Tabla 4: Asignación de pines de la Basys3 (XC7A35T-1CPG236C).

Señal	Pin FPGA	Conector	I/O	Descripción
<code>rst</code>	U18	btnC	I	Reset síncrono
<code>enc_A</code>	J1	Pmod JA1	I	Canal A encoder
<code>enc_B</code>	L2	Pmod JA2	I	Canal B encoder
<code>limit_fdc_i</code>	G3	Pmod JA10	I	Final de carrera
<code>step_out</code>	G2	Pmod JA4	O	STEP DRV8825
<code>dir_out</code>	H1	Pmod JA7	O	DIR DRV8825
<code>motor_en</code>	H2	Pmod JA9	O	ENBL DRV8825
<code>tick_1ms</code>	U16	LD0	O	Tick de muestreo
<code>lqr_enable_dbg</code>	P1	LD14	O	LQR activo
<code>state_dbg[0]</code>	P3	LD10	O	Bit 0 FSM
<code>state_dbg[1]</code>	U3	LD11	O	Bit 1 FSM
<code>state_dbg[2]</code>	W3	LD12	O	Bit 2 FSM
<code>still_led_o</code>	E19	LD1	O	Péndulo quieto

Los aspectos más relevantes del conexionado son los siguientes:

- Todos los periféricos comparten una referencia de masa común (FPGA, driver DRV8825, fuente del motor y encoder), imprescindible para la correcta interpretación de los niveles lógicos.
- El driver DRV8825 se alimenta con 12 V, desde una fuente externa, mientras que la Basys3 se alimenta por USB a 5 V.
- Los pines MODE0, MODE1 y MODE2 del driver se configuran conectándolos a masa o a 3,3 V según el modo de micropasos deseado. Como se ha mencionado previamente, en este

trabajo se usa la configuración $MODE0=1$, $MODE1=1$, $MODE2=0$, correspondiente a 1/8 de micropaso.

- Se coloca un condensador electrolítico de desacoplo entre los terminales de alimentación del motor (VMOT y GND) del driver DRV8825 de capacidad $100\mu F$ y tensión nominal 20 V, para absorber los picos de corriente generados por la conmutación del motor y proteger el driver frente a sobretensiones transitorias.
- El final de carrera es un interruptor normalmente cerrado (NC) situado en el extremo izquierdo del carril, alimentado directamente desde la FPGA con pull-up interno activado en el pin correspondiente.
- El encoder óptico soporta tensiones de alimentación de hasta 24 V, pero se alimenta a 5 V desde la fuente de alimentación externa. Sus salidas son de colector abierto, por lo que se activan los pull-up internos de la FPGA en los pines JA1 y JA2.

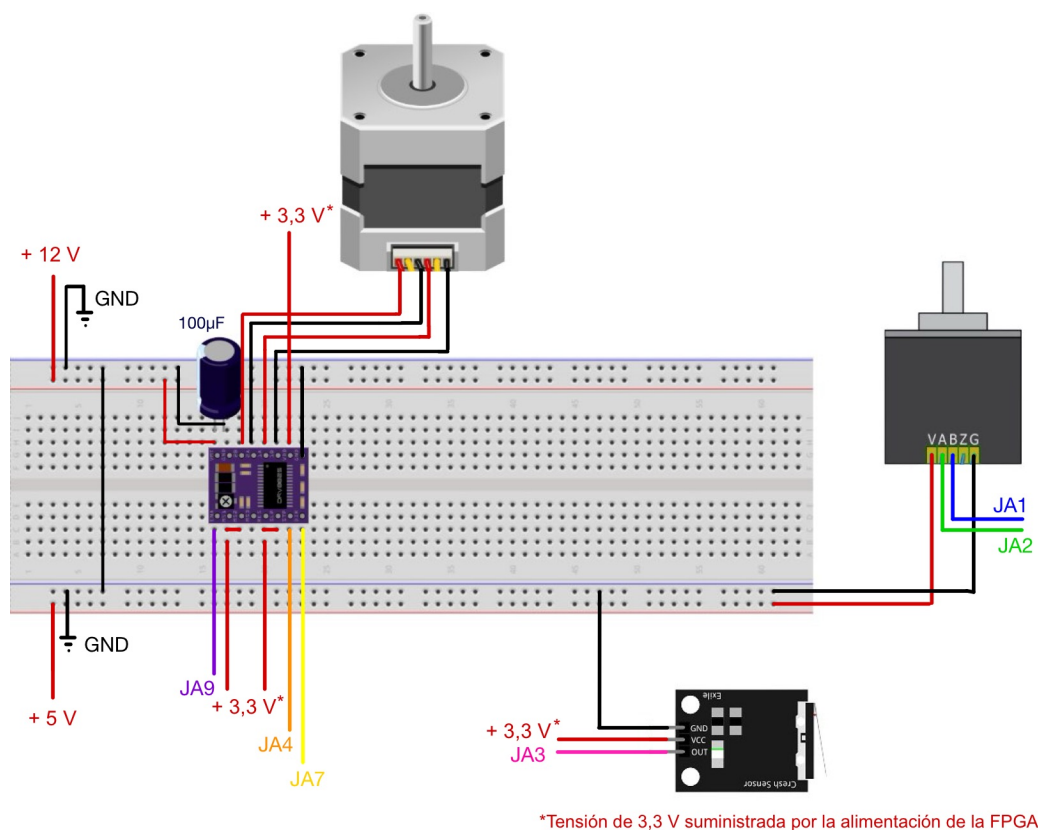


Figura 9: Diagrama de conexionado del sistema.

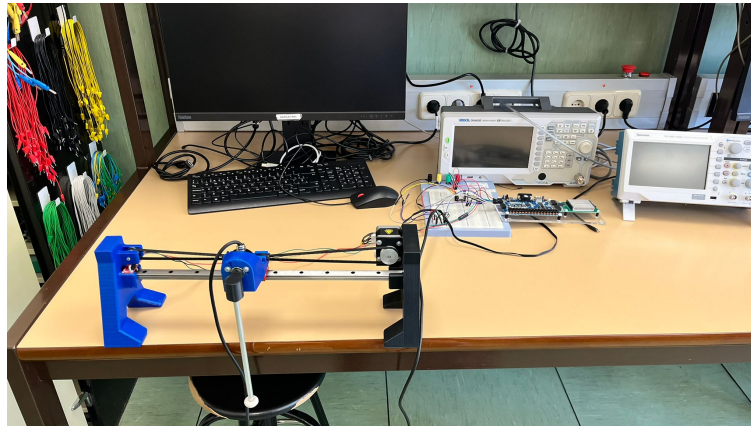


Figura 10: Sistema péndulo invertido sobre carro completamente montado en el laboratorio.

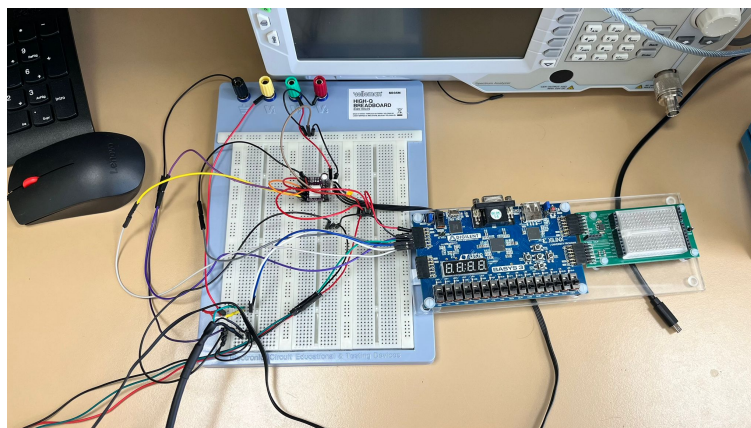


Figura 11: Circuito del sistema completamente montado en el laboratorio.

3.3. Diseño del controlador LQR

3.3.1. Fundamentos teóricos del regulador LQR

Las técnicas de control óptimo constituyen una rama del control automático cuyo objetivo es encontrar la ley de control que minimiza una función de coste definida sobre el comportamiento del sistema. A diferencia de los métodos clásicos, que diseñan el controlador a partir de especificaciones en frecuencia o en el lugar de las raíces, el control óptimo parte de un criterio de optimización explícito que permite incorporar de forma sistemática los requisitos de rendimiento y las limitaciones del actuador (Franklin et al., 2014).

Entre las técnicas más destacadas se encuentran el control por modelo predictivo (MPC),

que optimiza la acción de control sobre un horizonte temporal finito teniendo en cuenta restricciones explícitas sobre estados y entradas, y el control óptimo estocástico, que extiende el problema al caso en que el sistema está sometido a perturbaciones aleatorias (Kouvaritakis & Cannon, 2016). En concreto, el controlador lineal cuadrático (LQR) se ha establecido como una de las técnicas más recurrentes en aplicaciones de ingeniería gracias a su capacidad de proporcionar soluciones que logran un compromiso entre la rapidez de respuesta del sistema y el esfuerzo de control requerido, con garantías formales de estabilidad en lazo cerrado (Lewis et al., 2012).

El desarrollo teórico que se presenta a continuación sigue la formulación presentada en Ogata (2010). El regulador LQR se formula partiendo de un sistema lineal en espacio de estados de la forma $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$, donde $\mathbf{x} \in \mathbb{R}^n$ es el vector de estado y $\mathbf{u} \in \mathbb{R}^m$ es la señal de control. El objetivo es encontrar la ley de control $\mathbf{u} = -K\mathbf{x}$ que minimice la función de coste cuadrática:

$$J = \int_0^\infty (\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u}) dt \quad (3.4)$$

donde $Q \in \mathbb{R}^{n \times n}$ es una matriz semidefinida positiva que penaliza las desviaciones del estado, y $R \in \mathbb{R}^{m \times m}$ es una matriz definida positiva que penaliza el esfuerzo de control.

Una matriz Q es semidefinida positiva ($Q \succeq 0$) si para todo vector $\mathbf{x} \neq \mathbf{0}$ se cumple $\mathbf{x}^T Q \mathbf{x} \geq 0$; esto permite asignar peso nulo a estados cuya desviación no se desea penalizar directamente. En contraste, R debe ser definida positiva ($R \succ 0$, es decir, $\mathbf{x}^T R \mathbf{x} > 0$ para todo $\mathbf{x} \neq \mathbf{0}$) para garantizar que el problema de optimización tenga solución única y que el esfuerzo de control esté siempre acotado.

La elección de estas matrices determina el compromiso entre rapidez de respuesta y coste energético del actuador.

La solución óptima se obtiene resolviendo la ecuación algebraica de Riccati (ARE):

$$A^T P + P A - P B R^{-1} B^T P + Q = 0 \quad (3.5)$$

donde P es la solución matricial simétrica definida positiva. A partir de ella, la ganancia óptima es $K = R^{-1} B^T P$, lo que da lugar a un controlador de realimentación de estado que garantiza estabilidad asintótica del sistema en lazo cerrado.

Para que el regulador LQR tenga solución, es condición necesaria que el sistema sea

controlable, es decir, que sea posible llevar el estado $\mathbf{x}$ desde cualquier condición inicial hasta el origen en tiempo finito mediante una entrada $\mathbf{u}$ adecuada. Formalmente, esto se verifica comprobando que la matriz de controlabilidad $\mathcal{C} = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix}$ tiene rango completo. En la práctica, para el diseño LQR es suficiente con la condición más débil de estabilizabilidad (todos los modos inestables del sistema son controlables) junto con la detectabilidad del par $(A, Q^{1/2})$ (todos los modos inestables son observables), lo que garantiza que la solución P de la ecuación de Riccati sea definida positiva. Cuando el sistema es completamente controlable y observable, ambas condiciones se satisfacen automáticamente.

Para su implementación en sistemas digitales, el controlador debe formularse en tiempo discreto; la formulación seguida en esta sección se basa en Ogata (1995). El proceso de discretización parte del sistema continuo $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$ y asume que la señal de control se mantiene constante entre instantes de muestreo, lo que se conoce como retención de orden cero (ZOH, *Zero-Order Hold*). Bajo esta hipótesis, las matrices discretas equivalentes son:

$$A_d = e^{AT_s}, \quad B_d = \int_0^{T_s} e^{A\tau} B d\tau \quad (3.6)$$

donde T_s es el periodo de muestreo. La elección de T_s es crítica: un valor demasiado grande introduce error de discretización y puede desestabilizar el sistema, mientras que un valor demasiado pequeño aumenta la carga computacional del hardware. Como regla general, T_s debe ser entre 10 y 20 veces menor que la constante de tiempo dominante del sistema en lazo cerrado.

El problema de optimización discreto minimiza la función de coste:

$$J = \sum_{k=0}^{\infty} (\mathbf{x}_k^T Q \mathbf{x}_k + \mathbf{u}_k^T R \mathbf{u}_k) \quad (3.7)$$

cuya solución requiere resolver la ecuación discreta de Riccati (DARE), implementada en MATLAB mediante la función `dlqr()`. La ganancia discreta K_d resultante se aplica en cada instante de muestreo k como $\mathbf{u}_k = -K_d \mathbf{x}_k$, lo que hace este enfoque directamente implementable en hardware digital.

3.3.2. Modelo del péndulo invertido sobre carro

El sistema del péndulo invertido sobre un carro (*cart-pole*) consiste en un carro de masa M que se desplaza horizontalmente sobre un raíl. Sobre el carro se articula un péndulo de masa m y longitud hasta el centro de gravedad l . La única entrada de control es la fuerza horizontal u aplicada al carro. Cabe destacar que en el sistema real el actuador es un motor paso a paso, cuya magnitud regulable no es una fuerza sino la velocidad (frecuencia de pasos). La forma en que se resuelve esta discrepancia se detalla en la Sección 3.7.5.

La formulación matemática que se desarrolla a continuación sigue el modelo presentado en University of Michigan (2024).

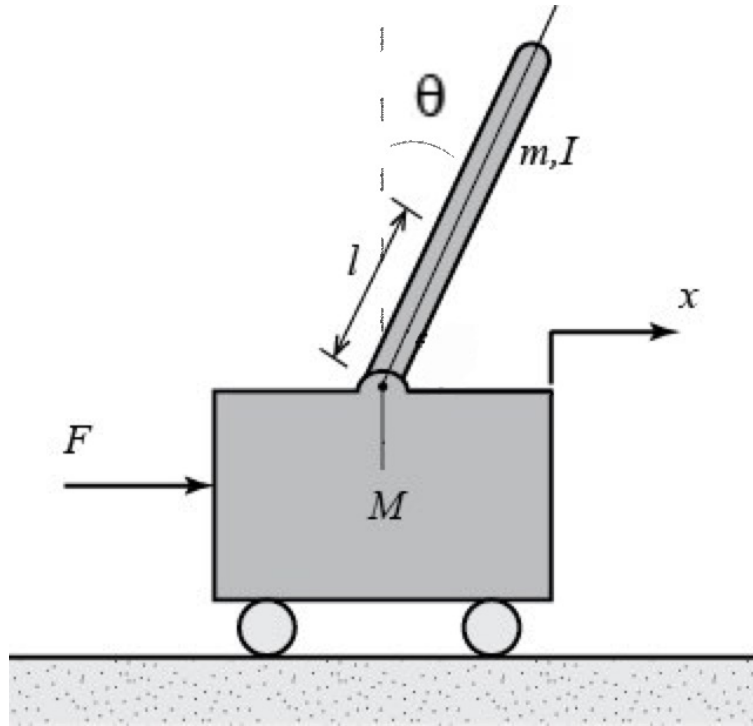


Figura 12: Esquema del sistema péndulo invertido sobre carro (University of Michigan, 2024).

El vector de estado del sistema es:

$$\mathbf{x} = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^T \quad (3.8)$$

donde x es la posición del carro, $\dot{x}$ su velocidad, θ el ángulo del péndulo respecto a la vertical y $\dot{\theta}$ su velocidad angular. El punto de equilibrio inestable que se desea estabilizar es $\mathbf{x}^* = \mathbf{0}$, correspondiente al péndulo en posición vertical con el carro en el origen.

Los diagramas de cuerpo libre de ambos elementos se muestran en la Figura 13, donde N y V son las reacciones horizontal y vertical que ejerce el péndulo sobre el carro, y b es el coeficiente de rozamiento viscoso del raíl.

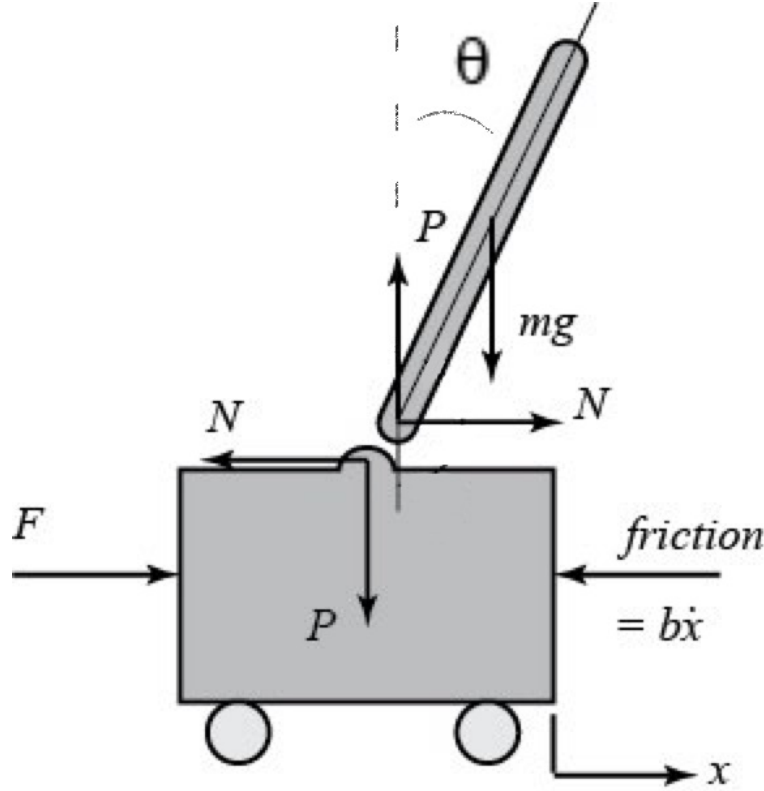


Figura 13: Fuerzas actuando sobre el carro y el péndulo (University of Michigan, 2024).

Aplicando la segunda ley de Newton al carro en la dirección horizontal:

$$M\ddot{x} + b\dot{x} = u - N \quad (3.9)$$

La posición del centro de masa del péndulo en coordenadas absolutas es $x_{cm} = x + l \sin \theta$, $y_{cm} = l \cos \theta$. Diferenciando dos veces:

$$\ddot{x}_{cm} = \ddot{x} + l\ddot{\theta} \cos \theta - l\dot{\theta}^2 \sin \theta \quad (3.10)$$

$$\ddot{y}_{cm} = -l\ddot{\theta} \sin \theta - l\dot{\theta}^2 \cos \theta \quad (3.11)$$

Aplicando la segunda ley de Newton al péndulo en la dirección horizontal se obtiene la

expresión de N :

$$N = m\ddot{x} + ml\ddot{\theta} \cos \theta - ml\dot{\theta}^2 \sin \theta \quad (3.12)$$

Sustituyendo (3.12) en (3.9) se obtiene la primera ecuación del movimiento:

$$(M + m)\ddot{x} + b\dot{x} + ml\ddot{\theta} \cos \theta - ml\dot{\theta}^2 \sin \theta = u \quad (3.13)$$

Para la segunda ecuación se suman momentos respecto al pivote, considerando el momento de inercia del péndulo respecto al CG, $I_p = \frac{1}{12}m(2l)^2$, trasladado al pivote mediante el teorema de Steiner:

$$(I_p + ml^2)\ddot{\theta} - mgl \sin \theta = -ml\ddot{x} \cos \theta \quad (3.14)$$

Las ecuaciones (3.13) y (3.14) son no lineales debido a los términos trigonométricos y al producto $\dot{\theta}^2$. Para aplicar el diseño LQR se linealizan en torno al equilibrio $\theta = 0$, $\dot{\theta} = 0$, empleando las aproximaciones de ángulo pequeño:

$$\sin \theta \approx \theta, \quad \cos \theta \approx 1, \quad \dot{\theta}^2 \approx 0 \quad (3.15)$$

con lo que las ecuaciones del movimiento se simplifican a:

$$(M + m)\ddot{x} + b\dot{x} + ml\ddot{\theta} = u \quad (3.16)$$

$$(I_p + ml^2)\ddot{\theta} - mgl\theta = -ml\ddot{x} \quad (3.17)$$

Definiendo el denominador $\Delta = I_p(M + m) + Mml^2$ y resolviendo el sistema (3.16)–(3.17) para $\ddot{x}$ y $\ddot{\theta}$:

$$\ddot{x} = \frac{(I_p + ml^2)(u - b\dot{x}) + m^2gl^2\theta}{\Delta} \quad (3.18)$$

$$\ddot{\theta} = \frac{ml(u - b\dot{x}) + mgl(M + m)\theta}{\Delta} \quad (3.19)$$

Introduciendo el vector de estado $\mathbf{x} = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^T$, las expresiones (3.18) y (3.19)

permiten escribir directamente el sistema en espacio de estados $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{(I_p + ml^2)b}{\Delta} & \frac{m^2 gl^2}{\Delta} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{mlb}{\Delta} & \frac{mgl(M+m)}{\Delta} & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{I_p + ml^2}{\Delta} \\ 0 \\ \frac{ml}{\Delta} \end{bmatrix} \quad (3.20)$$

En el modelo empleado en este trabajo se considera $b = 0$ como aproximación inicial. Bajo esta simplificación, las matrices quedan:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{m^2 gl^2}{\Delta} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{mgl(M+m)}{\Delta} & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ \frac{I_p + ml^2}{\Delta} \\ 0 \\ \frac{ml}{\Delta} \end{bmatrix} \quad (3.21)$$

Este modelo linealizado es válido únicamente en un entorno del punto de equilibrio, por lo que el controlador LQR diseñado a partir de él garantiza estabilidad local. En la práctica, esto implica que el sistema debe partir de condiciones iniciales suficientemente próximas a la vertical para que el controlador pueda actuar eficazmente antes de que la linealización deje de ser una aproximación válida.

Tras sustituir los parámetros físicos de la Tabla 2, los valores numéricos de las matrices A y B son:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0,7127 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 65,7666 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 3,5874 \\ 0 \\ 22,4215 \end{bmatrix} \quad (3.22)$$

Los valores propios de A son $\{0, 0, +8,1097, -8,1097\}$: los dos en el origen corresponden a los modos de traslación del carro (integradores puros), y el valor propio positivo confirma la inestabilidad intrínseca del sistema, justificando la necesidad de control activo. La verificación de controlabilidad mediante `rank(ctrb(A,B))` devuelve rango 4, condición necesaria para la existencia de solución del problema LQR.

3.3.3. Elección de las matrices de peso Q y R

El regulador LQR minimiza la función de coste cuadrática $J = \int_0^\infty (\mathbf{x}^T Q \mathbf{x} + u^T R u) dt$. Las matrices Q y R constituyen los grados de libertad de diseño: Q penaliza las desviaciones del estado y R penaliza el esfuerzo de control.

La elección de los pesos parte del método de Bryson como punto de partida, fijando los valores iniciales inversamente proporcionales al cuadrado del máximo desvío admisible en cada variable de estado (Bryson & Ho, 1975):

$$Q_{ii} = \frac{1}{x_{i,\text{máx}}^2}, \quad R = \frac{1}{u_{\text{máx}}^2} \quad (3.23)$$

No obstante, este criterio habitualmente es tan solo el punto de partida de un proceso de iteraciones de prueba y error orientado a obtener las propiedades deseadas en el sistema en lazo cerrado (Hespanha, 2005). Siguiendo este enfoque, a partir de los valores físicos (carrera máxima ± 160 mm, velocidad máxima estimada $\pm 0,5$ m/s, ángulo máximo de actuación $\pm 15^\circ \approx \pm 0,26$ rad, velocidad angular máxima ± 2 rad/s) se realizó un barrido paramétrico de las diagonales de Q , adoptando la siguiente configuración final:

$$Q = \text{diag}(10, 10, 100, 1), \quad R = 1 \quad (3.24)$$

El hecho de que el peso más alto corresponda al ángulo del péndulo ($Q_{33} = 100$), refleja que la prioridad del controlador es mantener la verticalidad del péndulo. Un peso más moderado es empleado para penalizar la posición y velocidad del carro ($Q_{11} = 10$). El valor $R = 1$ hace que la señal de control no sea demasiado agresiva intentando evitar así que el controlador exija velocidades de carro superiores a las mecánicamente posibles.

Los resultados del barrido confirman que la configuración (3.24) ofrece compromiso razonable entre rapidez de respuesta ($t_s < 1,5$ s al 2 %), desplazamiento máximo del carro ($|x|_{\text{máx}} < 120$ mm) y velocidad máxima del carro ($\dot{x}_{\text{máx}} \approx 0,48$ m/s), dentro de los límites físicos del sistema y de las capacidades dinámicas del motor paso a paso NEMA17. Las gráficas del barrido paramétrico se presentan en el Capítulo 4.

3.3.4. Ganancias del LQR continuo

La ganancia óptima K se calcula resolviendo la ecuación algebraica de Riccati (3.5) mediante la función `lqr(A,B,Q,R)` de MATLAB. La ganancia resultante es:

$$K = \begin{bmatrix} -3,1623 & -4,6283 & 20,6479 & 2,4877 \end{bmatrix} \quad (3.25)$$

3.3.5. Verificación de estabilidad en tiempo continuo

La estabilidad del sistema en lazo cerrado se verifica comprobando que todos los valores propios de $A - BK_c$ tienen parte real estrictamente negativa, condición necesaria y suficiente para la estabilidad asintótica de un sistema lineal continuo (Ogata, 2010):

$$\text{Re}(\lambda_i(A - BK_c)) < 0 \quad \forall i = 1, \dots, 4 \quad (3.26)$$

Los valores propios obtenidos en MATLAB mediante `eig(A - B*Kc)` son:

$$\lambda_1 = -25,6272 \quad \lambda_2 = -9,9283 \quad \lambda_3 = -2,5448 \quad \lambda_4 = -1,0743 \quad (3.27)$$

Todos presentan parte real negativa, confirmando la estabilidad asintótica del sistema en lazo cerrado.

3.3.6. Simulación en tiempo continuo

Se valida el diseño del controlador mediante simulación en MATLAB/Simulink antes de pasar a la discretización del mismo. Para ello, se integra numéricamente el sistema en lazo cerrado $\dot{\mathbf{x}} = (A - BK)\mathbf{x}$ con condiciones iniciales $\mathbf{x}_0 = [0, 0, 0, 1745, 0]^T$ (péndulo desplazado 10° de la vertical) y se observa la evolución temporal de los estados.

Los resultados de la simulación se explorarán más en detalle en el Capítulo 4, pero de manera preliminar se observa que el sistema converge al equilibrio en menos de 3s, con desplazamiento máximo del carro inferior a 120 mm y velocidad máxima del carro inferior a 0,5 m/s, cumpliendo así los objetivos de diseño.

3.4. Discretización del controlador

3.4.1. Método de retención de orden cero (ZOH)

Para implementar el controlador LQR en un sistema digital de tiempo discreto se discretiza el sistema continuo mediante el método de retención de orden cero (ZOH, *Zero-Order Hold*). Este método asume que la señal de control permanece constante entre instantes de muestreo, que es precisamente la hipótesis de los sistemas de control digital implementados con conversores DAC con retención.

Las matrices discretas equivalentes son:

$$A_d = e^{AT_s}, \quad B_d = \int_0^{T_s} e^{A\tau} B d\tau \quad (3.28)$$

donde T_s es el periodo de muestreo. La exponencial de matriz se calcula numéricamente en MATLAB mediante `c2d(sys, Ts, 'zoh')` (Franklin et al., 2014).

3.4.2. Elección del periodo de muestreo

La constante de tiempo dominante del sistema en lazo cerrado continuo, obtenida a partir de los valores propios de $A - BK_c$, es $\tau_{\text{dom}} = 0,931$ s. Aplicando el criterio habitual de diseño $T_s \leq \tau_{\text{dom}}/10$, se obtiene un límite superior de $T_s \leq 93$ ms. El periodo de muestreo adoptado $T_s = 1$ ms es aproximadamente 93 veces inferior a dicho límite, garantizando que la discretización no introduce error dinámico apreciable.

Se adopta $T_s = 1$ ms por los siguientes motivos:

- Proporciona margen de seguridad amplio respecto al criterio anterior.
- En la FPGA, el tick de 1 ms se genera mediante un contador de 100 000 ciclos sobre el reloj de 100 MHz, sin coste de recursos apreciable.
- La latencia total del lazo de control VHDL es de unas pocas decenas de ciclos (véase Sección 3.6), lo que deja 99 950+ ciclos disponibles dentro del periodo para otras tareas.

Para $T_s = 1$ ms las matrices discretas resultan:

$$A = \begin{bmatrix} 1 & 0,001 & 0 & 0 \\ 0 & 1 & 0,0007 & 0 \\ 0 & 0 & 1 & 0,001 \\ 0 & 0 & 0,0658 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0,0036 \\ 0 \\ 0,0224 \end{bmatrix} \quad (3.29)$$

3.4.3. Ganancias del LQR discreto

Una vez discretizado el sistema, la ganancia óptima K_d se calcula mediante la función `dlqr(A_d,B_d,Q,R)` de MATLAB, que minimiza la función de coste discreta $J = \sum_{k=0}^{\infty} (\mathbf{x}_k^T Q \mathbf{x}_k + \mathbf{u}_k^T R \mathbf{u}_k)$ y resuelve la ecuación discreta de Riccati (DARE).

La ganancia discreta resultante es:

$$K_d = [-3,1009, -4,5400, 20,3267, 2,4497] \quad (3.30)$$

cada componente del vector pondera la contribución de cada estado a la señal de control en el instante de muestreo k . Por tanto la ley de control discreta aplicada en cada instante de muestreo es:

$$u_k = -K_d \mathbf{x}_k \quad (3.31)$$

3.4.4. Verificación de estabilidad discreta

Para verificar la estabilidad del sistema discreto en lazo cerrado, se calculan los autovalores de la matriz $A_d - B_d K_d$. La condición de estabilidad requiere que todos los autovalores estén dentro del círculo unitario en el plano complejo, es decir, que su módulo sea menor que 1:

$$|\lambda_i(A_d - B_d K_d)| < 1 \quad \forall i = 1, \dots, 4 \quad (3.32)$$

Los valores propios obtenidos en MATLAB mediante `eig(A_d - B_d K_d)` cumplen esta condición:

$$|\lambda_1| = 0,9747, \quad |\lambda_2| = 0,9901, \quad |\lambda_3| = 0,9975, \quad |\lambda_4| = 0,9989 \quad (3.33)$$

Como se aprecia, todos los autovalores tienen módulo menor que 1, confirmando que el sistema discreto en lazo cerrado es estable.

3.4.5. Simulación en tiempo discreto

Se valida el controlador discreto simulando el sistema paso a paso con $T_s = 1$ ms, aplicando la ley de control (3.31) en cada instante de muestreo. El modelo utilizado es el sistema lineal discreto en lazo cerrado $\mathbf{x}_{k+1} = (A_d - B_d K_d) \mathbf{x}_k$, integrado para condición inicial $\theta_0 = 10^\circ$ con el carro en reposo en el origen.

Los resultados muestran convergencia en todos los casos, con un comportamiento muy similar al de la simulación en tiempo continuo, lo que confirma que el periodo de muestreo elegido ($T_s = 1$ ms) no introduce error de discretización apreciable. Las gráficas comparativas entre la simulación continua y discreta se presentan en el Capítulo 4.

3.5. Aritmética en punto fijo

La primera decisión previa a la implementación hardware es la elección del formato aritmético. Las dos opciones principales son la aritmética en punto flotante (estándar IEEE 754) y la aritmética en punto fijo. Aunque el punto flotante ofrece mayor rango dinámico y facilidad de uso, su implementación en FPGA presenta inconvenientes significativos: la emulación de una unidad de punto flotante en lógica LUT consume entre 200 y 1000 LUTs por operación, y su latencia es considerablemente mayor que la de un multiplicador entero. Si bien la Artix-7 XC7A35T dispone de bloques DSP48E1 capaces de realizar multiplicaciones enteras de 18×25 bits con latencia de un ciclo de reloj (Xilinx Inc., 2018), no incluye unidades de punto flotante en hardware (AMD Xilinx, 2021).

En contraste, la aritmética en punto fijo opera directamente sobre enteros, aprovechando al máximo los bloques DSP48E1 y manteniendo una latencia predecible y mínima. El coste es una reducción del rango dinámico y la necesidad de un análisis previo para garantizar que las señales no desbordan el formato elegido (Bishop, 2008). Dado que las variables del sistema (posición, velocidad, ángulo) operan en rangos acotados y conocidos de antemano, este análisis es perfectamente tratable, lo que hace del punto fijo la elección natural para este proyecto.

3.5.1. Representación en punto fijo: formato $Qm.n$

En la notación $Qm.n$, siguiendo la formulación de Yates (2009), m representa el número de bits enteros (incluyendo el bit de signo en representación con signo) y n el número de bits fraccionarios, sobre un total de $m + n$ bits. La relación entre el valor entero almacenado v_{int} y el valor real representado v_{real} es:

$$v_{\text{real}} = \frac{v_{\text{int}}}{2^n} \quad (3.34)$$

El rango representable es $[-2^{m-1}, 2^{m-1} - 2^{-n}]$ y la resolución (valor de 1 LSB) es 2^{-n} .

Suma y resta. Dos números en el mismo formato $Qm.n$ se suman directamente como enteros; el resultado es también $Qm.n$. Si los operandos tienen formatos distintos, es necesario alinear previamente el punto binario mediante desplazamiento aritmético. El principal riesgo es el desbordamiento (*overflow*): si la suma supera el rango representable, el resultado se corrompe. La solución habitual es ampliar el número de bits del acumulador antes de sumar y aplicar saturación al resultado final.

Multiplicación. El producto de un número $Qm_1.n_1$ por un número $Qm_2.n_2$ produce un resultado $Q(m_1 + m_2).(n_1 + n_2)$ con el doble de bits. Para restituir el formato original $Qm.n$ es necesario desplazar el resultado n posiciones a la derecha (dividir entre 2^n) y truncar o redondear al número de bits deseado:

$$(a \times b)_{Qm.n} = \left\lfloor \frac{a_{\text{int}} \times b_{\text{int}}}{2^n} \right\rfloor \quad (3.35)$$

En VHDL esto se implementa extrayendo la ventana de bits correcta del resultado completo, sin necesidad de una división explícita.

Desbordamiento y saturación. El desbordamiento ocurre cuando el resultado de una operación supera el rango del formato. En aritmética en complemento a dos sin saturación, el desbordamiento produce un cambio de signo incorrecto (el resultado da la vuelta). Para evitarlo se implementa saturación: si el resultado supera el máximo representable se fija al máximo, y si es inferior al mínimo se fija al mínimo. En VHDL esto requiere calcular el resultado en un formato extendido y comparar antes de truncar.

3.5.2. Formatos elegidos y análisis de rango de variables

El análisis de rango de las variables del sistema determinará el formato $Qm.n$ mínimo necesario para representar las variables sin desbordamiento. Se consideran los valores máximos esperados de cada variable durante la operación normal del controlador, siendo estos obtenidos de la simulación en MATLAB en tiempo discreto.

Formato principal: Q8.8. Se adopta el formato Q8.8 con signo en palabras de 16 bits como formato estándar del sistema, con las características recogidas en la Tabla 5:

Tabla 5: Características del formato Q8.8 signed de 16 bits.

Propiedad	Valor
Bits totales	16
Bits enteros (con signo)	8
Bits fraccionarios	8
Resolución (1 LSB)	$1/256 \approx 3,906 \times 10^{-3}$
Rango	$[-128,000, +127,996]$

La adecuación de este formato a cada variable de estado se verifica en la Tabla 6:

Tabla 6: Rango físico de los estados para verificar su representabilidad en formato Q8.8.

Estado	Variable	Rango físico
x_1	Posición carro	$\pm 0,16$ m
x_2	Velocidad carro	$\pm 0,5$ m/s
x_3	Ángulo péndulo	$\pm 0,26$ rad
x_4	Vel. angular	± 2 rad/s

Como se observa en la Tabla 6, todos los estados son representables en formato Q8.8 dentro de su rango físico esperado.

3.5.3. Error de cuantización de las ganancias y constantes

La conversión de las ganancias K_d al formato Q8.8 introduce un error de cuantización:

$$\varepsilon_{\text{quant}} = k_i - \frac{\text{round}(k_i \times 256)}{256} \quad (3.36)$$

Tabla 7: Errores de cuantización de las ganancias LQR en formato Q8.8.

Ganancia	k_i real	Entero Q8.8	k_i Q8.8	Error relativo
k_1	-3,1009	-794	-3,1016	0,02 %
k_2	-4,5400	-1162	-4,5391	0,02 %
k_3	+20,3267	+5204	+20,3281	0,007 %
k_4	+2,4497	+627	+2,4492	0,02 %

Los errores de cuantización son inferiores al 0,03 % en todos los casos, lo que es despreciable frente a las incertidumbres del modelo físico (estimación de masas, fricción, posición del CG). Se concluye que el formato Q8.8 de 16 bits es adecuado para representar las ganancias LQR del sistema.

Además del vector K , el diseño emplea otras dos constantes en formato Q8.8 para las conversiones de unidades del diferenciador y del estimador de posición del carro, cuyos errores de cuantización se muestran en la Tabla 8:

Tabla 8: Otras constantes en formato Q8.8 y su error de cuantización.

Constante	Valor real	Entero Q8.8	Valor representado	Error relativo
$K_{VEL} = 2\pi/2400 \times 1000$	2,617994	670	2,617188	-0,03 %
$K_{x_2} = 3277/256$ (Q8.8/pasos) $\approx 12,8$	12,8	3277	12,80078	+0,006 %

Los errores de estas constantes son también despreciables, concluyendo que el formato es adecuado para su representación.

3.5.4. Análisis de los productos intermedios

Una vez fijadas las ganancias en formato Q8.8, se verifica que los productos intermedios $k_i \cdot x_i$ y su suma no desbordan el rango representable. Para cada producto, el valor máximo esperado se obtiene combinando la ganancia cuantizada con el valor máximo del estado correspondiente (Tabla 6):

Tabla 9: Rango máximo de los productos intermedios LQR para verificar su representabilidad en formato Q8.8.

Producto	k_i Q8.8	$x_{i,\max}$	$ k_i \cdot x_{i,\max} $
$k_1 x_1$	-3,1016	0,16 m	0,496
$k_2 x_2$	-4,5391	0,5 m/s	2,270
$k_3 x_3$	+20,3281	0,26 rad	5,285
$k_4 x_4$	+2,4492	2 rad/s	4,898
$u = \sum k_i x_i$	—	—	< 13

Todos los productos intermedios y la suma tienen magnitud muy inferior al límite del formato, lo que confirma que no se produce desbordamiento en ninguna operación del controlador durante la operación normal. La saturación implementada en el `sumador_saturador` actúa únicamente como salvaguarda ante perturbaciones excepcionales, no en régimen normal de funcionamiento.

Formato Q0.16 para constantes sub-unitarias. Las constantes de conversión del encoder a variables de estado tienen magnitud inferior a $1/256$, lo que haría que su representación en Q8.8 redondeara a cero. Las características del formato se recogen en la Tabla 10:

Tabla 10: Características del formato Q0.16 de 16 bits y error relativo tras la conversión.

Propiedad	Valor
Bits totales	16
Bits enteros (con signo)	0
Bits fraccionarios	16
Resolución (1 LSB)	$1/65536 \approx 1,526 \times 10^{-5}$
Rango	$[-1,000, +0,99998]$

Las constantes que usan este formato son K_{ANG} (para convertir el número de cuentas a unidades angulares), $K_1 = T_s/M$ (para convertir la fuerza de control a aceleración del carro) y $K_{x_1} = 256/20000$ (para convertir el conteo de pasos del motor a posición del carro en Q8.8). En la Tabla 11 se pueden observar los errores de cuantización por la conversión:

Tabla 11: Constantes sub-unitarias en formato Q0.16 y su error de cuantización.

Constante	Valor real	Entero Q0.16	Valor representado	Error relativo
$K_{ANG} = 2\pi/2400$ rad/cuenta	0,002618	172	0,002625	+0,25 %
$K_1 = 0,001/0,272$ s/kg	0,003676	241	0,003677	+0,02 %
$K_{x_1} = 256/20000$ (Q8.8/pasos)	0,012800	839	0,012802	+0,017 %

3.6. Arquitectura hardware en FPGA

3.6.1. Visión general del sistema

El sistema implementado en la FPGA sigue una arquitectura modular jerárquica en la que cada función del controlador está encapsulada en un módulo VHDL independiente. Esta organización que facilita el ajuste de parámetros y la reutilización de los módulos en iteraciones futuras.

La jerarquía de módulos se indica en la Figura 14 y se describe a continuación:

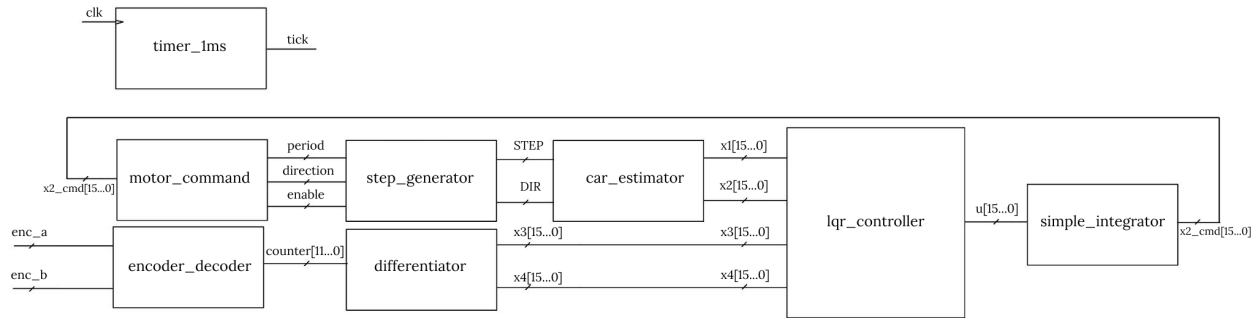


Figura 14: Diagrama de bloques del sistema completo (`lqr_system_top`).

- `lqr_system_top` — Módulo de nivel superior en el que se instancian todos los componentes del sistema. Realiza las conexiones entre señales internas y mapea los puertos a los pines físicos del dispositivo.
- `timer_1ms` — Generador del tick de cada periodo de muestreo. Divide el reloj de 100 MHz para producir un pulso de 1 ciclo de ancho cada 1 ms (T_s).

- **encoder_decoder** — Decodificador de cuadratura. Convierte las señales A y B del encoder óptico en un contador con signo de 12 bits.
- **differentiator** — Calcula el ángulo y la velocidad angular del péndulo (señales x_2 y x_3) mediante diferencia finita de la señal del encoder.
- **lqr_controller** — Implementa la ley de control LQR en cada ciclo de muestreo, calculando la señal de mando u a partir del vector de estados $\mathbf{x} = [x_1, x_2, x_3, x_4]^T$ y las ganancias K_d . Haciendo uso de los siguientes bloques, tal y como se indica en la Figura 15 :

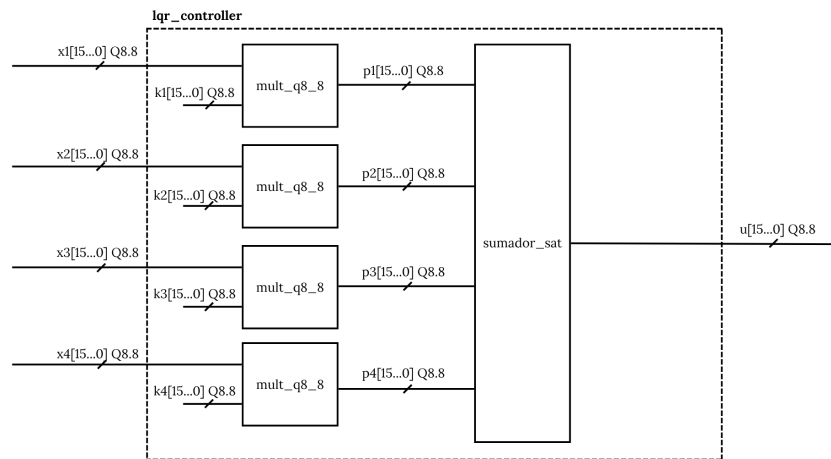


Figura 15: Diagrama de bloques del controlador LQR (**lqr_controller**).

- **mult_q8_8** — Módulo de multiplicación en punto fijo Q8.8. Calcula los productos $k_i \cdot x_i$ para cada estado.
- **sumador_sat** — Suma los cuatro productos y aplica saturación para evitar desbordamiento.
- **simple_integrator** — Módulo de integración discreta para estimar la velocidad del carro a partir de la señal de control u en cada ciclo de muestreo.
- **motor_command** — Encargado de convertir la señal de velocidad del carro en un periodo de pulso para el generador de pasos del motor. Implementa además la máquina de estados para la rampa de aceleración y desaceleración del motor paso a paso, asegurando que la velocidad de giro no supere los límites mecánicos del sistema.

- **step_generator** — Genera los pulsos STEP y DIR para el motor paso a paso, basándose en el periodo calculado por **motor_command**.
- **cart_estimator** — A partir de los pulsos STEP y DIR, estima la posición y la velocidad del carro con las que se calcula la ley de control en cada periodo de muestreo.

3.6.2. Máquina de estados del sistema

La secuencia de operación del sistema se gestiona mediante una máquina de estados finitos (FSM) de Moore con cinco estados:

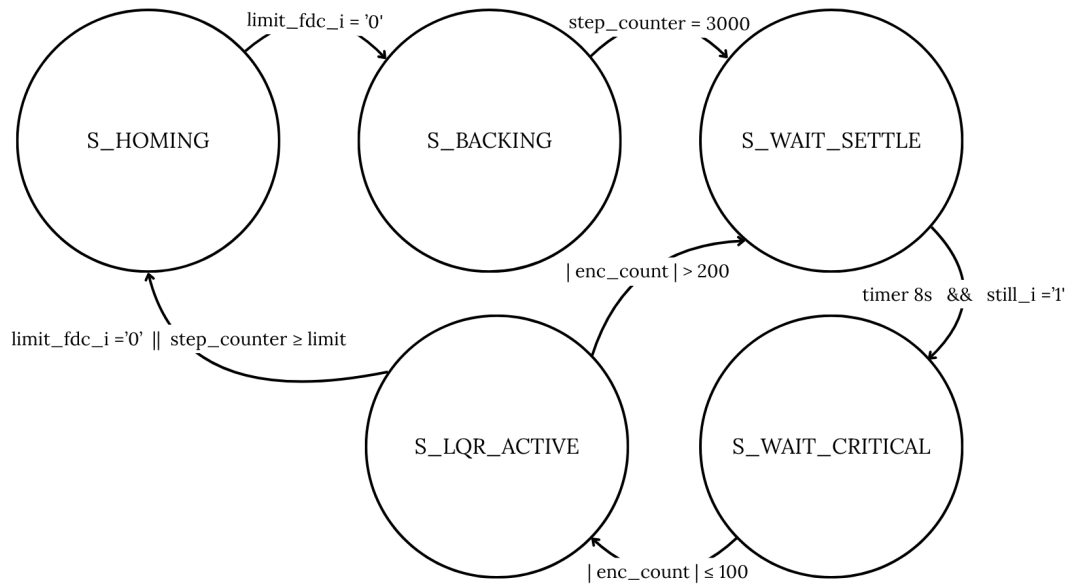


Figura 16: Máquina de estados del sistema (**lqr_system_fsm**).

Esta máquina de estados se describe en el bloque **lqr_system_fsm** y controla la transición entre los estados descritos en función de las señales de entrada (finales de carrera, lectura del encoder) y las condiciones de seguridad (desviación angular del péndulo, límite de recorrido útil).

S_HOMING: El motor avanza a frecuencia fija (5 kHz, $T_{\text{homing}} = g_clk_hz / g_homing_freq_hz$ ciclos de reloj) hacia el final de carrera izquierdo. Cuando la señal `limit_fdc.i` se activa (nivel lógico '0', activo bajo), se transita a **S_BACKING**.

S_BACKING: El motor retrocede $N = 3\,000$ micropasos para posicionar el carro en el centro del carril. La distancia equivalente es $3000 \times 0,05\text{ mm} = 150\text{ mm}$, aproximadamente la mitad de la carrera útil. Los pasos se cuentan mediante realimentación de `step_tick.i`. Al llegar, se transita a **S_WAIT_SETTLE**.

S_WAIT_SETTLE: Estado de dos subfases. En la primera, el motor permanece parado durante un tiempo genérico configurable (*g_settle_time_s*) para que el operador coloque el péndulo. Agotado ese tiempo, la FSM espera a una segunda condición: que el encoder lleve 1 s sin cambios (`still_i='1'`, señal del `stillness_detector`). Al cumplirse ambas subfases, se captura el valor del encoder como referencia `count_offset_reg` y se transita a **S_WAIT_CRITICAL**.

S_WAIT_CRITICAL: Motor inactivo y LQR deshabilitado. El operador eleva manualmente el péndulo hacia la vertical. La FSM monitoriza continuamente `count_rel = enc_count_i - count_offset_reg + 1200`; cuando $|\text{count_rel}| \leq 100$ ($\approx \pm 15^\circ$), se activa el LQR y se transita a **S_LQR_ACTIVE**.

S_LQR_ACTIVE: La señal `lqr_enable_o='1'` habilita el tick gateado hacia el controlador. El movimiento del motor es responsabilidad de la cadena `lqr_controller` $\rightarrow$ `motor_command` $\rightarrow$ `step_generator`. Existen tres salidas de este estado:

- Si el péndulo supera 200 cuentas ($\approx \pm 30^\circ$), el controlador ha perdido la capacidad de recuperación y se transita a **S_WAIT_CRITICAL** para reiniciar el proceso de estabilización.
- Si el carro activa el final de carrera izquierdo, se transita a **S_HOMING**.
- Si el carro alcanza el límite de pasos del lado derecho (donde no existe final de carrera físico), el conteo interno de pasos (`step_cnt_lqr = 2 \times g\_center\_offset\_steps`) detecta el límite y se transita igualmente a **S_HOMING**.

Ante cualquiera de los tres fallos, el sistema reintenta automáticamente sin límite de reintentos, sin necesidad de intervención externa.

3.7. Implementación en VHDL

Esta sección describe en detalle cada módulo VHDL del sistema, sus interfaces y las decisiones de implementación relevantes. Todos los códigos fuente se encuentran en el repositorio de GitHub del proyecto (<https://github.com/alopezgarciarom/lqr-inverted-pendulum-fpga>), y los más relevantes se incluyen en el Anexo D.

3.7.1. Módulo `timer_1ms`

El generador de tick implementa un contador ascendente desde 0 hasta $LIMITE = (f_{CLK}/10^6) \times T_{s,\mu s} - 1 = 99\,999$. En el ciclo en que el contador alcanza su valor máximo se emite un pulso de 1 ciclo de ancho (`tick='1'`, equivalente a 10 ns) y el contador se resetea a cero. Los parámetros `CLK_FREQ` y `Ts_US` son genéricos, lo que permite reutilizar el módulo con distintos relojes o periodos de muestreo sin modificar el código.

3.7.2. Módulo `encoder_decoder`

Decodificador de cuadratura para el encoder rotativo. Registra las señales A y B del encoder en dos etapas consecutivas de flip-flop para sincronización con el dominio de reloj de la FPGA y eliminación de metaestabilidad. La decodificación se realiza comparando el estado actual (A_n, B_n) con el estado anterior (A_{n-1}, B_{n-1}) para determinar la dirección de giro, incrementando o decrementando un contador de `BITS=12` bits con signo (rango ± 2047 , equivalente a $\pm 307^\circ$), este límite no es problemático ya que se espera que el péndulo opere en un rango de ± 1 vuelta. El código fuente completo se incluye en el Anexo D.

3.7.3. Módulo `stillness_detector`

Detecta cuándo el péndulo lleva un tiempo determinado sin generar cambios en el encoder, señal empleada por la FSM del sistema (`S_WAIT_SETTLE`) para confirmar que el operador ha soltado el péndulo antes de capturar la referencia angular. En cada ciclo de reloj se compara el valor actual de `count_in` con el registrado en el ciclo anterior: si difieren, se reinicia un contador interno y `still_led` baja a '0'; si permanecen iguales durante `T_STILL_S = 1 s` (10^8 ciclos a 100 MHz), `still_led` se activa a '1' y se mantiene mientras no se detecten nuevos cambios en el encoder.

3.7.4. Módulo `differentiator`

Convierte las cuentas del encoder en los estados x_3 (ángulo, Q8.8) y x_4 (velocidad angular, Q8.8) mediante multiplicación en punto fijo.

Cálculo de x_3 : La constante de conversión $K_{\text{ANG}} = 2\pi/2400 \approx 0,002618 \text{ rad/cuenta}$ se representa en formato Q0.16 con valor entero 172 (error relativo +0,25 %), ya que en Q8.8 redondearía a cero. El producto `count` (extendido a 28 bits) $\times$ `K_ANG` (18 bits) produce un resultado Q12.16 de 46 bits; los bits [23:8] dan x_3 en Q8.8.

Cálculo de x_4 : Se calcula la diferencia finita hacia atrás $\Delta\text{count} = \text{count}[k] - \text{count}[k-1]$, extendida a 14 bits para absorber transitorios. La constante $K_{\text{VEL}} = 670$ en Q8.8 (error -0,031 %) se multiplica directamente por Δcount (entero puro); el resultado hereda los 8 bits fraccionarios de K_{VEL} sin desplazamiento adicional. Se aplica saturación explícita al rango Q8.8 antes de registrar.

Ambos estados se registran sincronamente en el flanco de `tick`, con latencia de 1 ciclo de reloj.

3.7.5. Módulo `simple_integrator`

Integra la señal de control u (Q8.8) mediante el método de Euler hacia adelante para estimar la consigna de velocidad del carro `x2_cmd` (Q8.8, m/s):

$$x_{2,\text{cmd}}[k+1] = x_{2,\text{cmd}}[k] + u[k] \cdot \frac{T_s}{M} \quad (3.37)$$

La constante $K_1 = T_s/M = 0,001/0,272 \approx 0,003676$ se representa en Q0.16 con valor entero 241 (error +0,024 %). El producto $u \times K_1$ produce un resultado Q8.24 de 34 bits; los bits [31:16] dan el incremento $\Delta x_{2,\text{cmd}}$ en Q8.8. Se aplica saturación al resultado antes de registrar. Este módulo proporciona la consigna de velocidad `x2_cmd` a `motor_command`, resolviendo el problema de arranque desde velocidad cero (arranque en frío).

3.7.6. Módulo `mult_q8_8`

Multiplicador genérico $\text{Q8.8} \times \text{Q8.8} \rightarrow \text{Q8.8}$. El producto completo de 32 bits se calcula en lógica combinacional; los bits [23:8] se extraen para reescalar el resultado Q16.16 a Q8.8 (desplazamiento aritmético de 8 bits a la derecha). El resultado se registra en un flip-flop

habilitado por `en` (conectado al `tick`). Esta estructura (lógica combinacional seguida de registro de salida) es la que Vivado infiere como bloque DSP48E1 con registro P interno, minimizando la ruta crítica.

3.7.7. Módulo `sumador_sat`

Recibe los cuatro productos parciales p_1, p_2, p_3, p_4 en Q8.8, los suma en aritmética de 19 bits para evitar desbordamiento intermedio (cuatro sumandos de 16 bits pueden crecer hasta 18 bits), niega el resultado ($u = -(k_1x_1 + k_2x_2 + k_3x_3 + k_4x_4)$) y satura al rango Q8.8 $[-32768, +32767]$. El módulo se habilita con `tick_d1` (tick retrasado 1 ciclo en `lqr_controller`), garantizando que los cuatro multiplicadores han completado y registrado sus productos antes de sumar, evitando la captura de valores obsoletos.

3.7.8. Módulo `lqr_controller`

Módulo estructural que instancia los cuatro multiplicadores Q8.8 y el sumador-saturador, implementando la ley de control $u = -K_d \mathbf{x}$. Las ganancias se definen como constantes de tipo `signed(15 downto 0)`: $K_1 = -794$, $K_2 = -1162$, $K_3 = +5204$, $K_4 = +627$ (valores en Q8.8, escala 256). Un proceso síncrono genera `tick_d1` retrasando `tick` un ciclo de reloj, que se conecta al enable del `sumador_sat` para resolver el desfase de pipeline entre multiplicadores y sumador. La latencia total del módulo es de 2 ciclos de reloj (20 ns a 100 MHz).

3.7.9. Módulo `motor_command`

Convierte la consigna de velocidad `x2_cmd` (Q8.8, m/s) en las señales `period`, `direction` y `enable` para el generador de pulsos, implementando además una rampa trapezoidal de aceleración y deceleración.

Conversión de velocidad a periodo: El número de micropasos por segundo se calcula como $(|x_2|_{\text{int}} \times 20000) \gg 8$ (factor Q8.8), y el periodo en ciclos de reloj se obtiene dividiendo $f_{\text{CLK}} = 100 \text{ MHz}$ entre dicho valor mediante el IP Core *Divider Generator* de Vivado (Radix-2, 32 bits, latencia fija de 34 ciclos). Se protege la división por cero sustituyendo divisores nulos por 1 antes de lanzar la operación.

Máquina de estados de la rampa: El módulo implementa una FSM de tres estados para gestionar la rampa de velocidad, tal y como se muestra en la Figura 17.

LQR_IDLE: Motor parado ($|x_2| \leq X2_MIN$, umbral $\approx 0,012$ m/s). El registro de periodo se precarga a $PER_ARRANQUE = 50\,000$ ciclos (equivalente a 2 000 pasos/s, velocidad mínima de arranque), de modo que la siguiente activación parte siempre desde la velocidad más baja.

LQR_RIGHT / LQR_LEFT: Motor activo en sentido positivo o negativo. En cada ciclo de llegada del resultado de la división ($dout_tvalid='1'$, 34 ciclos tras el tick), se calcula el periodo objetivo per_vel saturado al rango $[PER_MINIMA, PER_ARRANQUE] = [6\,250, 50\,000]$ ciclos, y se mueve $period_reg$ un paso de $PER_STEP = 100$ ciclos hacia per_vel . Esto limita la aceleración máxima y evita la pérdida de paso por *pull-in torque* del motor.

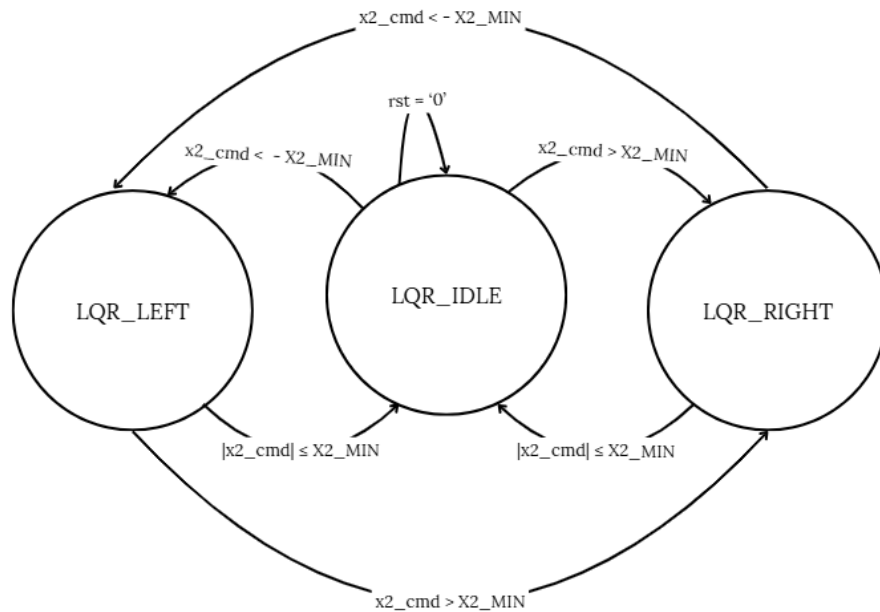


Figura 17: Máquina de estados de la rampa de velocidad (**motor_command**).

3.7.10. Módulo **step_generator**

Genera los pulsos STEP y DIR para el driver DRV8825 a partir del periodo especificado por **motor_command** durante el control LQR, o por la propia FSM del sistema durante las fases de arranque (*homing* y centrado). Implementa un contador descendente que se recarga con **period** en cada flanco de STEP. El ancho del pulso STEP está fijado por el genérico **PULSE_CICLOS** para garantizar el valor que cumple con el requisito mínimo de anchura de

pulso del DRV8825. La señal `step_tick_o` produce un pulso de 1 ciclo en cada micropaso generado, utilizada tanto por `cart_estimator` para estimar la posición y velocidad del carro como por la FSM para contabilizar los pasos durante la fase de centrado (`S_BACKING`).

3.7.11. Módulo `cart_estimator`

Estima la posición x_1 (m, Q8.8) y la velocidad x_2 (m/s, Q8.8) del carro a partir del conteo de pulsos STEP y la señal de dirección real del motor, proporcionando los estados reales del carro para la ley de control.

Un contador de 32 bits (`step_counter`) se incrementa o decrementa en cada pulso `step_tick` según la dirección, y se reinicia cuando `lqr_enable='0'`. En cada tick de muestreo se calculan:

- x_1 a partir de `step_counter` saturado a ± 3200 pasos ($\pm 0,16$ m), multiplicado por la constante 839 en Q0.16 y desplazado 16 bits: error +0,017 %.
- x_2 a partir de $\Delta \text{steps} = \text{step_counter}[k] - \text{step_counter}[k - 1]$, multiplicado por 3277 en Q8.8 y desplazado 8 bits, con saturación a ± 512 (equivalente a ± 2 m/s): error +0,006 %.

El uso de pulsos STEP reales (en lugar de integrar u) hace que la estimación sea coherente con el movimiento físico real del motor, eliminando la acumulación de error de integración presente en el módulo `simple_integrator`, que en este diseño se reserva para generar únicamente la consigna `x2_cmd` durante el arranque en frío.

3.7.12. Módulo `lqr_system_fsm`

Implementa la lógica de la máquina de estados de alto nivel que secuencia las fases de arranque y control del sistema (*homing*, centrado, asentamiento, espera de ángulo crítico y control LQR activo), descrita en detalle en la Sección 3.6. Genera las señales de gobierno del motor durante las fases previas al control (`step_enable_o`, `step_dir_o`, `period_cycles_o`), la referencia angular capturada al finalizar el asentamiento (`count_offset_o`) y la señal de habilitación del controlador LQR (`lqr_enable_o`).

3.7.13. Módulo `lqr_system_top`

Módulo de nivel superior que instancia y conecta todos los bloques anteriores, mapea los puertos externos a los pines físicos de la Basys3, y resuelve mediante lógica combinacional el arbitraje entre las dos fuentes que pueden gobernar el motor:

- Durante las fases de arranque de la FSM (*homing* y centrado), `step_generator` recibe la dirección y el periodo generados directamente por `lqr_system_fsm`.
- Durante el control LQR activo (`lqr_enable='1'`), `step_generator` recibe en su lugar la dirección y el periodo calculados por `motor_command` a partir de `x2_cmd`.

Esta señal de habilitación `lqr_enable` se emplea además para generar `tick_lqr` (el `tick` de muestreo enmascarado, que solo llega a `lqr_controller` y `simple_integrator` cuando el control está activo) y para resetear sincronamente el contador de `cart_estimator` entre activaciones sucesivas del LQR. El módulo expone también las señales de depuración `tick_1ms`, `state_dbg`, `lqr_enable_dbg` y `still_led_o`, mapeadas a los LEDs de la placa según la Tabla 4.

3.8. Implementación en microcontrolador

Para poder realizar la comparativa cuantitativa frente a la implementación en FPGA, se estudia también una implementación equivalente sobre un microcontrolador dsPIC33FJ32MC202 a 40 MIPS, reutilizando el mismo hardware como planta real (descrito en la Sección 3.2). Se analizan dos implementaciones:

- Controlador PID con compensación (-00), heredado del proyecto de referencia (Rodríguez Criado et al., 2025).
- Versión que implementaría el mismo algoritmo LQR discreto y la misma aritmética en punto fijo Q8.8 que se utiliza en la FPGA, compilada con optimización -02.

La segunda versión permitirá realizar una comparación más justa entre ambas plataformas, ya que emplea el mismo algoritmo y la misma aritmética optimizada en ambos casos, por lo que las diferencias de respuesta observadas serán debidas a la plataforma en sí.

A continuación se describe cómo se implementan los bloques funcionales análogos a los de VHDL para estas dos versiones.

3.8.1. Lectura del encoder

La lectura del ángulo del encoder se realiza con la ayuda de un módulo periférico de interfaz de encoder en cuadratura (QEI) integrado en el microcontrolador, que decodifica en hardware las señales A y B sin utilizar la CPU. En la implementación en FPGA, la decodificación de cuadratura se realiza mediante lógica dedicada.

3.8.2. Temporizador del lazo de control

Para el microcontrolador se empleó la misma frecuencia de muestreo, ejecutándose el lazo de control de forma periódica cada 1 ms, de forma análoga al módulo `timer_1ms.vhd` empleado en la FPGA. En cada periodo se generaba una interrupción en la que se ejecutaban de manera secuencial la lectura del encoder, el cálculo de la ley de control y la actualización de los pulsos STEP/DIR; estas mismas operaciones se realizaban de forma paralela en la FPGA.

3.8.3. Generación de pulsos STEP/DIR

La generación de las señales de pulso STEP y dirección DIR se realiza mediante instrucciones ejecutadas por la CPU dentro de la rutina de interrupción del lazo de control en cada periodo de muestreo. Se utilizó la misma resolución de micropasos (1/8) que en la implementación en FPGA. El módulo `step_generator.vhd` opera de forma dedicada y en paralelo al resto del sistema en la FPGA, mientras que en el microcontrolador la generación de los STEP compite por tiempo de CPU con el resto de tareas del lazo de control, convirtiéndose esto en una limitación del diseño.

3.8.4. Cálculo de la ley de control

Para la versión heredada del proyecto de referencia, se implementa un PID con compensación, calculado en aritmética de coma flotante de 64 bits (`double`) sin optimización de compilación (`-O0`).

Para la versión análoga al LQR diseñado en este TFG, se realizaría la misma operación $u_k = -Kx_k$ que se hace en `lqr_controller.vhd`, solo que en vez de realizarse de forma paralela, se calcularía de forma secuencial, instrucción a instrucción.

Capítulo 4 Resultados

4.1. Validación mediante simulación

En esta sección se busca mostrar los resultados obtenidos en la simulación que se realizó previamente a la implementación en hardware con la planta real. Conviene señalar que estas simulaciones validan el controlador frente al mismo modelo lineal empleado en su diseño, por lo que constituyen una comprobación de consistencia más que una validación independiente; la validación de referencia es la realizada sobre la planta física, descrita en la Sección 4.2.4. Simular el sistema no lineal completo, en lugar del modelo linealizado, se plantea como línea de trabajo futuro.

4.1.1. Simulación MATLAB del sistema en lazo cerrado

Una vez se obtuvieron los parámetros físicos de la planta se realizó la simulación del sistema en lazo cerrado con el objetivo de seleccionar la matriz de pesos Q más adecuada para el controlador. El proceso de diseño se estructuró en dos etapas: primero se determinó a qué variable de estado debía asignarse mayor peso, y posteriormente se refinó la elección entre distintas configuraciones que priorizaban dicha variable.

Primera etapa: selección de la variable de mayor peso. Se compararon cuatro configuraciones de Q asignando sucesivamente mayor peso a cada una de las cuatro variables de estado, manteniendo los demás pesos en un valor de referencia:

$$\begin{aligned} Q_{\text{base}} &= \text{diag}(10, 10, 10, 1) \\ Q_{\theta} &= \text{diag}(10, 10, 100, 1) \\ Q_x &= \text{diag}(100, 10, 10, 1) \\ Q_{\dot{\theta}} &= \text{diag}(10, 10, 10, 100) \end{aligned} \tag{4.1}$$

La condición inicial empleada fue $\mathbf{x}_0 = [0, 0, \deg 2rad(15), 0]^T$ ($\theta_0 = 15^\circ$, caso más exigente dentro de la región de activación del controlador). La respuesta obtenida se muestra en la Figura 18.

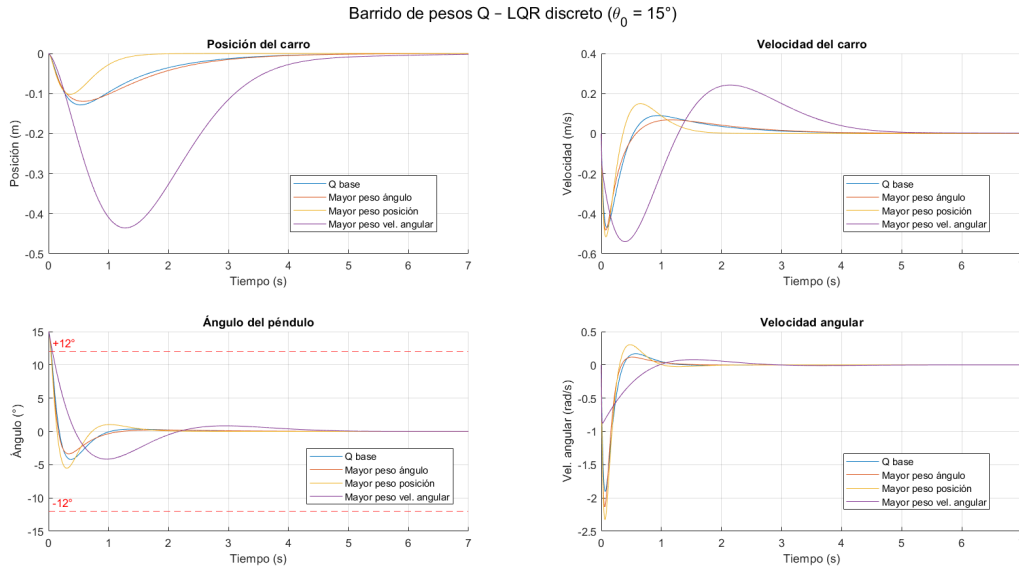


Figura 18: Efecto de aumentar el peso de cada variable de estado ($\theta_0 = 15^\circ$).

Como era de esperar, la mejor respuesta en términos de velocidad de convergencia del ángulo se obtiene cuando la variable $x_3 = \theta$ tiene mayor peso respecto al resto. Un peso elevado sobre θ penaliza fuertemente cualquier desviación angular, lo que se traduce en una acción de control más agresiva sobre el péndulo y un tiempo de establecimiento menor. Esta configuración es coherente con el objetivo principal del controlador: mantener el péndulo en la vertical.

Segunda etapa: ajuste de los pesos con x_3 prioritaria. Una vez determinado que x_3 debe recibir el mayor peso, se compararon cuatro configuraciones que mantienen ese criterio variando los pesos de posición y velocidad del carro:

$$Q_1 = \text{diag}(10, 10, 100, 1)$$

$$Q_2 = \text{diag}(10, 10, 200, 1)$$

$$Q_3 = \text{diag}(20, 20, 200, 1)$$

$$Q_4 = \text{diag}(20, 20, 100, 1)$$

(4.2)

La Figura 19 muestra la respuesta de los cuatro estados para $\theta_0 = 15^\circ$.

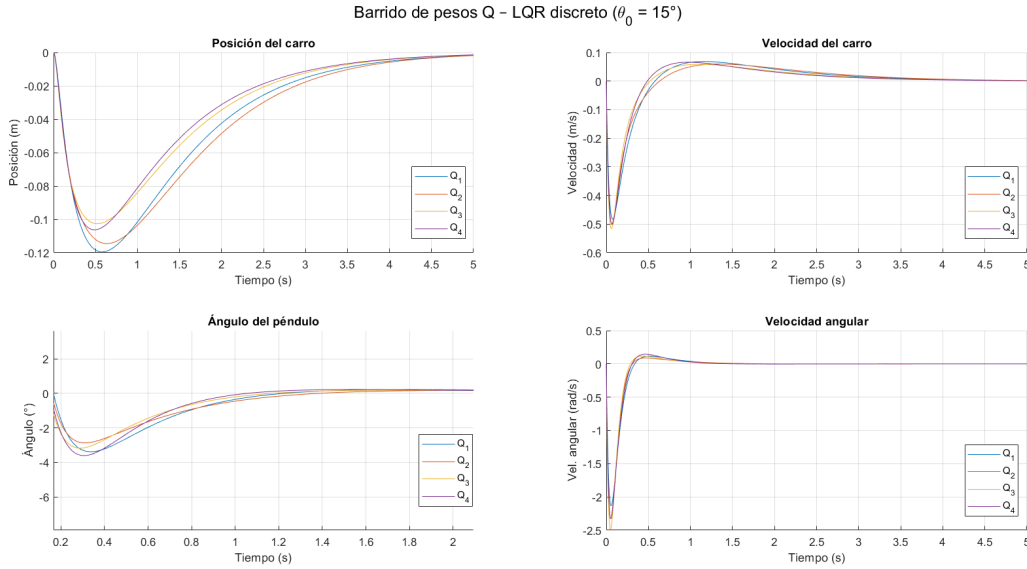


Figura 19: Barrido de pesos Q con prioridad sobre x_3 ($\theta_0 = 15^\circ$).

Para evaluar cuantitativamente qué configuración es más adecuada para el sistema físico, se calcularon el tiempo de establecimiento al 2 % del ángulo y el máximo desplazamiento del carro, recogidos en la Tabla 12.

Tabla 12: Resultados del barrido de Q ($\theta_0 = 15^\circ$).

Configuración	t_s 2 % (s)	$ x_1 _{\text{máx}}$ (m)
Q_1	1,022	0,1195
Q_2	1,091	0,1145
Q_3	0,943	0,1026
Q_4	0,892	0,1063

Las configuraciones Q_1 , Q_3 y Q_4 presentan los menores tiempos de establecimiento y se seleccionaron para un análisis adicional de rangos. En el sistema físico real, la velocidad máxima que el motor paso a paso NEMA17 puede imprimir al carro está limitada a $\dot{x}_{\text{máx}} = 0,5 \text{ m/s}$. Saturar la señal de velocidad de referencia implica que el motor no puede ejecutar la orden del controlador, lo que provoca un fallo sistemático del lazo de control. Por ello se analizó el rango máximo de x_2 para estas tres opciones, recogido en la Tabla 13.

Tabla 13: Rango de velocidad del carro para las configuraciones finalistas ($\theta_0 = 15^\circ$).

Configuración	$ x_2 _{\text{máx}}$ (m/s)	Saturación
Q_1	0,4825	NO
Q_3	0,5165	SI
Q_4	0,5015	SI

Tras analizar los rangos, la decisión final es utilizar la configuración $Q_1 = \text{diag}(10, 10, 100, 1)$, ya que es la única que no satura la velocidad del carro dentro del límite físico del actuador. Aunque Q_3 y Q_4 ofrecen tiempos de establecimiento ligeramente menores, exigirían velocidades superiores a las mecánicamente alcanzables, lo que provocaría pérdida de paso y fallo del control.

Barrido de ángulos iniciales. Con la matriz Q_1 definitiva se realizó un barrido de condiciones iniciales $\theta_0 \in \{3^\circ, 5^\circ, 8^\circ, 10^\circ, 15^\circ\}$ para caracterizar la región de operación del controlador. La respuesta obtenida se muestra en la Figura 20.

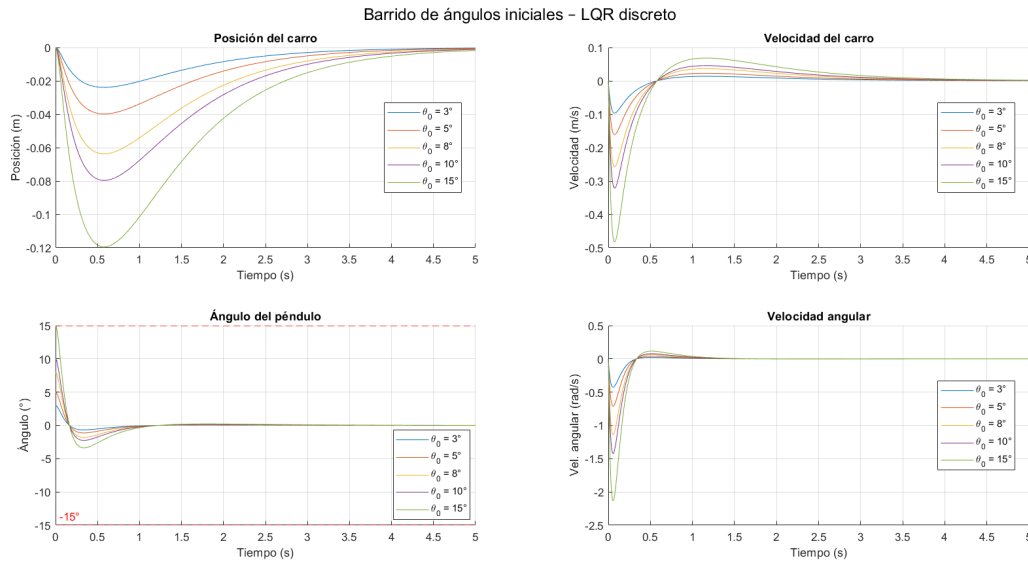


Figura 20: Barrido de ángulos iniciales con Q_1 .

El controlador estabiliza el sistema para todos los ángulos ensayados. Las líneas de $\pm 15^\circ$ marcadas en la figura corresponden al umbral de activación del LQR en la máquina de

estados del sistema: el controlador solo se activa cuando $|\theta| \leq 15^\circ$, lo que es coherente con los límites de validez de la linealización y con el desplazamiento máximo de carro observado en simulación.

Comparación entre respuesta continua y discreta. Por último, se comparó la respuesta del sistema en tiempo continuo con la del modelo discreto (ZOH, $T_s = 1$ ms) para la condición inicial $\theta_0 = 10^\circ$. Las simulaciones se realizaron en Simulink y los resultados se muestran respectivamente en la Figura 21 y Figura 22 también se muestra la fuerza de control para ambos sistemas en las Figuras 23 y 24 .

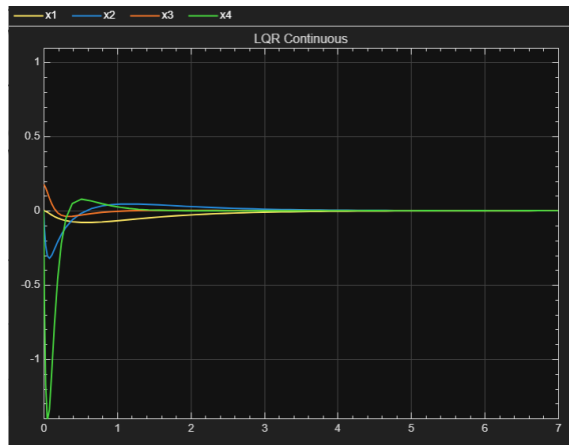


Figura 21: Respuesta continua del sistema en lazo cerrado ($\theta_0 = 10^\circ$).

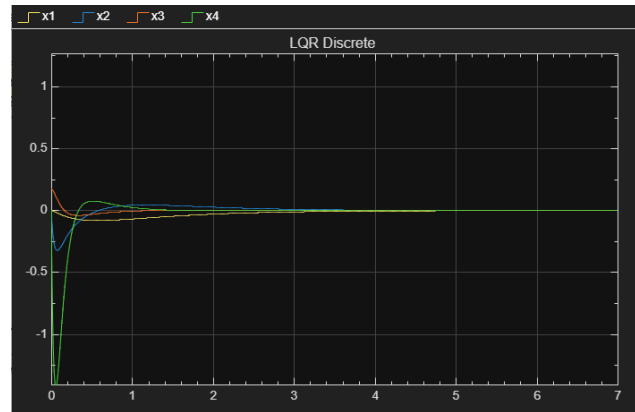


Figura 22: Respuesta discreta del sistema en lazo cerrado ($\theta_0 = 10^\circ$).

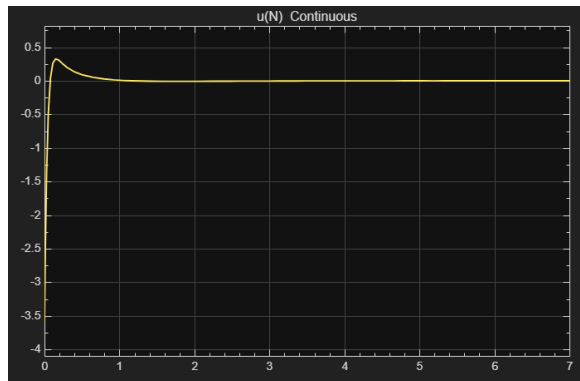


Figura 23: Fuerza de mando u sistema continuo ($\theta_0 = 10^\circ$).

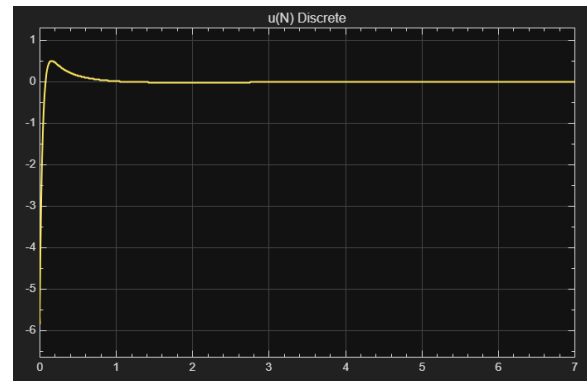


Figura 24: Fuerza de mando u sistema discreto ($\theta_0 = 10^\circ$).

La discretización con $T_s = 1\text{ ms}$ no introduce un error significativo respecto al modelo continuo: ambas respuestas son prácticamente indistinguibles, lo que valida la elección del periodo de muestreo y confirma que el paso de discretización ZOH preserva fielmente la dinámica del controlador diseñado en tiempo continuo.

Se realizaron más simulaciones para distintos vectores de condiciones iniciales, cuyas respuestas se pueden observar en el Anexo C. Los scripts de MATLAB empleados para realizar estas simulaciones se puede encontrar en el repositorio de GitHub del proyecto (<https://github.com/alopezgarciarom/lqr-inverted-pendulum-fpga>), sin embargo, en el Anexo B se puede encontrar una síntesis de las partes más relevantes del código.

4.1.2. Simulación funcional VHDL (testbench)

Con la simulación de MATLAB completada se pasó a realizar la simulación de cada bloque con su testbench, en esta sección únicamente se mostrarán los resultados de la simulación de los bloques más relevantes de la implementación en VHDL. El resto de simulaciones de cada bloque se encuentra en el Anexo E.

4.1.3. `tb_lqr_system_fsm`

La primera simulación que se muestra en la Figura 25 corresponde al testbench de la máquina de estados (`tb_lqr_system_fsm`). Se verifica que el sistema funciona correctamente, la inicialización parte del estado `S_HOMING` tras el reset, con el fin de carrera activado la FSM transiciona a `S_BACKING` y tras completar el retroceso de los pasos establecidos pasa a `S_WAIT_SETTLE`, una vez el temporizador finaliza y el péndulo ha permanecido quieto un segundo se activa el estado `S_WAIT_CRITICAL` y una vez el número de cuentas está dentro del ángulo crítico se activa el control con `S_LQR_ACTIVE` del cual sale por rebasar el límite de pasos.

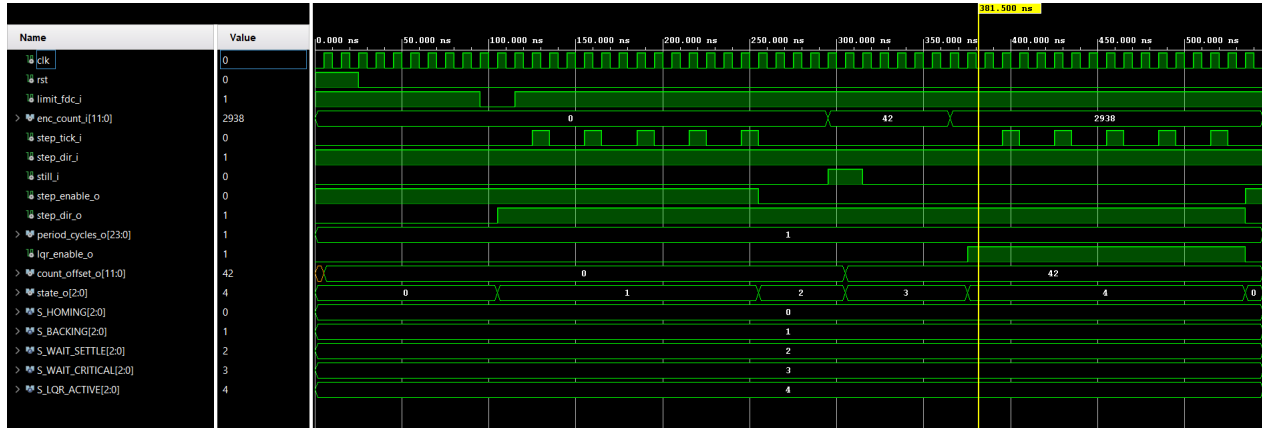


Figura 25: Simulación funcional del módulo `lqr_system_fsm`.

4.1.4. `lqr_controller`

En la Figura 26 se muestra la simulación del testbench del módulo `lqr_controller`. La simulación permite verificar que la señal de control u se calcula correctamente en el formato deseado a partir del vector de estados $\mathbf{x}$.

Para el instante capturado en simulación con los estados $x_1 = x_2 = x_4 = 0$ y $x_3 = -45$ (en formato Q8.8), la ley de control produce:

$$u = -(K_1x_1 + K_2x_2 + K_3x_3 + K_4x_4) = -\left(\frac{5204 \times (-45)}{256}\right) = -\left[\frac{-234180}{256}\right] = -(-915) = 915 \quad (4.3)$$

valor que coincide con la salida $u[15:0] = 915$ observada en la Figura 26.

En la Figura también se aprecia que el vector de estados se actualiza en cada periodo de muestreo y que la señal de mando u se actualiza en el segundo flanco de reloj después del `tick` de muestreo, ya que necesita un ciclo de reloj para realizar las multiplicaciones parciales y otro para la suma de la ley de control.

Los dos casos de saturación se verifican en los dos últimos `tick` de la simulación. Con los estados en $x_1 = x_2 = x_3 = x_4 = -32768$ en Q8.8, la suma acumulada supera el rango positivo y u satura a $+32767$. De forma simétrica, con todos los estados en el valor máximo ($x_1 = x_2 = x_3 = x_4 = 32767$), u satura a -32768 . Esto demuestra el correcto funcionamiento del saturador de la suma que limita la salida al rango $[-32768, 32767]$ correspondiente a un entero con signo de 16 bits.

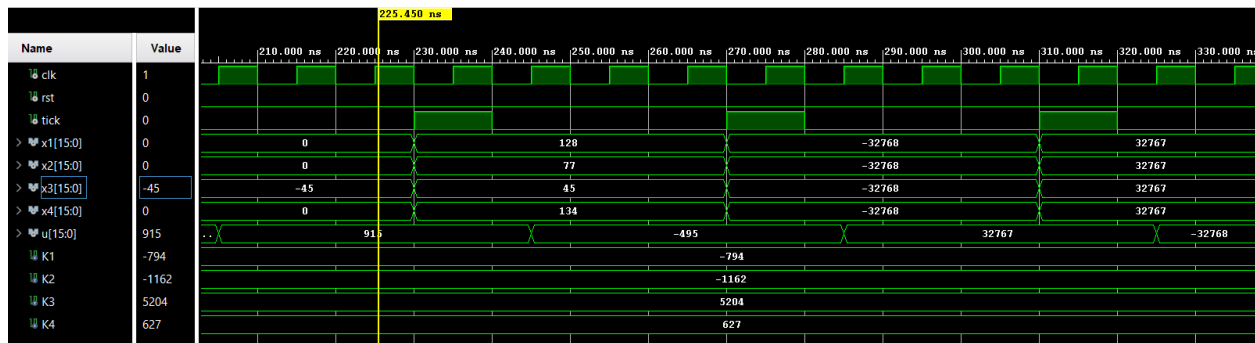


Figura 26: Simulación funcional del módulo `lqr_controller`.

4.1.5. motor_command

Por último, se muestra en la Figura 27 la simulación del módulo `motor_command`. Una vez recibida la señal de control u y tras realizar la integración en el bloque `simple_integrator`, el módulo genera la señal de periodo, dirección y enable en cada periodo de muestreo para el bloque `step_generator`. La simulación permite verificar que el valor de periodo es correcto y que la señal de enable se activa solamente cuando el valor de la señal de mando es distinta de cero.

La Figura 27 muestra la saturación de `period` al valor mínimo `PER_MIN`= 6250 ciclos con $x_2_cmd = 205$ (0,8 m/s en Q8.8). La rampa desciende en pasos de $\Delta T = 100$ ciclos por tick hasta alcanzar el límite inferior, momento en que `quotient`= 6244 < `PER_MIN`, confirmando que el saturador impide sobrepasar la frecuencia máxima de paso. Al final de la captura, x_2_cmd cambia a 51 y `period` vuelve a `PER_ARR`= 50000 al pasar por el estado `LQR_IDLE`, reiniciando la rampa.

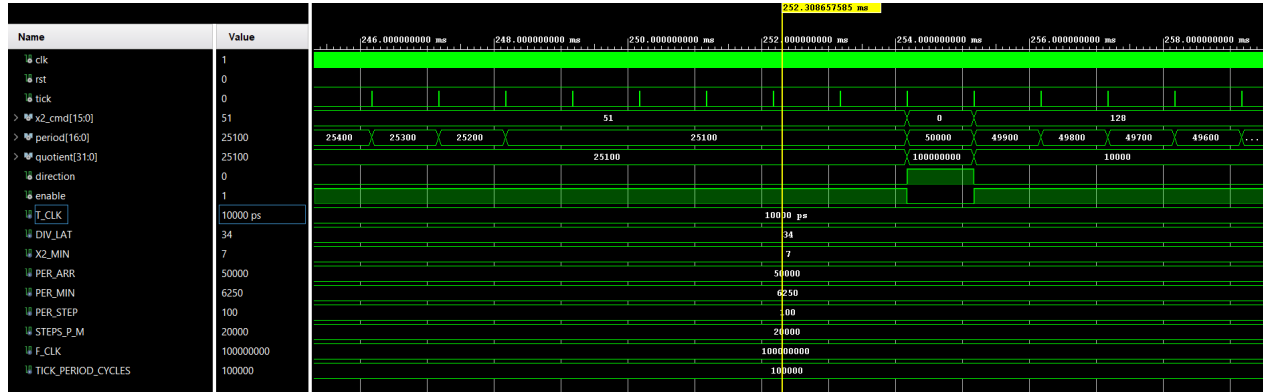


Figura 27: Simulación de motor_command: saturación de period a PER.MIN= 6250 con $x_2_cmd = 205$ y reinicio de rampa al cambiar la referencia de velocidad.

La Figura 28 muestra la convergencia de period al valor del cociente con $x_2_cmd = 51$ (0,2 m/s). La frecuencia de paso objetivo se calcula como:

$$f_{step} = \frac{51 \times 20000}{256} = 3984 \text{ pasos/s} \Rightarrow T = \frac{10^8}{3984} = 25100 \text{ ciclos} \quad (4.4)$$

La rampa desciende desde PER_ARR= 50000 hasta estabilizarse exactamente en quotient= 25100, verificando que el divisor calcula correctamente el periodo correspondiente a la velocidad de referencia. A continuación, x_2_cmd cambia a 0 (LQR.IDLE, enable= 0) y posteriormente a 128 (0,5 m/s), arrancando una nueva rampa hacia quotient= 100000 ciclos.

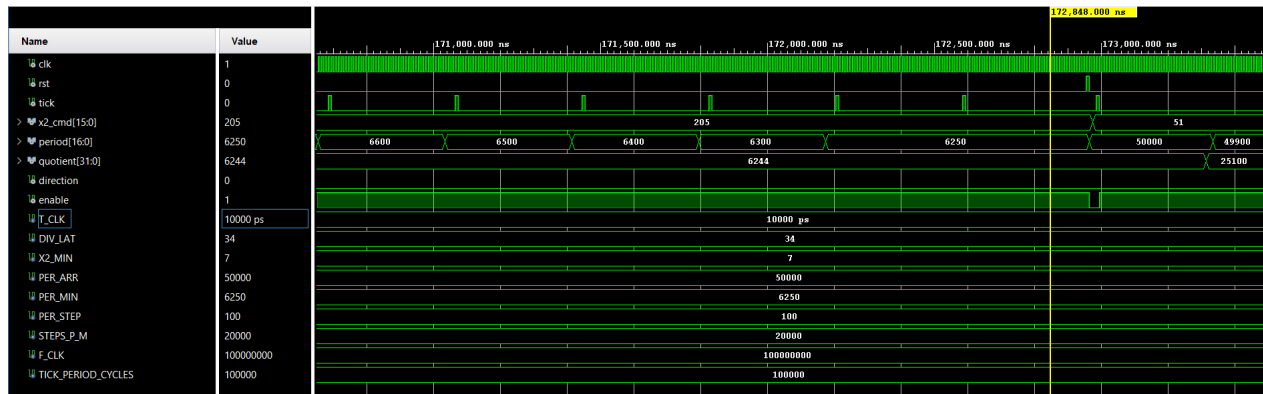


Figura 28: Simulación de motor_command: convergencia de period al cociente $T = 25100$ ciclos con $x_2_cmd = 51$, transición por LQR.IDLE y arranque de nueva rampa con $x_2_cmd = 128$.

4.2. Resultados de implementación en FPGA

4.2.1. Síntesis y uso de recursos (Vivado)

En primer lugar, se realizó la síntesis del diseño completo en Vivado. El uso de recursos estimado por Vivado tras la síntesis se muestra en la Tabla 14.

Tabla 14: Utilización de recursos tras síntesis (`synth_1`), sin `div_gen_0`.

Recurso	Utilizado	Disponible	Ocupación (%)
Slice LUTs	877	20 800	4,22
Slice Registers	280	41 600	0,67
DSP48E1	9	90	10,00
Block RAM	0	50	0,00
Bonded IOB	14	106	13,21

Cabe destacar que los 9 bloques DSP48E1 son utilizados por el diseño propio, no por `div_gen_0`, que implementa su aritmética de división Radix-2 exclusivamente en lógica distribuida y su uso estimado de recursos se muestra en la Tabla 15. Los DSPs de la Tabla 14 corresponden a los cuatro multiplicadores Q8.8 del módulo `lqr_controller` y a las multiplicaciones de conversión de `cart_estimator` y `simple_integrator`.

Tabla 15: Utilización de recursos del IP Core `div_gen_0` (síntesis independiente).

Recurso	Utilizado	Disponible	Ocupación (%)
Slice LUTs	1 185	20 800	5,70
Slice Registers	3 269	41 600	7,86
DSP48E1	0	90	0,00
Block RAM	0	50	0,00

Los datos reales de uso de recursos de la FPGA se recoge en la Tabla 16, los datos han sido obtenidos del report de Vivado tras ejecutar la implementación del diseño completo.

Tabla 16: Utilización de recursos de la Basys3 (Artix-7 XC7A35T).

Recurso	Utilizado	Disponible	Ocupación (%)
Slice LUTs	1 936	20 800	9,31
LUT como lógica	1 935	20 800	9,30
LUT como memoria	1	9 600	0,01
Slice Registers (FF)	2 310	41 600	5,55
DSP48E1	9	90	10,00
Block RAM	0	50	0,00
Bonded IOB	14	106	13,21
BUFG	1	32	3,13

Como se puede observar en la Tabla 16, el diseño hace un uso bajo de recursos, lo que deja espacio para futuras extensiones del sistema. No se utilizan bloques de memoria BRAM, dado que todas las estructuras de datos del sistema caben en registros distribuidos. Por otro lado, los 14 IOBs utilizados corresponden a las señales de entrada y salida del sistema, : `clk`, `rst`, `enc_A`, `enc_B`, `limit_i`, `STEP`, `DIR`, `ENABLE` y los LEDs de depuración (señales de estado, el reloj y la señal `still_led_o`). Además, el diseño opera en un único dominio de reloj a 100 MHz, por lo que solo se requiere un BUFG para distribuir la señal de reloj a todos los módulos, eliminando la necesidad de sincronización entre dominios de reloj.

4.2.2. Análisis energético

Tras la implementación Vivado proporciona un informe con la estimación de consumo energético del diseño, el resultado de este análisis se muestra en la Tabla 17. Esta potencia es estimada por lo que se interpreta como orden de magnitud aproximado y no exacto.

El consumo total estimado es de 121 mW, de los cuales 72 mW son consumo estático que no depende del diseño y 50 mW son potencia dinámica. La temperatura de juntura estimada es de 25,6 °C, con márgenes amplios de operación térmica (74,4 °C).

Tabla 17: Consumo estimado de la implementación (Power Report, `impl_1`).

Componente	Potencia (W)	Porcentaje (%)
Potencia dinámica total	0,050	41
Lógica	0,017	33
Señales	0,012	25
Reloj	0,009	18
DSP	0,008	16
E/S	0,004	8
Potencia estática	0,072	59
Total on-chip	0,121	100

Como se ha comentado, estos datos son estimaciones de Vivado. Para obtener el consumo real se utilizó un circuito auxiliar y se sustituyó la alimentación de la FPGA por USB desde el PC por una fuente de alimentación de laboratorio a 5 V. El circuito empleado consistió en una resistencia *shunt* de $1\ \Omega$ conectada en serie con la fuente, permitiendo calcular la corriente consumida a partir de la caída de tensión en la resistencia y, con ello, la potencia total consumida por la FPGA.

La caída de tensión medida en la resistencia de $1\ \Omega$ fue de 0,04 V, por lo que, aplicando la ley de Ohm, la corriente consumida por la FPGA es de 0,04 A. Con el osciloscopio se midió directamente la tensión de entrada a la FPGA tras la caída en la resistencia, obteniéndose un valor de 4,78 V. A partir de estos datos, la potencia consumida por la FPGA resulta:

$$P = V \cdot I = 4,78 \times 0,04 = 0,1928\ \text{W} \quad (4.5)$$

Esta cifra es mayor que la estimada por Vivado, aunque del mismo orden de magnitud.

4.2.3. Latencia del lazo de control

La latencia total del lazo se define como el tiempo transcurrido desde que se emite el pulso de muestreo (`tick`) hasta que la señal de mando del motor (`STEP`) se activa, tras haber calculado la velocidad en `motor_command` con la señal u actualizada.

Tabla 18: Desglose de latencia del lazo de control.

Etapa	Ciclos	Tiempo (ns)
mult_q8_8 $\times 4$ (paralelo)	1	10
sumador_sat (habilitado con tick_d1)	1	10
div_gen_0 en motor_command	34	340
Total	36	360

Como se aprecia en la Tabla 18, la latencia total es de 36 ciclos de reloj, es decir 360 ns a 100 MHz. Teniendo en cuenta que el periodo de muestreo seleccionado es de 1 ms, la latencia es despreciable, siendo una fracción de $360 \text{ ns} / 1 \text{ ms} = 0,036 \%$ y encontrándose muy por debajo del límite habitual de periodo de muestreo recomendado en implementaciones de control digital.

Por otra parte, el análisis de temporización de Vivado tras la implementación se muestra en la Tabla 19. Este informe confirma que el diseño funciona correctamente a la frecuencia de reloj de 100 MHz, con un Worst Negative Slack (WNS) positivo de +0,148 ns cumpliendo todos los requisitos de temporización del sistema.

Tabla 19: Resumen del análisis de temporización (impl_1).

Parámetro	Valor
Worst Negative Slack (WNS)	+0,148 ns
Total Negative Slack (TNS)	0,000 ns
Worst Hold Slack (WHS)	+0,049 ns
Endpoints fallidos	0 / 2 902

4.2.4. Validación experimental en hardware

Una vez finalizada la fase de simulaciones, se evaluó el controlador en la planta real. Los resultados obtenidos se resumen a continuación:

- **Máquina de estados del sistema.** Las fases de `S_HOMING` y `S_BACKING` se ejecutan correctamente. El temporizador de espera de `S_WAIT_SETTLE` también ofrece los resultados esperados, así como el LED `still_led`, que se enciende cuando el encoder lleva 1 s sin variar su medida.

- **Desbordamiento del contador del encoder.** Se observa un problema de desbordamiento en el contador de `encoder_decoder`: al disponer únicamente de ± 2047 cuentas, el desplazamiento de referencia de 180° aplicado sobre `count_offset_reg` puede superar dicho rango según la posición absoluta en la que se capture la referencia angular, provocando un desbordamiento en complemento a dos que rompe la continuidad de la medida. En la práctica, esto se traduce en que el controlador solo se activa de forma fiable cuando el péndulo se aproxima al ángulo crítico en sentido antihorario. Esta limitación, junto con su solución propuesta, se retoma en la Sección de Trabajo Futuro.
- **Estabilización del péndulo.** Una vez activado el controlador, la rampa de velocidad se ejecuta correctamente y el sistema estabiliza el péndulo en menos de un segundo, manteniéndolo en equilibrio durante breves segundos. Se observa, no obstante, una tendencia a caer hacia el lado izquierdo. Una causa probable, además de un posible desalineamiento inicial entre el péndulo y la vertical de referencia, es la limitación de modelado descrita en la Sección 3.7.5: el LQR se diseñó para una planta con entrada de fuerza, pero el actuador real regula velocidad, por lo que la planta controlada en hardware difiere de la modelada.
- **Tiempo de establecimiento.** No es posible determinarlo con claridad, ya que la estabilización solo se produce correctamente una vez que el motor completa la rampa de aceleración.
- **Límites de carrera.** El límite del lado izquierdo, implementado mediante el pulsador de fin de carrera, funciona correctamente. El límite derecho, en cambio, falla en ocasiones: a altas velocidades el motor pierde pasos, perdiendo con ello la referencia interna de posición del carro.
- **Condición de salida angular.** La condición de salida del controlador cuando el ángulo supera los $\pm 30^\circ$ se cumple con éxito.

4.3. Resultados de implementación en microcontrolador

4.3.1. Uso de recursos

Para la versión PID heredada del proyecto de referencia (Rodríguez Criado et al., 2025), el archivo de mapeo de memoria (.map) generado tras la compilación indica una ocupación

real de 17,3 KB de memoria Flash (52 % de la disponible en el dsPIC33FJ32MC202) y 606 B de RAM (30 % de la disponible).

Para la versión análoga del controlador LQR, al no haberse implementado ni compilado dentro de este Trabajo Fin de Grado, se ofrece una estimación de ingeniería de entre 4 y 8 KB de Flash y entre 200 y 300 B de RAM, algo inferior a la versión PID por tratarse de una ley de control más simple (sin términos integral/derivativo ni lógica de anti-windup) y con menos código de soporte para aritmética en coma flotante.

4.3.2. Análisis energético

No existe medición real del consumo de potencia eléctrica del dsPIC33FJ32MC202 ejecutando el control por no haber llegado a completar la fase de validación con la planta real para esta plataforma. Por lo que se ofrece una estimación basada en el consumo típico de datasheet del dispositivo, de aproximadamente 178 mW en operación activa a 40 MIPS, valor equiparable en orden de magnitud entre las dos versiones, ya que el consumo del núcleo depende principalmente de la frecuencia de reloj y de los periféricos activos, y no varía de manera significativa por representar los cálculos en coma flotante o en punto fijo.

4.3.3. Latencia del lazo de control

Al no disponer de medición directa, se presentan dos estimaciones de ingeniería, correspondientes a las dos versiones descritas en la Sección 3.8:

- PID (coma flotante de 64 bits, -00): entre 40 y 50 μ s, estimación conservadora dado el coste elevado de las operaciones en coma flotante por software en un núcleo sin unidad de coma flotante dedicada.
- LQR (punto fijo Q8.8, -02): entre 1 y 3 μ s, estimación basada en el reducido número de operaciones (4 multiplicaciones y 3 sumas en enteros) ejecutadas de forma optimizada.

Se puede observar que los dos valores son muy inferiores al periodo de muestreo de 1 ms, por lo que en ambos casos el microcontrolador cumpliría el requisito de tiempo real.

4.3.4. Validación experimental en hardware

En el caso del microcontrolador, no se llegó a llevar a cabo la validación sobre hardware real en este Trabajo Fin de Grado. Sin embargo, los datos estimados en las secciones anteriores servirán para realizar la comparativa en la Sección de 4.

4.4. Comparativa FPGA vs. microcontrolador

4.4.1. Latencia y frecuencia de actualización

La FPGA presenta una latencia de cómputo real de 360 ns, frente a las estimaciones de 40-50 μ s para la versión PID y de 1-3 μ s para la versión LQR en el dsPIC33FJ32MC202. Esto supone una diferencia de aproximadamente 100 veces frente a la primera versión, pero de solo 3 a 8 veces frente a la segunda versión (LQR equivalente). Gran parte de la ventaja aparente de la FPGA frente al primer caso procede de estar comparando con una implementación en coma flotante sin optimizar, no de una ventaja intrínseca de la plataforma.

4.4.2. Recursos

Tabla 20: Comparativa de recursos entre FPGA y microcontrolador.

	LUT	FF	DSP	BRAM	Flash	RAM
FPGA	1936 (9 %)	2310 (5,6 %)	9 (10 %)	0 (0 %)	—	—
dsPIC (PID, real)	—	—	—	—	17,3 KB (52 %)	606 B (30 %)
dsPIC (LQR, est.)	—	—	—	—	4-8 KB (~12-25 %)	200-300 B (~10-15 %)

En la Tabla 20, se observa un consumo de recursos reducido para todas las categorías del LQR sobre FPGA, así como también una estimación baja de uso de recursos en el dsPIC con el diseño LQR. En cambio, los valores reales para el PID sobre microcontrolador son más elevados.

4.4.3. Complejidad de desarrollo

El desarrollo en FPGA requiere describir el hardware en VHDL, verificar mediante simulación funcional y sintetizar el diseño, lo que implica una curva de aprendizaje y un tiempo de desarrollo mayores. El desarrollo en microcontrolador se realiza en lenguaje C sobre un

entorno de desarrollo integrado convencional (MPLAB), con un ciclo de compilación y depuración más rápido y una menor complejidad de diseño.

4.4.4. Análisis comparativo global

Tabla 21: Comparativa global entre FPGA y microcontrolador.

Métrica	FPGA Artix-7	dsPIC: PID	dsPIC: LQR
Algoritmo	LQR discreto	PID + compensación	LQR discreto
Aritmética	Punto fijo Q8.8	Coma flotante (double 64 bits), -00	Punto fijo Q8.8, -02
Plataforma / reloj	XC7A35T @ 100 MHz	dsPIC33FJ32MC202 @ 40 MIPS	
Latencia de cómputo	360 ns [real]	40-50 μ s [est.]	1-3 μ s [est.]
Determinismo	Fijo por hardware, sin <i>jitter</i>	Bajo (ramas + IRQ motor/UART)	Mejor, pero secuencial
Paralelismo	Total	No (todo en la CPU)	
Consumo	193 mW [real]	~178 mW [est.]	
Recursos	Ver Tabla 20		
Complejidad	Mayor (HDL, síntesis)	Menor (C, MPLAB)	

Además de los resultados ya comentados en las secciones anteriores, de la Tabla 21 se extraen las siguientes ideas adicionales. La ventaja real de la FPGA es su determinismo temporal y su capacidad de ejecución en paralelo, no en un factor de velocidad de 100 veces. En cuanto al consumo energético, ambas plataformas se encuentran en el mismo orden de magnitud: la FPGA es competitiva, pero no representa una gran mejora. Para este problema concreto, además, ambas plataformas cumplen el requisito de tiempo real con margen sobrado (control a 1 kHz), por lo que el cuello de botella real del sistema es el motor paso a paso, y no la capacidad de cómputo de ninguna de las dos plataformas.

4.5. Proceso de depuración y principales dificultades

El proceso de implementación se llevó a cabo de manera iterativa. Durante este desarrollo, se presentaron dificultades no anticipadas. Esta sección describe los hallazgos más relevantes

y las soluciones que se decidieron para cada caso.

4.5.1. Corrección del pipeline en `sumador_sat`

En una primera simulación del módulo `lqr_controller` se detectó un fallo al realizar la suma de los cuatro productos parciales. Tras el análisis de la simulación se pudo observar que la suma se estaba realizando en el mismo `tick` en el que los multiplicadores recibían los nuevos estados de entrada. Sin embargo, los módulos `mult_q8_8` tienen una latencia de un ciclo de reloj, es decir registran su resultado en el flanco siguiente en el que se realizan. Por ello el sumador estaba utilizando los productos del ciclo anterior.

La solución consistió en generar una señal `tick_d1` que retrasa la señal `tick` exactamente 1 ciclo de reloj y conectarla como enable al bloque `sumador_sat` en lugar de `tick`. Logrando así que el cálculo de la señal de control fuera correcto para cada periodo. Este cambio hizo que la latencia total del módulo pasara a ser de 2 ciclos de reloj (20 ns a 100 MHz), lo que no tiene impacto sobre el periodo de muestreo de 1 ms como se ha demostrado en el análisis de latencia.

4.5.2. Sustitución de `double_integrator` por `cart_estimator` y `simple_integrator`

En un principio se implementó un integrador doble para estimar la velocidad y posición del carro a partir de la señal de control u . Sin embargo, tras la puesta en marcha del sistema finalmente se optó por cambiar la implementación por un estimador de posición y velocidad basado en el conteo de pulsos STEP del motor paso a paso, lo cual resultó más robusto frente a errores de cuantización. Cabe señalar que, aun así, la posición y la velocidad del carro no se miden directamente, sino que se estiman contando los pulsos STEP enviados al motor, por lo que, si el motor pierde pasos, esta estimación se desvía del valor real. El integrador doble pasó a ser uno simple en el que se integraba una única vez para conocer la velocidad de control necesaria para generar los pulsos STEP en cada instante de tiempo.

4.5.3. Limitación de frecuencia del motor

Durante el diseño del sistema se estimó mediante simulación en MATLAB que la velocidad máxima del carro partiendo desde reposo con un ángulo inicial de 15° era de 0,4825 m/s. Para verificar si el motor paso a paso era capaz de alcanzar dicha velocidad, se realizaron pruebas experimentales generando pulsos STEP a distintas frecuencias mediante programas de prueba

sencillos. Los resultados mostraron que la frecuencia máxima de arranque desde reposo con microstepping de 1/16 era de aproximadamente 20 kHz.

Con la polea original de 20 dientes y paso de correa GT2 de 2 mm, el desplazamiento lineal por revolución era de $20 \times 2 = 40$ mm/rev, lo que con 3 200 micropasos/rev resultaba en 80 000 micropasos/metro. La frecuencia de 20 kHz se traducía por tanto en una velocidad máxima de $20\,000/80\,000 = 0,25$ m/s, insuficiente para el rango de operación requerido.

La solución adoptada fue sustituir la polea por una de 40 dientes, duplicando el desplazamiento lineal por revolución a 80 mm/rev y reduciendo a la mitad el número de micropasos por metro ($80\,000 \rightarrow 40\,000$ micropasos/m). Con esta configuración, la misma frecuencia máxima de 20 kHz permite alcanzar $20\,000/40\,000 = 0,5$ m/s, cubriendo el rango de velocidad estimado por simulación. Cabe destacar que en estas pruebas se utilizó microstepping de 1/16, pero para el diseño final se adaptó a 1/8 como se ha indicado a lo largo de esta memoria.

4.5.4. Problema de arranque del motor

Durante las primeras pruebas en hardware se identificaron dos problemas relacionados con el arranque del motor que condicionaron el diseño final del sistema.

El primero fue la pérdida de paso por arranque a alta velocidad (*stalling*). El controlador LQR puede exigir en el primer periodo de muestreo una velocidad de consigna elevada que el motor no es capaz de alcanzar desde reposo, ya que el par de arranque del NEMA17 es significativamente inferior al par sostenido. Esto motivó la implementación de la rampa trapezoidal de aceleración en el módulo `motor_command`, descrita en la Sección 3.7, que limita la tasa de cambio de velocidad y garantiza que el motor opera siempre dentro de su curva de par.

El segundo problema, más sutil, fue un bloqueo de arranque por condición inicial (*bootstrap deadlock*). La máquina de estados de `motor_command` permanece en el estado `LQR_IDLE` mientras $|x_2| \leq X2_MIN$, arrancando el movimiento únicamente cuando la consigna de velocidad supera ese umbral. Sin embargo, en el instante de activación del LQR el carro está en reposo, por lo que $x_2 = 0$ y la condición de arranque nunca se satisface: el motor no arranca, el carro no se mueve, x_2 permanece a cero y el sistema queda bloqueado indefinidamente.

La solución adoptada fue introducir el módulo `simple_integrator`, que integra la señal de control u mediante el método de Euler para generar una consigna de velocidad `x2_cmd` que

crece desde cero en cuanto el LQR emite su primera acción de control. Esta señal se proporciona a `motor_command` como referencia de arranque, mientras que el módulo `cart_estimator` sigue proporcionando la estimación real de x_2 basada en pulsos STEP para la ley de control LQR. De este modo se desacopla la consigna de arranque del motor de la estimación de estado, resolviendo el deadlock sin comprometer la precisión del lazo de control.

4.5.5. Ajuste de parámetros rampa de aceleración motor

Para lograr una aceleración óptima en el menor tiempo posible se utilizó un programa sencillo en el que, con la ayuda del SW de la FPGA, se seleccionaba por un lado la frecuencia máxima (máxima velocidad del carro) y los ciclos de reducción del periodo por cada ms en la rampa. El ensayo se realizó con microstepping 1/8, partiendo en todos los casos de una velocidad de arranque de 2 000 micropasos/s.

La Tabla 22 muestra las frecuencias máximas y las reducciones de ciclos ensayadas, así como los ensayos exitosos y los que no lo fueron.

Tabla 22: Ensayo de ajuste de la rampa de aceleración `motor_command`.

Vel. máxima (kHz)	Reducción de periodo (ciclos/ms)			
	150	200	250	300
6	SI	SI	SI	SI
8	SI	SI	SI	SI
10	SI	SI	SI	SI
12	SI	SI	SI/NO	NO
14	NO	NO	NO	NO
16	NO	NO	NO	NO
20	NO	NO	NO	NO
24	NO	NO	NO	NO

Se observa una frontera clara: hasta 10 kHz el motor arranca correctamente en todo el rango de pendientes ensayado, mientras que a partir de 14 kHz el arranque falla independientemente de la agresividad de la rampa. El punto de 12 kHz constituye el límite de operación, con éxito dependiente de la pendiente elegida.

Una vez se obtuvieron las velocidades sin *stalling* y las reducciones de ciclos factibles, se representaron las curvas de aceleración, mostrando la velocidad en función del tiempo para

cada ensayo exitoso. Dicha velocidad se calculó a partir del modelo cinemático del periodo de pulsos STEP, con la relación de 1 600 micropasos/rev (microstepping 1/8) y polea de 40 dientes (80 mm/rev), es decir 20 micropasos/mm. Los resultados se observan en la Figura 29.

Estos resultados sirvieron para seleccionar el compromiso entre las dos variables: como se aprecia en la Figura 29, la combinación de 10 kHz de velocidad máxima con 300 ciclos/ms de reducción de periodo alcanza la mayor velocidad validada como fiable, sin embargo se bajó la reducción de periodo a 200 ciclos/ms para tener más flexibilidad en el módulo `motor_command`.

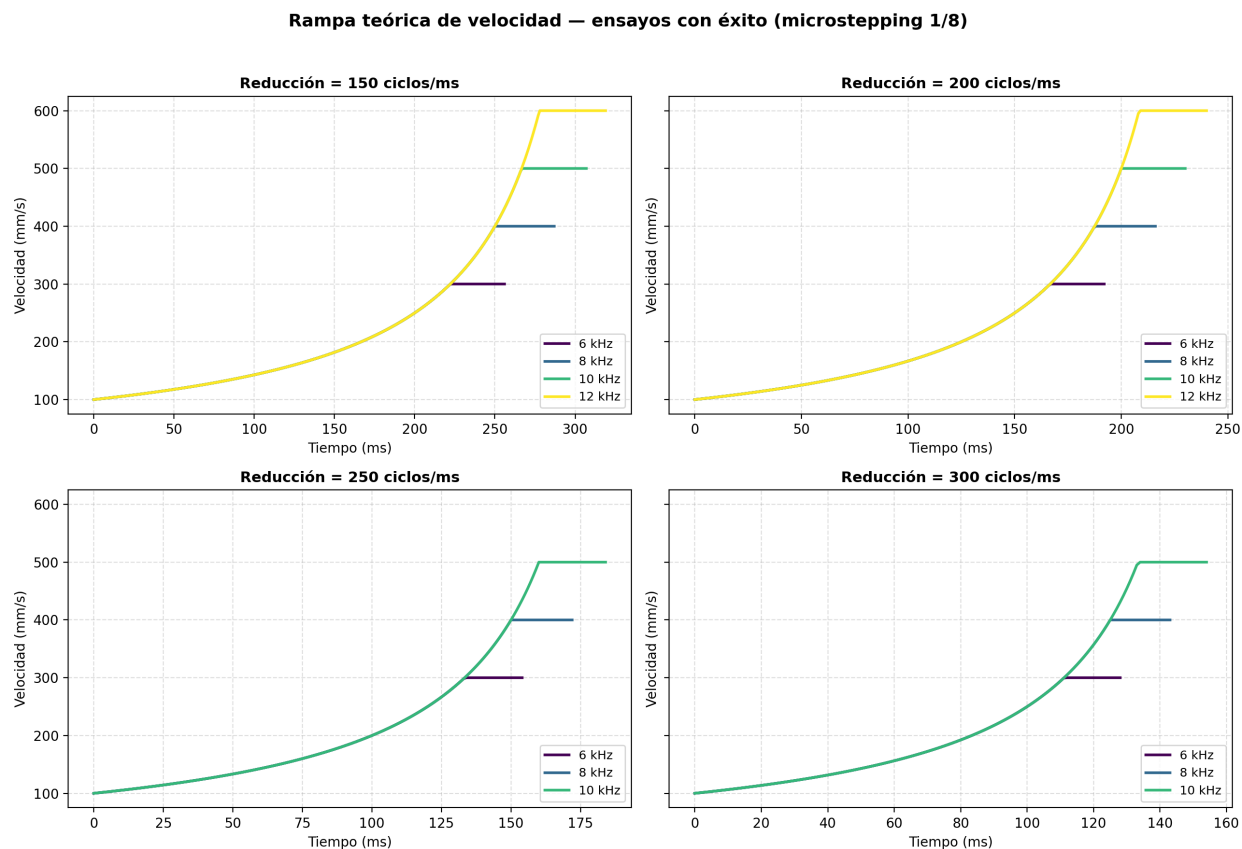


Figura 29: Rampa teórica de velocidad lineal del carro para las combinaciones exitosas del ensayo (microstepping 1/8).

4.5.6. Experimento de control de subida (*swing-up*)

Aunque la elevación del péndulo desde el reposo no era uno de los objetivos principales del sistema, sino uno de los requisitos propuestos para iniciar el control, se realizó un experimento adicional para intentar incluirlo en el diseño final, reutilizando un código de *swing-up* ya

existente, proporcionado por el director de este Trabajo Fin de Grado. La estrategia empleada es un bombeo de energía realimentado de tipo Åström-Furuta: el carro se empuja siempre en el sentido dado por el signo de $\dot{\theta} \cos \theta$, sin necesidad de conocer los parámetros físicos del sistema (masas, longitud o inercia del péndulo) (Åström & Furuta, 1996). Esto equivale al principio con el que se impulsa un columpio, empujando siempre en el sentido que refuerza el balanceo: cuando el péndulo se aleja de la vertical superior con velocidad angular en el mismo sentido en el que ya se está inclinando hacia ese lado, se empuja el carro para reforzar ese movimiento, inyectando energía mecánica en cada ciclo hasta que el péndulo alcanza la región de ángulo crítico donde se activa el controlador LQR.

El experimento se realizó para las cuatro velocidades máximas que habían superado el ensayo de la rampa de aceleración de forma fiable (6, 8, 10 y 12 kHz), combinadas con las cuatro reducciones de periodo ensayadas anteriormente (150, 200, 250 y 300 ciclos/ms), comprobando en cada caso si el péndulo alcanzaba correctamente la región de ángulo crítico mediante el bombeo de energía. La Tabla 23 resume los resultados.

Tabla 23: Ensayo del experimento de *swing-up* por bombeo de energía.

Vel. máxima (kHz)	Reducción de periodo (ciclos/ms)			
	150	200	250	300
6	SI	SI	SI	SI
8	SI*	SI	SI	SI
10	SI*	SI	SI	SI
12	SI*	SI	SI	SI

*Se observó un comportamiento inicialmente compatible con pérdida de paso del motor (*stalling*) en las combinaciones de 8, 10 y 12 kHz con 150 ciclos/ms de reducción de periodo. Tras su análisis, se determinó que no se trataba de un atasco real del motor, sino del ruido mecánico generado por los cambios de sentido a alta frecuencia propios de esta reducción de periodo más agresiva.

El experimento resultó exitoso para las combinaciones ensayadas: en todos los casos el péndulo alcanzó la región de ángulo crítico mediante el bombeo de energía, momento en el que se realizaría la transición al controlador LQR descrito en el resto de esta memoria.

Capítulo 5 Conclusiones y Trabajo Futuro

5.1. Conclusiones

5.1.1. Cumplimiento de objetivos e implementación en FPGA

La mayoría de los objetivos específicos planteados se han cumplido de manera satisfactoria. Se logró diseñar y validar mediante simulación el regulador lineal cuadrático discreto para la planta seleccionada y se implementó de forma exitosa en FPGA en aritmética de punto fijo Q8.8, cumpliendo los requisitos de temporización establecidos con margen positivo y ocupando una fracción reducida de los recursos disponibles en el dispositivo, lo que deja espacio suficiente en la plataforma para futuras extensiones de diseño. El análisis de rendimiento temporal y eficiencia energética también fue completado con datos reales de implementación física, obteniéndose una latencia de cómputo de 360 ns, despreciable frente al periodo de muestreo de 1 ms, y un consumo medido de 193 mW; ambos resultados confirman que el cuello de botella del sistema no reside en la capacidad de cómputo de la FPGA sino en la dinámica mecánica del motor paso a paso.

El objetivo de validación sobre la planta física real se cumplió parcialmente: el lazo de control se cierra correctamente y el sistema logra estabilizar el péndulo tras la activación, aunque no de forma sostenida de manera indefinida, debido principalmente a dos limitaciones identificadas en la implementación VHDL (el desbordamiento del contador del encoder y la pérdida ocasional de pasos del motor a alta velocidad). La comparación de rendimiento frente a microcontrolador también se completó, si bien apoyándose en datos reales únicamente para la FPGA y para los recursos de la versión PID del microcontrolador, siendo el resto métricas estimadas al no haber llegado a la fase de implementación física. Por último, la extensión hacia el control LQG mediante filtro de Kalman no se llegó a implementar dentro del alcance de este TFG, aunque se concluye que sería viable dado el bajo uso de recursos del diseño actual, quedando como línea de trabajo futuro.

5.1.2. Conclusiones sobre la comparativa FPGA vs. microcontrolador

La comparativa muestra que la ventaja de la FPGA frente al microcontrolador depende fuertemente de las condiciones de comparación: frente a una implementación en coma flotante sin optimizar, la diferencia de latencia estimada es de aproximadamente 100 veces; frente a

una implementación equivalente en algoritmo y aritmética, se reduce a un factor de 3 a 8 veces. La ventaja genuina de la FPGA no está, por tanto, en una diferencia de velocidad bruta, sino en su determinismo temporal y en su capacidad de ejecución en paralelo. En consumo energético ambas plataformas resultan comparables en orden de magnitud, por lo que la FPGA no representa una optimización energética clara frente al microcontrolador. Esta ventaja de la FPGA tiene como contrapartida una mayor complejidad de diseño (descripción en VHDL, verificación mediante simulación y síntesis) frente a la mayor sencillez de desarrollo en C sobre un microcontrolador. En consecuencia, la FPGA se justifica en escenarios donde el determinismo temporal es un requisito crítico, mientras que el microcontrolador resulta preferible cuando prima la sencillez de desarrollo y el requisito de tiempo real se cumple con margen suficiente, como ocurre en este problema concreto, donde el actuador mecánico es la limitación real del sistema.

5.1.3. Lecciones aprendidas del proceso de desarrollo

El desarrollo del proyecto puso de manifiesto la importancia de la disciplina en aritmética de punto fijo: errores de truncamiento en constantes pequeñas, como la ganancia angular del diferenciador, exigieron el uso de un formato Q0.16 en lugar de Q8.8 para no perder precisión. La puesta en marcha del motor reveló la necesidad de una rampa de aceleración para respetar la curva de par del NEMA17 y de un mecanismo de arranque desacoplado (`simple_integrator`) para resolver el bloqueo de condición inicial del sistema. En definitiva, la implementación sobre la planta física reveló problemas que la simulación no había anticipado, confirmando que esta última, aunque necesaria para validar el diseño antes de pasar a hardware, no sustituye a la puesta en marcha sobre el sistema real.

5.2. Coste normalizado

Se incluye a continuación una estimación de coste normalizado, para complementar el análisis de eficiencia realizado a lo largo de la memoria.

Tabla 24: Componentes principales.

Componente	Cantidad	€/Unidad	Coste total (€)
Placa FPGA Basys3 (Digilent, precio académico)	1	111,75	111,75
Motor paso a paso NEMA17	1	12,00	12,00
Driver DRV8825	1	3,50	3,50
Encoder óptico en cuadratura (600 P/R)	1	25,00	25,00
Correa y poleas GT2	1	8,00	8,00
Estructura mecánica (raíl, carro, soportes)	1	30,00	30,00
Total			190,25

Tabla 25: Mano de obra.

Concepto	Horas	€/Hora	Coste total (€)
Diseño y simulación del controlador	100	40,00	4 000,00
Implementación en VHDL y depuración en FPGA	150	60,00	9 000,00
Implementación en microcontrolador y comparativa	60	45,00	2 700,00
Redacción de la memoria	110	40,00	4 400,00
Total			20 100,00

Tabla 26: Presupuesto general.

Concepto	Coste (€)
Componentes principales	190,25
Mano de obra	20 100,00
TOTAL	20 290,25

5.3. Trabajo Futuro

5.3.1. Extensión a control LQG (filtro de Kalman)

Para la implementación es necesario conocer de manera exacta el vector de estado, en la práctica existe degradación en la señal de control debido al ruido que pueden presentar las mediciones de los sensores. La extensión al filtro de Kalman como estimador de estado óptimo sería el siguiente paso para evitar este problema, transformando así el controlador LQR en un LQG completo. El filtro de Kalman requiere operaciones de multiplicación matricial y una división (para el cálculo de la ganancia de Kalman), lo que supondría mayor complejidad de diseño. La división en concreto, añadiría latencia al pipeline (como se ha visto con la operación de división en este Trabajo Fin de Grado, que añadía 34 ciclos extra de latencia). No obstante dado que se ha demostrado que el uso de recursos para el LQR es bajo, esta incorporación sería técnicamente viable y constituye la línea de trabajo futuro más directa sobre este mismo hardware.

5.3.2. Expansión del contador del encoder a 16 bits

El módulo `encoder_decoder` implementado en el diseño actual utiliza un contador de 12 bits con signo, lo que limita el rango de medición angular a ± 2047 cuentas ($\pm 307^\circ$ aproximadamente). Un rango suficiente para la operación normal del regulador en torno al equilibrio, aunque un movimiento sostenido continuo en un mismo sentido puede provocar desbordamiento antes de completar una vuelta completa (2400 cuentas/rev). Para evitarlo se debe ampliar el contador a 16 bits con signo lo que supondría un aumento de rango llegando a ± 32767 cuentas ($\pm 4915^\circ$), eliminando por completo el problema sin un impacto significativo en los recursos de la FPGA. Esta mejora no se desarrolló en el proyecto porque no se preveía operar fuera de la región crítica donde no había riesgo de desbordamiento.

5.3.3. Optimización de las matrices Q y R y simulación del sistema no lineal

Este trabajo se centraba en la implementación y el análisis de recursos del controlador en la FPGA. El diseño del control con la selección manual de las matrices de ponderaciones $Q = \text{diag}(10, 10, 100, 1)$ y $R = 1$ se ha realizado sin tener en cuenta factores del modelo real como lo son la fricción real del sistema mecánico y la inercia efectiva del motor, que introducen discrepancias entre el modelo linealizado y la planta real. Una línea de trabajo futuro es

identificar experimentalmente estos parámetros no modelados e incorporarlos a un modelo no lineal del sistema sobre el cual realizar un barrido del espacio (Q, R) evaluando métricas de robustez como el margen de estabilidad, el sobrepaso y el tiempo de establecimiento. Todo ello con el fin de conseguir un vector de ganancias que maximice la robustez de la respuesta frente a las no linealidades del sistema.

5.3.4. Portabilidad a FPGA de menor coste

El hecho de que el diseño desarrollado sobre la Basys 3 (Artix-7 XC7A35T) ocupe una fracción reducida de los recursos disponibles, sugiere que podría portarse el diseño a una FPGA de menor capacidad, como la familia ECP5 de Lattice Semiconductor (Lattice Semiconductor, 2024), que dispone de recursos suficientes para implementar el diseño y es compatible con un flujo de síntesis de código abierto (*toolchain* Yosys/nextpnr) (Shah et al., 2019), lo que facilitaría además la reproducibilidad del trabajo.

Bibliografía

- Alavinasab, A., Moharrami, H., & Khajepour, A. (2006). Active Control of Structures Using Energy-Based LQR Method. *Computer-Aided Civil and Infrastructure Engineering*, 21(8), 605-611.
- AMD Xilinx. (2021). *Floating-Point Operator v7.1 LogiCORE IP Product Guide (PG060)*. AMD Xilinx. Consultado el 8 de julio de 2026, desde <https://docs.amd.com/v/u/en-US/pg060-floating-point>
- Åström, K. J., & Furuta, K. (1996). Swing-up a Pendulum by Energy Control. *Automatica*, 36(2), 287-295.
- Åström, K. J., & Kumar, P. R. (2014). Control: A Perspective. *Automatica*, 50(1), 3-43.
- Bellman, R. (1966). Dynamic Programming. *Science*, 153(3731), 34-37.
- Bennett, S. (1993). *A History of Control Engineering 1930–1955*. Peter Peregrinus.
- Bishop, D. W. (2008). *Fixed Point Package User's Guide* [IEEE VHDL Fixed-Point Packages]. Consultado el 6 de mayo de 2026, desde https://fphdl.readthedocs.io/en/docs/fixed_pkg.html
- Bissell, C. (2009). A History of Automatic Control. En S. Y. Nof (Ed.), *Springer Handbook of Automation* (pp. 53-69). Springer. https://doi.org/10.1007/978-3-540-78831-7_4
- Bouabdallah, S. (2007). *Design and Control of Quadrotors with Application to Autonomous Flying* [Tesis doctoral, EPFL].
- Brogårdh, T. (2009). Robot Control Overview: An Industrial Perspective. 30, 167-180. <https://doi.org/10.4173/MIC.2009.3.7>
- Bryson, A. E., & Ho, Y.-C. (1975). *Applied Optimal Control: Optimization, Estimation, and Control*. Hemisphere Publishing.
- Burns, R. S. (2001). *Advanced Control Engineering*. Butterworth-Heinemann.
- Creality. (2024). *Motor PaP Stepper NEMA 17 40mm – Creality 42-40*. Consultado el 1 de junio de 2025, desde <https://naylampmechatronics.com/motores-pap-steppers/915-motor-pap-stepper-nema-17-40mm-creality.html>
- Dhewa, O. A., Dharmawan, A., & Priyambodo, T. K. (2017). Model of Linear Quadratic Regulator (LQR) Control Method in Hovering State of Quadrotor. *Journal of Telecommunication, Electronic and Computer Engineering*, 9(3), 135-143.

- Digilent Inc. (2016). *Basys 3 FPGA Board Reference Manual*. Digilent Inc. Consultado el 13 de junio de 2026, desde <https://digilent.com/reference/programmable-logic/basys-3/reference-manual>
- Dul, F., Lichota, P., & Rusowicz, A. (2020). Generalized Linear Quadratic Control for a Full Tracking Problem in Aviation. *Sensors*, 20(10), 2955.
- Franklin, G. F., Powell, J. D., & Emami-Naeini, A. (2014). *Feedback Control of Dynamic Systems* (7.^a ed.). Pearson.
- Gamkrelidze, R. V. (2006). Discovery of the Maximum Principle. En A. A. Bolibruch, Y. S. Osipov & Y. G. Sinai (Eds.), *Mathematical Events of the Twentieth Century* (pp. 85-99). Springer.
- Gong, Y., Wu, X., & Ma, H. (2015). Research on Control Strategy of Two-Wheeled Self-Balancing Robot. *2015 International Conference on Computer Science and Mechanical Automation (CSMA)*, 281-284. <https://doi.org/10.1109/CSMA.2015.63>
- Graf, C., & Röfer, T. (2012). A Center of Mass Observing 3D-LIPM Gait for the RoboCup Standard Platform League Humanoid. En *RoboCup 2011: Robot Soccer World Cup XV* (pp. 102-113). https://doi.org/10.1007/978-3-642-32060-6_9
- Hespanha, J. P. (2005). Lecture Notes on LQR/LQG Controller Design [Apuntes/lecture notes en línea. Fecha de acceso: 2026-05-12].
- Industrial Shields. (2024). *Two Phase AB Incremental Rotary Encoder 600 p/r, 6 mm Shaft, DC 5-24 V*. Consultado el 1 de junio de 2025, desde https://www.industrialshields.com/es_ES/shop/210001000100-two-phase-ab-incremental-rotary-encoder-600p-r-6-mm-shaft-dc5-24vdc-934
- Irfan, S., Zhao, L., Ullah, S., Mehmood, A., & Fasih Uddin Butt, M. (2024). Control strategies for inverted pendulum: A comparative analysis of linear, nonlinear, and artificial intelligence approaches. *PLOS ONE*, 19(3), 1-19. <https://doi.org/10.1371/journal.pone.0298093>
- Isermann, R. (1989). *Digital Control Systems*. Springer.
- Kahl, T., & Dieckerhoff, S. (2017). Comparison of FPGA- and Microcontroller-Based Control of a High-Dynamic Power Electronic Converter. *2017 IEEE 18th Workshop on Control and Modeling for Power Electronics (COMPEL)*, 1-6. <https://doi.org/10.1109/COMPEL.2017.8013288>
- Kálmán, R. E. (1960). Contributions to the Theory of Optimal Control. *Boletín de la Sociedad Matemática Mexicana*, 5(2), 102-119.

- Karhan, H. K., Yaren, T., & Kizir, S. (2024). Modelling and Control of a Reaction Wheel Pendulum Using a Linear Quadratic Regulator. *Sigma Journal of Engineering and Natural Sciences*, 42(6), 1907-1915. <https://doi.org/10.14744/sigma.2024.00157>
- Khan, M. (2026). Real-Time Implementation of FPGA-Based Controllers: Design Strategies and Performance Evaluation. *American Journal Of Antennas And Microwave Engineering*, 7, 24-31. <https://doi.org/10.71465/ajame.3571>
- Kilts, S. (2007). *Advanced FPGA design: architecture, implementation, and optimization*. John Wiley & Sons.
- Kouvaritakis, B., & Cannon, M. (2016). *Model Predictive Control: Classical, Robust and Stochastic*. Springer International Publishing. <https://doi.org/10.1007/978-3-319-24853-0>
- Lattice Semiconductor. (2024). *ECP5 and ECP5-5G Family Data Sheet*. Consultado el 8 de julio de 2026, desde https://www.latticesemi.com/view_document?document_id=50461
- Lewis, F. L., Vrabie, D., & Syrmos, V. L. (2012). *Optimal Control* (3.^a ed.). John Wiley & Sons. <https://onlinelibrary.wiley.com/doi/book/10.1002/9781118122631>
- Lin, Y., & Morton, T. D. (2017). A Microcontroller-based DSP Laboratory Curriculum. <https://doi.org/10.18260/1-2--27480>
- Mohd Isa, N. I. A. (2008). *DC Motor Controller Using Linear Quadratic Regulator (LQR) Algorithm Implementation on PIC* [Bachelor's thesis]. Universiti Malaysia Pahang.
- Neacșu, D. O., & Sîrbu, A. (2020). Design of a LQR-Based Boost Converter Controller for Energy Savings. *IEEE Transactions on Industrial Electronics*, 67(7), 5379-5388. <https://doi.org/10.1109/TIE.2019.2934062>
- Nise, N. S. (2020). *Control Systems Engineering* (8.^a ed.). Wiley.
- Ogata, K. (1995). *Discrete-Time Control Systems* (2.^a ed.). Prentice Hall.
- Ogata, K. (2010). *Modern Control Engineering*. Prentice Hall.
- Ozer, A. O. (2011). *Exact boundary controllability and feedback stabilization for a multi-layer Rao-Nakra beam* [ResearchGate]. Consultado el 6 de julio de 2026, desde https://www.researchgate.net/figure/Water-clock-of-Ktesibios-In-his-design-there-are-mainly-two-tanks-BDKE-and-KMOL-The_fig2.241809613
- Ramos, F., & Filho, W. (2007). Extending the Linear Quadratic Design of a Launch Vehicle Attitude Controller Through Genetic Optimization. *IFAC Proceedings Volumes*, 40, 780-785. <https://doi.org/10.3182/20070625-5-FR-2916.00133>

- Rodríguez Criado, J., López de Azcona, J., & Millán Sánchez, T. (2025). Péndulo invertido [Proyecto de la asignatura Sistemas Electrónicos (Máster en Ingeniería Industrial) no publicado].
- Saraf, P., Gupta, M., & Parimi, A. M. (2020). A Comparative Study Between a Classical and Optimal Controller for a Quadrotor. *2020 IEEE 17th India Council International Conference (INDICON)*, 1-6.
- Shah, D., Hung, E., Wolf, C., Bazanski, S., Gisselquist, D., & Milanovic, M. (2019). Yosys+nextpnr: An Open Source Framework from Verilog to Bitstream for Commercial FPGAs. *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.
- Snider, J. M. (2009). *Automatic Steering Methods for Autonomous Automobile Path Tracking* (inf. téc. N.º CMU-RI-TR-09-08). Robotics Institute, Carnegie Mellon University.
- Suzuki, K. (2013). Control System Basics and PID Control. En Y. Oka & K. Suzuki (Eds.), *Nuclear Reactor Kinetics and Plant Control* (pp. 49-91). Springer Japan. https://doi.org/10.1007/978-4-431-54195-0_5
- Texas Instruments. (2014). *DRV8825 Stepper Motor Controller IC*. Texas Instruments. Consultado el 27 de junio de 2026, desde <https://www.ti.com/lit/ds/symlink/drv8825.pdf>
- Torrestiana González, W., González González, L., González Mendoza, E., & Bautista Cruz, L. (2026). FPGA-Based LQG Control of a Low-Speed Active Magnetic Bearing Joint: Implementation and Validation. *Journal of Control, Automation and Electrical Systems*. <https://doi.org/10.1007/s40313-026-01286-6>
- University of Michigan. (2024). *Inverted Pendulum: System Modeling* [Control Tutorials for MATLAB and Simulink.]. Consultado el 1 de febrero de 2025, desde <https://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum§ion=SystemModeling>
- Xilinx Inc. (2018). *7 Series DSP48E1 Slice User Guide (UG479)*. Consultado el 7 de julio de 2026, desde https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1
- Yang, Y. (2014). Quaternion-Based LQR Spacecraft Control Design Is a Robust Pole Assignment Design. *Journal of Aerospace Engineering*, 27(1), 168-176. [https://doi.org/10.1061/\(ASCE\)AS.1943-5525.0000232](https://doi.org/10.1061/(ASCE)AS.1943-5525.0000232)
- Yates, R. (2009). Fixed-Point Arithmetic: An Introduction. Consultado el 20 de mayo de 2026, desde <https://api.semanticscholar.org/CorpusID:124376111>

- Zhang, P., Mills, A., Zambreno, J., & Jones, P. H. (2017). The Design and Integration of a Software Configurable and Parallelized Coprocessor Architecture for LQR Control. *Journal of Parallel and Distributed Computing*, 106, 121-131. <https://doi.org/10.1016/j.jpdc.2017.01.028>
- Zhao, D. (2024). Navigating the Electronic Landscape: Exploring FPGAs and MCUs Architectures in Electronic Design. *Applied and Computational Engineering*, 47, 87-91. <https://doi.org/10.54254/2755-2721/47/20241207>

Anexo A Alineación con los ODS

Los Objetivos de Desarrollo Sostenible (ODS) son un conjunto de 17 objetivos globales adoptados por las Naciones Unidas en 2015 como parte de la Agenda 2030, con el fin de promover el desarrollo sostenible en sus dimensiones económica, social y medioambiental (Naciones Unidas, [s.f.](#)). En el ámbito académico e investigador, se considera buena práctica identificar la contribución de cada trabajo a estos objetivos, para poder enmarcar su impacto en la práctica. Este trabajo se alinea con los siguientes Objetivos de Desarrollo Sostenible:

ODS 9: Industria, innovación e infraestructura

En primer lugar, este trabajo fin de grado se encuentra estrechamente relacionado con el ODS 9: Industria, innovación e infraestructura, cuya meta es aumentar la investigación científica y la capacidad tecnológica de los sectores industriales. La implementación de controladores óptimos en plataformas FPGA permite el desarrollo de tecnologías más eficientes y potentes, facilitando la modernización de los sistemas de control en entornos industriales y académicos.



Figura 30: Icono del ODS 9 (Naciones Unidas, [s.f.](#)).

ODS 7: Energía asequible y no contaminante

Por otro lado, el trabajo presenta una contribución al ODS 7: Energía asequible y no contaminante, dado que la mejora en la eficiencia en los sistemas de control supone una reducción del consumo energético en aplicaciones de sistemas automatizados.



Figura 31: Icono del ODS 7 (Naciones Unidas, [s.f.](#)).

ODS 8: Trabajo decente y crecimiento económico

Este trabajo contribuye indirectamente al ODS 8: Trabajo decente y crecimiento económico, cuya meta incluye el aumento de la productividad económica mediante la modernización tecnológica. La mejora en la eficiencia y reducción de latencia de los sistemas de control tiene un impacto directo en la productividad industrial, ya que procesos automatizados más rápidos y precisos reducen tiempos de ciclo y errores en la producción.



Figura 32: Icono del ODS 8 (Naciones Unidas, [s.f.](#)).

ODS 4: Educación de calidad

Por último, el trabajo se encuentra vinculado al ODS 4: Educación de calidad, por estar desarrollado en un entorno académico y participar en la formación en áreas como el control avanzado y los sistemas hardware reconfigurables. Además, la implementación práctica del controlador en la FPGA tiene la posibilidad de, en un futuro, servir de base para actividades docentes u otros proyectos de aprendizaje en ingeniería.



Figura 33: Icono del ODS 4 (Naciones Unidas, [s.f.](#)).

Anexo B Código MATLAB

```
1 % Parámetros físicos
2 M = 0.272; m = 0.027; l = 0.12;
3 I = 1/12*m*(2*l)^2;
4 b = 0; g = 9.81;
5
6 p = I*(M+m) + M*m*l^2;
7
8 % Matrices del sistema
9 A = [0, 1, 0, 0;
10      0, -(I+m*l^2)*b/p, m^2*g*l^2/p, 0;
11      0, 0, 0, 1;
12      0, -(m*l*b)/p, m*g*l*(M+m)/p, 0];
13
14 B = [0; (I+m*l^2)/p; 0; m*l/p];
15
16 C = eye(4);
17 D = zeros(4,1);
18
19 % LQR continuo
20 Q = diag([10, 10, 100, 1]);
21 R = 1;
22 [K, ~, ~] = lqr(A, B, Q, R);
23
24 % Discretización
25 Ts = 0.001;
26 sys_cont = ss(A, B, C, D);
27 sys_disc = c2d(sys_cont, Ts, 'zoh');
28 Ad = sys_disc.A;
29 Bd = sys_disc.B;
30 Cd = eye(4);
31 Dd = zeros(4,1);
```

```
32 [Kd, ~, ~] = dlqr(Ad, Bd, Q, R);
33
34 fprintf('K continua : [%.4f %.4f %.4f %.4f]\n', K)
35 fprintf('K discreta : [%.4f %.4f %.4f %.4f]\n', Kd)
36 fprintf('Ts = %g s\n', Ts)
```

Código 1: Diseño del regulador LQR: modelo linealizado, discretización y cálculo de ganancias

```
1 angulos_deg = [3, 5, 8, 10, 15];
2 t_final = 5;
3 n_pasos = round(t_final / Ts);
4 t_vec = (0:n_pasos)' * Ts;
5 Acl_d = Ad - Bd*Kd;
6
7 for ia = 1:length(angulos_deg)
8     ang = angulos_deg(ia);
9     x0_k = [0; 0; deg2rad(ang); 0];
10
11     X = zeros(n_pasos+1, 4);
12     X(1,:) = x0_k';
13     for k = 1:n_pasos
14         X(k+1,:) = Acl_d * X(k,:);
15     end
16     % almacenar / graficar X segun corresponda
17 end
```

Código 2: Barrido de ángulos iniciales en lazo cerrado discreto

```
1 x0_B = [0; 0; deg2rad(15); 0];
2
3 casos_Q = {diag([10, 10, 100, 1]), ...
4             diag([10, 10, 200, 1]), ...
5             diag([20, 20, 200, 1]), ...
6             diag([20, 20, 100, 1])};
7
8 tol_2pct = 0.02 * deg2rad(15);
```

```
9
10 for iq = 1:length(casos_Q)
11     Q_iq = casos_Q{iq};
12     [Kd_iq, ~, ~] = dlqr(Ad, Bd, Q_iq, R);
13     Acl_iq = Ad - Bd*Kd_iq;
14
15     X = zeros(n_pasos+1, 4);
16     X(1,:) = x0_B';
17     for k = 1:n_pasos
18         X(k+1,:) = Acl_iq * X(k,:)';
19     end
20
21     theta = X(:,3);
22     if max(abs(theta)) > 0.35
23         fprintf('No estabiliza\n');
24     else
25         idx_last = find(abs(theta) > tol_2pct, 1, 'last');
26         if isempty(idx_last)
27             fprintf('Tiempo de establecimiento (2%%): < %.4f s\
28                 n', t_vec(2));
29         else
30             fprintf('Tiempo de establecimiento (2%%): %.4f s\n'
31                 , t_vec(idx_last));
32         end
33     end
34 end
```

Código 3: Barrido de pesos Q y calculo del tiempo de establecimiento

Anexo C Simulaciones Simulink

Los esquemas utilizados para realizar las simulaciones del sistema discreto y continuo se muestran en las Figuras 34 y 35.

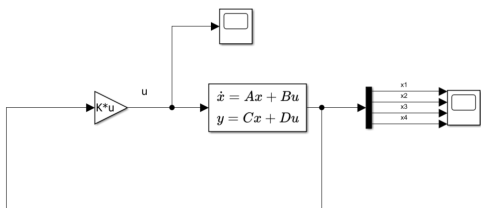


Figura 34: Esquema de Simulink sistema continuo.

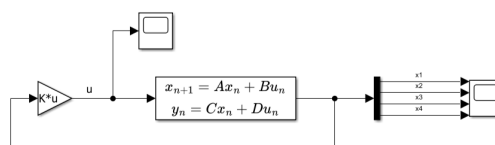


Figura 35: Esquema de Simulink sistema discreto.

A continuación se muestran las respuestas obtenidas para las distintas simulaciones.

La Figura 36 y la Figura 37 recoge la respuesta de los sistemas para el vector de condiciones iniciales: $\mathbf{x}_0 = [0,05 \text{ m}, 0, 10^\circ, 1 \text{ rad/s}]^T$.

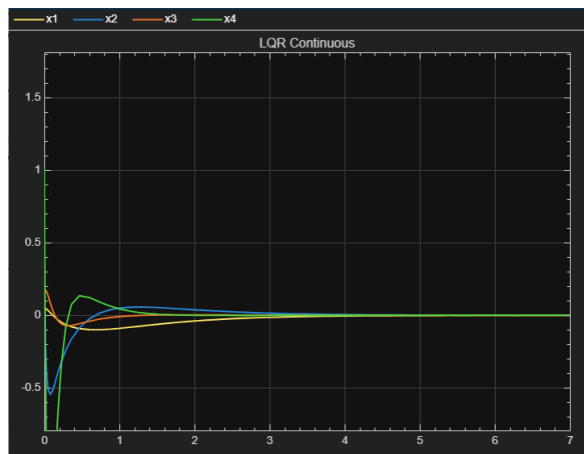


Figura 36: Simulación sistema continuo I.

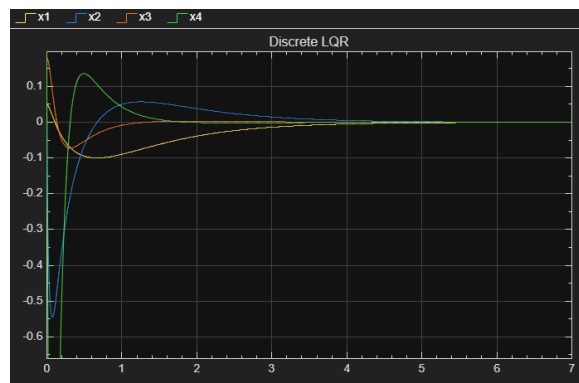


Figura 37: Simulación sistema discreto I.

La Figura 38 y la Figura 39 recoge la respuesta de los sistemas para el vector de condiciones iniciales: $\mathbf{x}_0 = [0, 0, 15^\circ, 1,5 \text{ rad/s}]^T$.

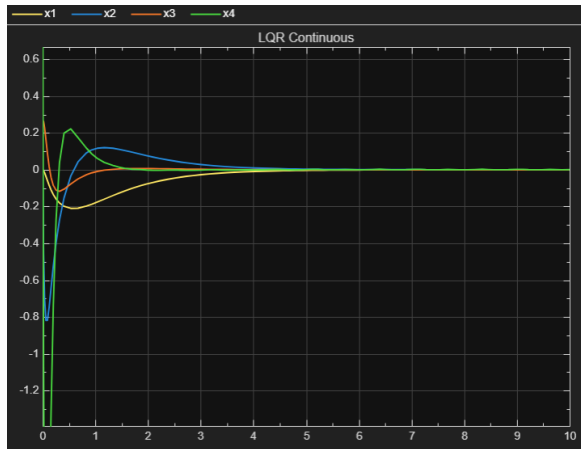


Figura 38: Simulación sistema continuo II.

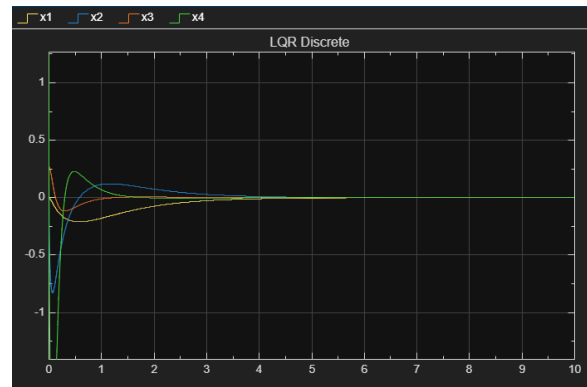


Figura 39: Simulación sistema discreto II.

La Figura 40 y la Figura 41 recoge la respuesta de los sistemas para el vector de condiciones iniciales: $\mathbf{x}_0 = [0, 0,1 \text{ m/s}, 5^\circ, 1,2 \text{ rad/s}]^T$.

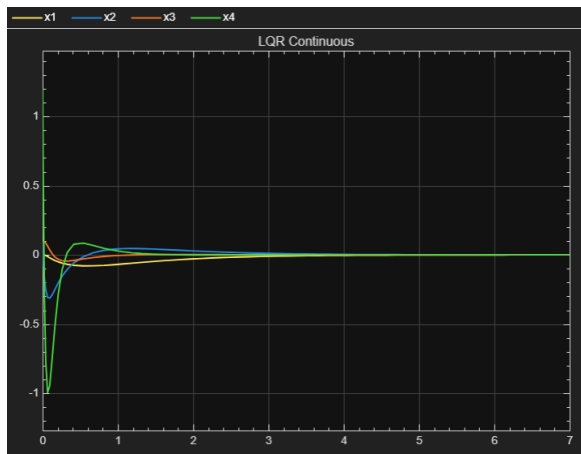


Figura 40: Simulación sistema continuo III.

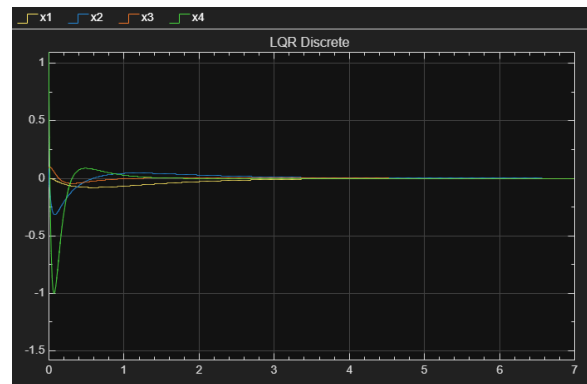


Figura 41: Simulación sistema discreto III.

Anexo D Código VHDL

`lqr_controller.vhd`

```
1  -----
2  -- MODULO 4: Controlador LQR
3  -- Formato: Q8.8 signed(15 downto 0), escala = 256, Ts = 1 ms
4  -- Funcion: Ley de control  $u = -(k_1x_1 + k_2x_2 + k_3x_3 + k_4x_4)$ 
5  --
6  -- Pesos MATLAB:  $Q = \text{diag}([10, 10, 100, 1])$ ,  $R = 1$ ,  $T_s = 1$  ms
7  -- Ganancias LQR calculadas en MATLAB, convertidas a Q8.8 =  $\text{round}(k * 256)$ :
8  --  $k_1 = -3.1009 \rightarrow \text{round}(-3.1009 * 256) = -794$ 
9  --  $k_2 = -4.5400 \rightarrow \text{round}(-4.5400 * 256) = -1162$ 
10 --  $k_3 = 20.3267 \rightarrow \text{round}(20.3267 * 256) = 5204$ 
11 --  $k_4 = 2.4497 \rightarrow \text{round}(2.4497 * 256) = 627$ 
12 --
13 -- Latencia total: 2 ciclos de reloj (1 ciclo para los multiplicadores +
14 -- 1 ciclo de retardo para sincronizar el sumador), equivalentes a 20 ns
15 -- a 100 MHz - totalmente despreciable frente a  $T_s = 1$  ms.
16 -- Autor: Alicia Lopez Garcia
17 -----
18
19 library IEEE;
20 use IEEE.STD_LOGIC_1164.ALL;
21 use IEEE.NUMERIC_STD.ALL;
22
23 entity lqr_controller is
24     port (
25         clk : in std_logic;
26         rst : in std_logic;
27         tick : in std_logic;
28         x1 : in signed(15 downto 0);
29         x2 : in signed(15 downto 0);
30         x3 : in signed(15 downto 0);
31         x4 : in signed(15 downto 0);
32         u : out signed(15 downto 0)
33     );
34 end lqr_controller;
35
36 architecture Structural of lqr_controller is
37
38     constant K1 : signed(15 downto 0) := to_signed(-794, 16);
39     constant K2 : signed(15 downto 0) := to_signed(-1162, 16);
40     constant K3 : signed(15 downto 0) := to_signed(5204, 16);
41     constant K4 : signed(15 downto 0) := to_signed(627, 16);
42
```

```
43     signal p1, p2, p3, p4 : signed(15 downto 0);
44     signal tick_d1 : std_logic;
45
46     component mult_q8_8 is
47     port (
48         clk : in  std_logic;
49         rst : in  std_logic;
50         en  : in  std_logic;
51         a   : in  signed(15 downto 0);
52         b   : in  signed(15 downto 0);
53         p   : out signed(15 downto 0)
54     );
55 end component;
56
57     component sumador_sat is
58     port (
59         clk      : in  std_logic;
60         rst      : in  std_logic;
61         en       : in  std_logic;
62         p1, p2, p3, p4 : in  signed(15 downto 0);
63         u        : out signed(15 downto 0)
64     );
65 end component;
66
67 begin
68
69     -- Retardo de 1 ciclo para sincronizar el sumador con los productos ya
70     -- registrados por los multiplicadores (evita capturar valores obsoletos)
71     proceso_delay_tick : process(clk)
72     begin
73         if rising_edge(clk) then
74             if rst = '1' then
75                 tick_d1 <= '0';
76             else
77                 tick_d1 <= tick;
78             end if;
79         end if;
80     end process proceso_delay_tick;
81
82     MULT1 : mult_q8_8 port map (clk => clk, rst => rst, en => tick, a => K1, b => x1, p =>
83         p1);
84     MULT2 : mult_q8_8 port map (clk => clk, rst => rst, en => tick, a => K2, b => x2, p =>
85         p2);
86     MULT3 : mult_q8_8 port map (clk => clk, rst => rst, en => tick, a => K3, b => x3, p =>
87         p3);
88     MULT4 : mult_q8_8 port map (clk => clk, rst => rst, en => tick, a => K4, b => x4, p =>
89         p4);
86
87     SUMA : sumador_sat
88         port map (
89             clk => clk, rst => rst, en => tick_d1,
```

```

90         p1 => p1, p2 => p2, p3 => p3, p4 => p4,
91         u  => u
92     );
93
94 end Structural;

```

Código 4: Controlador LQR: ley de control $u = -K_d \mathbf{x}$.

cart_estimator.vhd

```

1  -- =====
2  -- MODULE: cart_estimator
3  -- Estimates cart position (x1) and velocity (x2) using
4  -- step pulse counting from the stepper motor driver.
5  --
6  -- All outputs in Q8.8 signed fixed-point (scale = 256).
7  -- x1: position [m], range +-0.16 m (+-3200 steps)
8  -- x2: velocity [m/s], range +-2.0 m/s
9  --
10 -- Conversion (20000 microsteps/m, 1/8 microstepping, Ts = 1 ms):
11 --
12 -- x1 [Q8.8] = step_counter * (256 / 20000)
13 --             = (step_counter * 839) >> 16
14 -- where 839 = round(256/20000 * 65536) in Q0.16; error +0.017%
15 --
16 -- x2 [Q8.8] = delta_steps * (256 / 20000 / 0.001)
17 --             = delta_steps * 12.8
18 --             = (delta_steps * 3277) >> 8
19 -- where 3277 = round(12.8 * 256) in Q8.8 factor; error +0.006%
20 --
21 -- Target: Xilinx Artix-7 (Basys3), 100 MHz
22 -- Author: Alicia Lopez Garcia
23 -- =====
24
25 library ieee;
26 use ieee.std_logic_1164.all;
27 use ieee.numeric_std.all;
28
29 entity cart_estimator is
30     port (
31         clk        : in  std_logic;
32         rst        : in  std_logic;
33         tick       : in  std_logic;
34         step_tick  : in  std_logic;
35         step_dir   : in  std_logic;
36         lqr_enable : in  std_logic;
37         x1         : out signed(15 downto 0);
38         x2         : out signed(15 downto 0);
39     );

```

```

40 end entity cart_estimator;
41
42 architecture rtl of cart_estimator is
43
44     signal step_counter      : signed(31 downto 0) := (others => '0');
45     signal step_counter_prev : signed(31 downto 0) := (others => '0');
46     signal x1_reg : signed(15 downto 0) := (others => '0');
47     signal x2_reg : signed(15 downto 0) := (others => '0');
48
49 begin
50
51     x1 <= x1_reg;
52     x2 <= x2_reg;
53
54     main_proc : process(clk)
55         variable step_sat : signed(31 downto 0);
56         variable delta    : signed(31 downto 0);
57         variable x1_raw   : signed(42 downto 0);
58         variable x2_raw   : signed(44 downto 0);
59         variable x2_tmp   : signed(15 downto 0);
60     begin
61         if rising_edge(clk) then
62
63             if rst = '1' or lqr_enable = '0' then
64                 step_counter      <= (others => '0');
65                 step_counter_prev <= (others => '0');
66                 x1_reg            <= (others => '0');
67                 x2_reg            <= (others => '0');
68
69             else
70                 -- Conteo de micropasos: se actualiza cada ciclo de reloj
71                 if step_tick = '1' then
72                     if step_dir = '1' then
73                         step_counter <= step_counter + 1;
74                     else
75                         step_counter <= step_counter - 1;
76                     end if;
77                 end if;
78
79                 if tick = '1' then
80
81                     delta := step_counter - step_counter_prev;
82                     step_counter_prev <= step_counter;
83
84                     -- x1: satura a +-3200 pasos (+-0.16 m) antes de multiplicar
85                     if step_counter > to_signed(3200, 32) then
86                         step_sat := to_signed(3200, 32);
87                     elsif step_counter < to_signed(-3200, 32) then
88                         step_sat := to_signed(-3200, 32);
89                     else
90                         step_sat := step_counter;

```

```

91         end if;
92
93         x1_raw := step_sat * to_signed(839, 11);
94         x1_reg <= x1_raw(31 downto 16);
95
96         -- x2: satura a +-512 (+-2 m/s)
97         x2_raw := delta * to_signed(3277, 13);
98         x2_tmp := x2_raw(23 downto 8);
99
100        if x2_tmp > to_signed(512, 16) then
101            x2_reg <= to_signed(512, 16);
102        elsif x2_tmp < to_signed(-512, 16) then
103            x2_reg <= to_signed(-512, 16);
104        else
105            x2_reg <= x2_tmp;
106        end if;
107
108    end if;
109 end if;
110 end if;
111 end process main_proc;
112
113 end architecture rtl;

```

Código 5: Estimación de posición y velocidad del carro a partir de pulsos STEP reales.

motor_command.vhd

```

1  -----
2  -- MODULE 6: Motor Command with Acceleration Ramp
3  -- Converts cart velocity command x2_cmd (Q8.8) into period/direction/enable
4  -- signals for step_generator, with trapezoidal velocity ramp.
5  --
6  -- Mechanics: GT2 belt + 40-tooth pulley -> 80 mm/rev linear pitch
7  --             NEMA17 + 1/8 microstepping -> 1600 microsteps/rev
8  --             Constant: 1600 / 0.080 m = 20000 microsteps/metre
9  --
10 -- Clock: 100 MHz (f_clk = 100_000_000 Hz), Ts = 1 ms
11 -- Format: Q8.8 signed(15 downto 0), scale = 256
12 -- Author: Alicia Lopez Garcia
13 -----
14
15 library IEEE;
16 use IEEE.STD_LOGIC_1164.ALL;
17 use IEEE.NUMERIC_STD.ALL;
18
19 entity motor_command is
20     generic (
21         DIV_LATENCY : integer := 34;

```

```

22         X2_MIN      : integer := 7    -- umbral de activacion, Q8.8
23     );
24     port (
25         clk          : in  std_logic;
26         rst          : in  std_logic;
27         tick         : in  std_logic;
28         x2_cmd       : in  signed(15 downto 0);
29         period       : out unsigned(16 downto 0);
30         direction    : out std_logic;
31         enable       : out std_logic
32     );
33 end motor_command;
34
35 architecture Behavioral of motor_command is
36
37     constant F_CLK      : unsigned(31 downto 0) := to_unsigned(100_000_000, 32);
38     constant STEPS_P_M  : unsigned(16 downto 0) := to_unsigned(20_000, 17);
39     constant PER_ARRANQUE : unsigned(16 downto 0) := to_unsigned(50_000, 17); -- 2000 pasos/
40                                     s
41     constant PER_MINIMA  : unsigned(16 downto 0) := to_unsigned(10_000, 17); -- 10000 pasos
42                                     /s
43     constant PER_STEP    : unsigned(16 downto 0) := to_unsigned( 200, 17);
44
45     component div_gen_0
46     port (
47         aclk          : in  std_logic;
48         s_axis_divisor_tvalid : in  std_logic;
49         s_axis_divisor_tdata  : in  std_logic_vector(31 downto 0);
50         s_axis_dividend_tvalid : in  std_logic;
51         s_axis_dividend_tdata  : in  std_logic_vector(31 downto 0);
52         m_axis_dout_tvalid    : out std_logic;
53         m_axis_dout_tdata     : out std_logic_vector(63 downto 0)
54     );
55 end component;
56
57     type t_state is (LQR_IDLE, LQR_RIGHT, LQR_LEFT);
58     signal state : t_state;
59
60     signal x2_abs      : unsigned(15 downto 0);
61     signal prod_steps  : unsigned(32 downto 0);
62     signal steps_per_s : unsigned(24 downto 0);
63     signal divisor_safe : std_logic_vector(31 downto 0);
64
65     signal div_valid_in : std_logic;
66     signal dout_tdata   : std_logic_vector(63 downto 0);
67     signal dout_tvalid  : std_logic;
68     signal quotient     : unsigned(31 downto 0);
69
70     signal period_reg : unsigned(16 downto 0);
71     signal dir_latch  : std_logic;
72     signal en_latch   : std_logic;

```

```

71
72 begin
73
74     process(x2_cmd)
75     begin
76         if x2_cmd >= 0 then
77             x2_abs <= unsigned(x2_cmd);
78         else
79             x2_abs <= unsigned(-x2_cmd);
80         end if;
81     end process;
82
83     -- Escala Q8.8 a microsteps/s: (x2_abs / 256) * 20000
84     prod_steps <= x2_abs * STEPS_P_M;
85     steps_per_s <= prod_steps(32 downto 8);
86
87     -- Divisor seguro: evita division por cero
88     process(steps_per_s)
89     begin
90         if steps_per_s /= 0 then
91             divisor_safe <= std_logic_vector(resize(steps_per_s, 32));
92         else
93             divisor_safe <= std_logic_vector(to_unsigned(1, 32));
94         end if;
95     end process;
96
97     div_valid_in <= tick;
98     quotient <= unsigned(dout_tdata(31 downto 0));
99
100    inst_div : div_gen_0
101    port map (
102        aclk          => clk,
103        s_axis_divisor_tvalid => div_valid_in,
104        s_axis_divisor_tdata  => divisor_safe,
105        s_axis_dividend_tvalid => div_valid_in,
106        s_axis_dividend_tdata  => std_logic_vector(F_CLK),
107        m_axis_dout_tvalid    => dout_tvalid,
108        m_axis_dout_tdata     => dout_tdata
109    );
110
111    process(clk)
112        variable per_vel : unsigned(16 downto 0);
113        variable per_new : unsigned(16 downto 0);
114    begin
115        if rising_edge(clk) then
116            if rst = '1' then
117                state <= LQR_IDLE;
118                dir_latch <= '0';
119                en_latch <= '0';
120                period_reg <= PER_ARRANQUE;
121            end if;

```

```

122     elsif tick = '1' then
123         -- Transicion de estado evaluada cada tick de 1 ms
124         if x2_cmd > to_signed(X2_MIN, 16) then
125             state      <= LQR_RIGHT;
126             dir_latch <= '0';
127             en_latch  <= '1';
128         elsif x2_cmd < to_signed(-X2_MIN, 16) then
129             state      <= LQR_LEFT;
130             dir_latch <= '1';
131             en_latch  <= '1';
132         else
133             state      <= LQR_IDLE;
134             dir_latch <= '1';
135             en_latch  <= '0';
136             period_reg <= PER_ARRANQUE;
137         end if;
138
139     elsif dout_tvalid = '1' then
140         -- Actualizacion de rampa: solo con motor activo
141         if state /= LQR_IDLE then
142             if quotient > resize(PER_ARRANQUE, 32) then
143                 per_vel := PER_ARRANQUE;
144             elsif quotient < resize(PER_MINIMA, 32) then
145                 per_vel := PER_MINIMA;
146             else
147                 per_vel := quotient(16 downto 0);
148             end if;
149
150             if period_reg > per_vel + PER_STEP then
151                 per_new := period_reg - PER_STEP;
152             elsif period_reg > per_vel then
153                 per_new := per_vel;
154             elsif period_reg + PER_STEP < per_vel then
155                 per_new := period_reg + PER_STEP;
156             elsif period_reg < per_vel then
157                 per_new := per_vel;
158             else
159                 per_new := period_reg;
160             end if;
161
162             if per_new > PER_ARRANQUE then
163                 period_reg <= PER_ARRANQUE;
164             elsif per_new < PER_MINIMA then
165                 period_reg <= PER_MINIMA;
166             else
167                 period_reg <= per_new;
168             end if;
169         end if;
170     end if;
171 end if;
172 end process;

```

```

173
174     period    <= period_reg;
175     direction <= dir_latch;
176     enable    <= en_latch;
177
178 end Behavioral;

```

Código 6: Conversión de consigna de velocidad a periodo de pulso, con rampa trapezoidal.

lqr_system_fsm.vhd

```

1  -- lqr_system_fsm.vhd
2  -- Maquina de estados de alto nivel del sistema de control LQR (v2).
3  --
4  -- S_HOMING:          avanza hacia la izquierda hasta activar el FdC (nivel bajo). LQR
5  --                   deshabilitado, contador de pasos sin iniciar.
6  -- S_BACKING:         retrocede (derecha) g_center_offset_steps pasos para centrar el carro.
7  --                   Al llegar, captura count_offset_reg = enc_count_i (referencia +-180
8  --                   grados)
9  --                   y pasa a S_WAIT_CRITICAL.
10 -- S_WAIT_CRITICAL:   LQR deshabilitado; motor parado. El angulo (count_rel) se sigue
11 --                   actualizando en todo momento. Pasa a S_LQR_ACTIVE en cuanto
12 --                   |count_rel| <= g_critical_counts (aprox. +-15 grados).
13 -- S_LQR_ACTIVE:      lqr_enable_o='1'. El movimiento del motor lo gobierna la cadena
14 --                   lqr_controller -> motor_command -> step_generator (esta FSM no
15 --                   genera step_enable_o aqui). Sale por 3 condiciones:
16 --                   1) |count_rel| > g_safety_counts (control fallido)          ->
17 --                   S_WAIT_CRITICAL
18 --                   2) limit_fdc_i='0' (choque con el FdC izquierdo)          -> S_HOMING
19 --                   3) step_cnt_lqr = 2*g_center_offset_steps (fin de carril sin FdC) ->
20 --                   S_HOMING
21 --
22 -- En el caso 1) se conserva el contador de pasos (step_cnt_lqr) para
23 -- poder detectar el caso 3) si se reactiva el LQR y sigue desplazandose
24 -- en la misma direccion.
25 --
26 -- NOTA DE DISEÑO: no se incluye un estado S_SAFETY_STOP terminal. Ante fallo
27 -- (casos 1, 2 o 3) el sistema siempre vuelve a intentarlo automaticamente
28 -- (a WAIT_CRITICAL o HOMING segun el caso), sin limite de reintentos.
29 --
30 --
31 -- NOTA DE DISEÑO: g_center_offset_steps=3_000 centra el carro (3000 pasos x
32 -- 0.05 mm/paso = 150 mm, la mitad de la carrera util) con la configuracion
33 -- final de polea de 40 dientes y microstepping 1/8. Si cambia la polea o el
34 -- microstepping, este valor debe recalcularse.
35 --
36 --
37 -- Reset: SINCRONO
38 --
39 -- Author: Alicia Lopez Garcia
40 --
41 library ieee;

```

```

36 use ieee.std_logic_1164.all;
37 use ieee.numeric_std.all;
38 use ieee.math_real.all;
39
40 entity lqr_system_fsm is
41     generic (
42         g_clk_hz           : positive := 100_000_000;
43         g_homing_freq_hz   : positive := 5_000;
44         g_center_offset_steps : positive := 3_000;
45         g_period_width      : positive := 24;
46         g_count_width       : positive := 12;
47         g_critical_counts    : positive := 100;
48         g_safety_counts      : positive := 200;
49         g_settle_time_s      : positive := 8
50     );
51     port (
52         clk      : in std_logic;
53         reset     : in std_logic;
54
55         limit_fdc_i      : in std_logic;
56         enc_count_i      : in signed(g_count_width - 1 downto 0);
57         step_tick_i      : in std_logic;
58         step_dir_i       : in std_logic;
59         still_i          : in std_logic;
60
61         step_enable_o     : out std_logic;
62         step_dir_o        : out std_logic;
63         period_cycles_o   : out unsigned(g_period_width - 1 downto 0);
64         count_offset_o    : out signed(g_count_width - 1 downto 0);
65         lqr_enable_o      : out std_logic;
66         state_o           : out std_logic_vector(2 downto 0)
67     );
68 end entity lqr_system_fsm;
69
70 architecture rtl of lqr_system_fsm is
71
72     type t_state is (S_HOMING, S_BACKING, S_WAIT_SETTLE, S_WAIT_CRITICAL, S_LQR_ACTIVE);
73     signal state : t_state;
74
75     constant C_PERIOD_HOMING : natural := g_clk_hz / g_homing_freq_hz;
76
77     constant C_STEP_LIMIT : positive := 2 * g_center_offset_steps;
78     constant C_CNT_WIDTH  : positive := positive(ceil(log2(real(C_STEP_LIMIT + 1))));
79     signal step_cnt : unsigned(C_CNT_WIDTH - 1 downto 0);
80
81     constant C_SETTLE_TICKS : natural := g_settle_time_s * g_clk_hz;
82     constant C_SETTLE_CNT_W : positive := positive(ceil(log2(real(C_SETTLE_TICKS + 1))));
83     signal settle_cnt : unsigned(C_SETTLE_CNT_W - 1 downto 0);
84
85     signal count_offset_reg : signed(g_count_width - 1 downto 0);
86     signal count_rel : signed(13 downto 0);

```

```

87
88     constant C_DIR_LEFT  : std_logic := '0';
89     constant C_DIR_RIGHT : std_logic := '1';
90
91 begin
92
93     count_rel <= resize(enc_count_i, 14)
94         - resize(count_offset_reg, 14)
95         + to_signed(1200, 14);
96
97     count_offset_o <= count_offset_reg;
98
99     process (clk)
100     begin
101         if rising_edge(clk) then
102             if reset = '1' then
103                 state <= S_HOMING;
104                 step_cnt <= (others => '0');
105                 settle_cnt <= (others => '0');
106                 count_offset_reg <= (others => '0');
107             else
108                 case state is
109
110                     when S_HOMING =>
111                         if limit_fdc_i = '0' then
112                             state <= S_BACKING;
113                             step_cnt <= (others => '0');
114                         end if;
115
116                     when S_BACKING =>
117                         if step_tick_i = '1' then
118                             if step_cnt = g_center_offset_steps - 1 then
119                                 state <= S_WAIT_SETTLE;
120                                 settle_cnt <= (others => '0');
121                             else
122                                 step_cnt <= step_cnt + 1;
123                             end if;
124                         end if;
125
126                     when S_WAIT_SETTLE =>
127                         -- Subfase 1: timeout fijo g_settle_time_s (el operador coloca el pendulo)
128                         -- Subfase 2: espera a still_i='1' antes de capturar count_offset
129                         if settle_cnt < C_SETTLE_TICKS - 1 then
130                             settle_cnt <= settle_cnt + 1;
131                         elsif still_i = '1' then
132                             count_offset_reg <= enc_count_i;
133                             state <= S_WAIT_CRITICAL;
134                         end if;
135
136                     when S_WAIT_CRITICAL =>
137                         if (count_rel >= to_signed(-g_critical_counts, 14) and

```

```

138         count_rel <= to_signed( g_critical_counts, 14))
139     or
140     (count_rel >= to_signed(2400 - g_critical_counts, 14) and
141     count_rel <= to_signed(2400 + g_critical_counts, 14)) then
142     state <= S_LQR_ACTIVE;
143 end if;
144
145 when S_LQR_ACTIVE =>
146     if limit_fdc_i = '0' then
147         state <= S_HOMING;
148         step_cnt <= (others => '0');
149     elsif step_cnt = C_STEP_LIMIT - 1 then
150         state <= S_HOMING;
151         step_cnt <= (others => '0');
152     elsif (count_rel < to_signed(-g_safety_counts, 14) or
153     count_rel > to_signed( g_safety_counts, 14))
154         and
155         (count_rel < to_signed(2400 - g_safety_counts, 14) or
156         count_rel > to_signed(2400 + g_safety_counts, 14)) then
157         state <= S_WAIT_CRITICAL;
158     elsif step_tick_i = '1' then
159         if step_dir_i = C_DIR_RIGHT then
160             step_cnt <= step_cnt + 1;
161         elsif step_cnt > 0 then
162             step_cnt <= step_cnt - 1;
163         end if;
164     end if;
165
166 end case;
167 end if;
168 end if;
169 end process;
170
171 process (state)
172 begin
173     case state is
174
175         when S_HOMING =>
176             step_enable_o <= '1';
177             step_dir_o <= C_DIR_LEFT;
178             period_cycles_o <= to_unsigned(C_PERIOD_HOMING, g_period_width);
179             lqr_enable_o <= '0';
180
181         when S_BACKING =>
182             step_enable_o <= '1';
183             step_dir_o <= C_DIR_RIGHT;
184             period_cycles_o <= to_unsigned(C_PERIOD_HOMING, g_period_width);
185             lqr_enable_o <= '0';
186
187         when S_WAIT_SETTLE =>
188             step_enable_o <= '0';

```

```

189         step_dir_o      <= C_DIR_RIGHT;
190         period_cycles_o <= to_unsigned(C_PERIOD_HOMING, g_period_width);
191         lqr_enable_o     <= '0';
192
193         when S_WAIT_CRITICAL =>
194             step_enable_o <= '0';
195             step_dir_o    <= C_DIR_RIGHT;
196             period_cycles_o <= to_unsigned(C_PERIOD_HOMING, g_period_width);
197             lqr_enable_o  <= '0';
198
199         when S_LQR_ACTIVE =>
200             step_enable_o <= '0';
201             step_dir_o    <= C_DIR_RIGHT;
202             period_cycles_o <= to_unsigned(C_PERIOD_HOMING, g_period_width);
203             lqr_enable_o  <= '1';
204
205         end case;
206     end process;
207
208     with state select
209         state_o <= 000 when S_HOMING,
210                   001 when S_BACKING,
211                   010 when S_WAIT_SETTLE,
212                   011 when S_WAIT_CRITICAL,
213                   100 when S_LQR_ACTIVE,
214                   111 when others;
215
216 end architecture rtl;

```

Código 7: Maquina de estados de alto nivel del sistema (homing, centrado, asentamiento y control LQR).

lqr_system_top_v2.vhd

```

1  -- =====
2  -- TOP LEVEL: lqr_system_top
3  -- Sistema de control LQR del pendulo invertido sobre carro
4  -- Xilinx Artix-7 (Basys3) * 100 MHz * Ts = 1 ms * Punto fijo Q8.8
5  --
6  -- FLUJO COMPLETO DEL SISTEMA (fases de arranque + control):
7  --
8  --   RESET -> S_HOMING:          motor avanza hacia FdC (limit_fdc_i activo a nivel bajo)
9  --   -> S_BACKING:              motor retrocede 3 000 pasos; al llegar espera asentamiento
10 --   -> S_WAIT_SETTLE:          motor parado; subfase 1: timeout g_settle_time_s=8 s;
11 --                             subfase 2: espera still_i='1' (encoder quieto 1 s)
12 --   -> S_WAIT_CRITICAL:        espera |count_rel| <= 100 counts (operador eleva pendulo)
13 --   -> S_LQR_ACTIVE:           lqr_enable='1'; motor_command gobierna step_generator
14 --   * |count_rel| > 200         -> S_WAIT_CRITICAL

```

```

15  --          * limit_fdc_i='0'                      -> S_HOMING
16  --
17  -- Autor: Alicia Lopez Garcia
18  -- =====
19
20  library ieee;
21  use ieee.std_logic_1164.all;
22  use ieee.numeric_std.all;
23
24  entity lqr_system_top is
25      port (
26          clk : in std_logic;
27          rst : in std_logic;
28
29          enc_A : in std_logic;
30          enc_B : in std_logic;
31          limit_fdc_i : in std_logic;
32
33          step_out : out std_logic;
34          dir_out  : out std_logic;
35          motor_en : out std_logic;
36
37          tick_1ms      : out std_logic;
38          state_dbg     : out std_logic_vector(2 downto 0);
39          lqr_enable_dbg : out std_logic;
40          still_led_o    : out std_logic
41      );
42  end entity lqr_system_top;
43
44  architecture rtl of lqr_system_top is
45
46      component timer_1ms
47          generic (CLK_FREQ : integer := 100_000_000; Ts_US : integer := 1000);
48          port (clk : in std_logic; rst : in std_logic; tick : out std_logic);
49      end component;
50
51      component encoder_decoder
52          generic (BITS : integer := 12);
53          port (clk, rst, enc_A, enc_B : in std_logic; count : out signed(11 downto 0));
54      end component;
55
56      component lqr_system_fsm
57          generic (
58              g_clk_hz : positive := 100_000_000;
59              g_homing_freq_hz : positive := 5_000;
60              g_center_offset_steps : positive := 3_000;
61              g_period_width : positive := 24;
62              g_count_width : positive := 12;
63              g_critical_counts : positive := 100;
64              g_safety_counts : positive := 200;
65              g_settle_time_s : positive := 8

```

```
66     );
67     port (
68         clk, reset : in std_logic;
69         limit_fdc_i : in std_logic;
70         enc_count_i : in signed(11 downto 0);
71         step_tick_i, step_dir_i, still_i : in std_logic;
72         step_enable_o, step_dir_o : out std_logic;
73         period_cycles_o : out unsigned(23 downto 0);
74         count_offset_o : out signed(11 downto 0);
75         lqr_enable_o : out std_logic;
76         state_o : out std_logic_vector(2 downto 0)
77     );
78 end component;
79
80 component differentiator
81     port (
82         clk, rst, tick : in std_logic;
83         count : in signed(11 downto 0);
84         x3, x4 : out signed(15 downto 0)
85     );
86 end component;
87
88 component lqr_controller
89     port (
90         clk, rst, tick : in std_logic;
91         x1, x2, x3, x4 : in signed(15 downto 0);
92         u : out signed(15 downto 0)
93     );
94 end component;
95
96 component cart_estimator
97     port (
98         clk, rst, tick, step_tick, step_dir, lqr_enable : in std_logic;
99         x1, x2 : out signed(15 downto 0)
100     );
101 end component;
102
103 component simple_integrator
104     port (
105         clk, rst, tick : in std_logic;
106         u : in signed(15 downto 0);
107         x2 : out signed(15 downto 0)
108     );
109 end component;
110
111 component motor_command
112     generic (DIV_LATENCY : integer := 34; X2_MIN : integer := 7);
113     port (
114         clk, rst, tick : in std_logic;
115         x2_cmd : in signed(15 downto 0);
116         period : out unsigned(16 downto 0);
```

```
117         direction, enable : out std_logic
118     );
119 end component;
120
121 component step_generator
122     generic (CLK_FREQ : integer := 100_000_000; PERIOD_BITS : integer := 24;
123             PULSE_CICLOS : integer := 100);
124     port (
125         clk, rst, enable, direction : in std_logic;
126         period : in unsigned(23 downto 0);
127         step_out, dir_out, step_tick_o : out std_logic
128     );
129 end component;
130
131 component stillness_detector
132     generic (COUNT_BITS : integer := 12; CLK_FREQ_HZ : integer := 100_000_000; T_STILL_S
133             : integer := 1);
134     port (
135         clk, rst : in std_logic;
136         count_in : in signed(11 downto 0);
137         still_led : out std_logic
138     );
139 end component;
140
141 signal tick, tick_lqr : std_logic;
142 signal enc_count : signed(11 downto 0);
143 signal lqr_enable : std_logic;
144 signal count_offset : signed(11 downto 0);
145 signal fsm_state : std_logic_vector(2 downto 0);
146 signal fsm_step_en, fsm_step_dir : std_logic;
147 signal fsm_period : unsigned(23 downto 0);
148 signal step_tick, still : std_logic;
149 signal count_rel : signed(13 downto 0);
150 signal count_ang : signed(11 downto 0);
151 signal x1, x2, x3, x4, u, x2_cmd : signed(15 downto 0);
152 signal mot_period : unsigned(16 downto 0);
153 signal mot_dir, mot_enable : std_logic;
154 signal sg_period : unsigned(23 downto 0);
155 signal sg_dir, sg_enable : std_logic;
156
157 begin
158
159     tick_lqr <= tick and lqr_enable;
160
161     count_rel <= resize(enc_count, 14) - resize(count_offset, 14) + to_signed(1200, 14);
162
163     count_ang <= to_signed( 2047, 12) when count_rel > to_signed( 2047, 14) else
164         to_signed(-2048, 12) when count_rel < to_signed(-2048, 14) else
165         count_rel(11 downto 0);
166
167     -- Arbitraje: fases de arranque de la FSM vs. control LQR activo
```

```

166     sg_enable <= fsm_step_en or (mot_enable and lqr_enable);
167     sg_dir    <= mot_dir          when lqr_enable = '1' else fsm_step_dir;
168     sg_period <= resize(mot_period, 24) when lqr_enable = '1' else fsm_period;
169
170     U_TIMER : timer_1ms
171         generic map (CLK_FREQ => 100_000_000, Ts_US => 1000)
172         port map (clk => clk, rst => rst, tick => tick);
173
174     U_ENC : encoder_decoder
175         generic map (BITS => 12)
176         port map (clk => clk, rst => rst, enc_A => enc_A, enc_B => enc_B, count => enc_count
177             );
178
179     U_STILL : stillness_detector
180         generic map (COUNT_BITS => 12, CLK_FREQ_HZ => 100_000_000, T_STILL_S => 1)
181         port map (clk => clk, rst => rst, count_in => enc_count, still_led => still);
182
183     U_FSM : lqr_system_fsm
184         generic map (
185             g_clk_hz => 100_000_000, g_homing_freq_hz => 5_000,
186             g_center_offset_steps => 3_000, g_period_width => 24,
187             g_count_width => 12, g_critical_counts => 100,
188             g_safety_counts => 200, g_settle_time_s => 8
189         )
190         port map (
191             clk => clk, reset => rst,
192             limit_fdc_i => limit_fdc_i, enc_count_i => enc_count,
193             step_tick_i => step_tick, step_dir_i => sg_dir, still_i => still,
194             step_enable_o => fsm_step_en, step_dir_o => fsm_step_dir,
195             period_cycles_o => fsm_period, count_offset_o => count_offset,
196             lqr_enable_o => lqr_enable, state_o => fsm_state
197         );
198
199     U_ANG : differentiator
200         port map (clk => clk, rst => rst, tick => tick, count => count_ang, x3 => x3, x4 =>
201             x4);
202
203     U_LQR : lqr_controller
204         port map (clk => clk, rst => rst, tick => tick_lqr, x1 => x1, x2 => x2, x3 => x3, x4
205             => x4, u => u);
206
207     U_CART : cart_estimator
208         port map (
209             clk => clk, rst => rst, tick => tick,
210             step_tick => step_tick, step_dir => sg_dir,
211             lqr_enable => lqr_enable, x1 => x1, x2 => x2
212         );
213
214     U_INT : simple_integrator
215         port map (clk => clk, rst => rst, tick => tick_lqr, u => u, x2 => x2_cmd);

```

```
214     U_MCMD : motor_command
215         generic map (DIV_LATENCY => 34, X2_MIN => 3)
216         port map (
217             clk => clk, rst => rst, tick => tick, x2_cmd => x2_cmd,
218             period => mot_period, direction => mot_dir, enable => mot_enable
219         );
220
221     U_STEP : step_generator
222         generic map (CLK_FREQ => 100_000_000, PERIOD_BITS => 24, PULSE_CICLOS => 100)
223         port map (
224             clk => clk, rst => rst, enable => sg_enable, direction => sg_dir,
225             period => sg_period, step_out => step_out, dir_out => dir_out,
226             step_tick_o => step_tick
227         );
228
229     tick_1ms      <= tick;
230     state_dbg     <= fsm_state;
231     lqr_enable_dbg <= lqr_enable;
232     still_led_o   <= still;
233     motor_en      <= not sg_enable;
234
235 end architecture rtl;
```

Código 8: Nivel superior: instanciación e interconexión de todos los módulos del sistema.

Anexo E Simulaciones Vivado

En este anexo se incluyen las simulaciones de los testbench de los bloques del sistema implementado en VHDL que no se han incluido en la Sección 4. Todas ellas fueron realizadas con el simulador integrado de Vivado 2025.2, las *waveforms* obtenidas en cada simulación se muestran a continuación.

tb_encoder_decoder

En la Figura 42 se puede observar el comportamiento del decodificador del encoder rotativo, en la primera parte de la simulación el contador sube de 26 hasta 40 respondiendo a la secuencia de cuadratura de las señales `enc_a` y `enc_b` en sentido positivo. Sin embargo, cuando la secuencia se invierte hacia la mitad de la simulación el contador decrementa lo que verifica que la detección de dirección funciona correctamente.

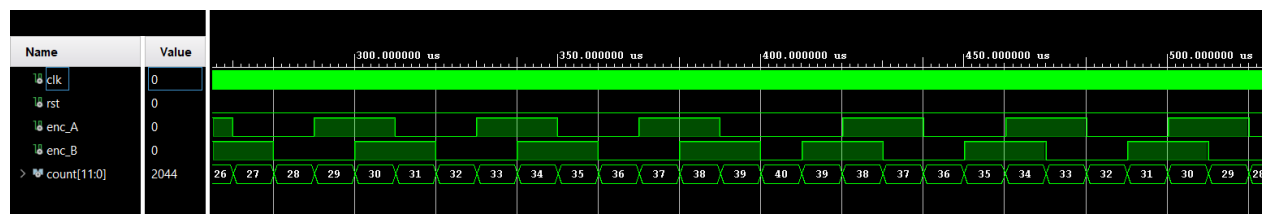


Figura 42: Simulación funcional del módulo `encoder_decoder`.

tb_stillness_detector

En la Figura 43 se observa la simulación para el bloque de detección de quietud, al inicio tras el reset síncrono `timer_reg` arranca desde 0 correctamente, tras el reset cuenta cíclicamente de 0 a 9, `still_led` se activa a nivel alto cuando `count_in` permanece con el mismo valor durante el número de ciclos establecido.

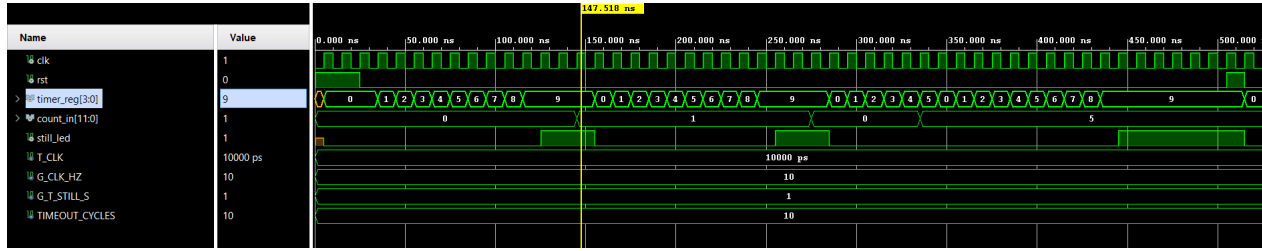


Figura 43: Simulación funcional del módulo `stillness_detector`.

tb_differentiator

Se verifica la conversión de cuentas del encoder a ángulo x_3 y velocidad angular x_4 en formato Q8.8. La Figura 44 muestra cómo `count=100` produce `x3=67`, coherente con $100 \times 172/256 \approx 67$ (constante $K_{ANG} = 172$ en Q0.16). La velocidad angular x_4 refleja la diferencia finita entre ticks, saturando correctamente a ± 32767 ante variaciones bruscas de `count`. Ambas salidas se actualizan sincronamente en cada flanco de `tick` con latencia de 1 ciclo.

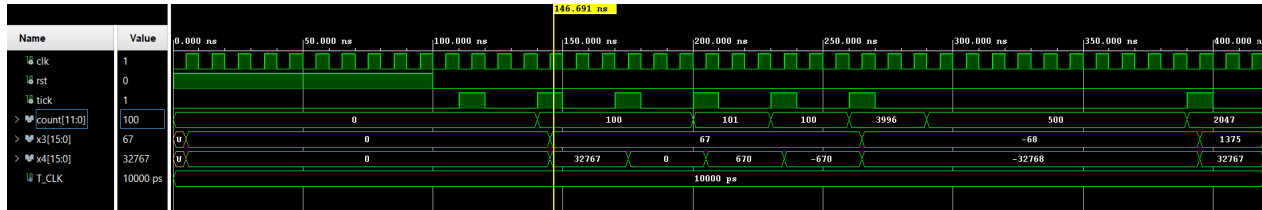


Figura 44: Simulación funcional del módulo `differentiator`.

tb_cart_estimator

Se verifica el cálculo de x_1 y x_2 a partir de pulsos STEP. En la Figura 45, con `step_dir=1`, `step_counter` incrementa correctamente y $x_2 = 512$ refleja la velocidad positiva. En la Figura 46, al invertir `step_dir=0`, el contador decrementa y x_2 toma el valor -129 , verificando el signo correcto. Al desactivar `lqr_enable`, `step_counter` se reinicia a cero.

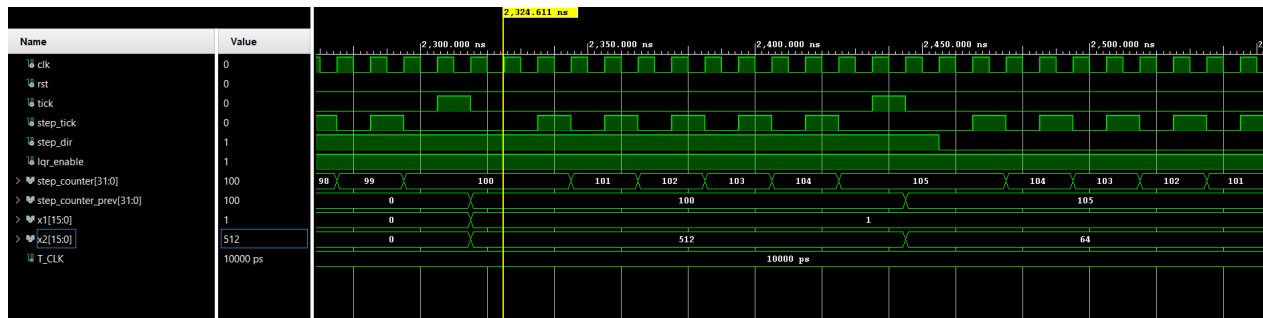


Figura 45: Simulación funcional del módulo `cart_estimator I`.

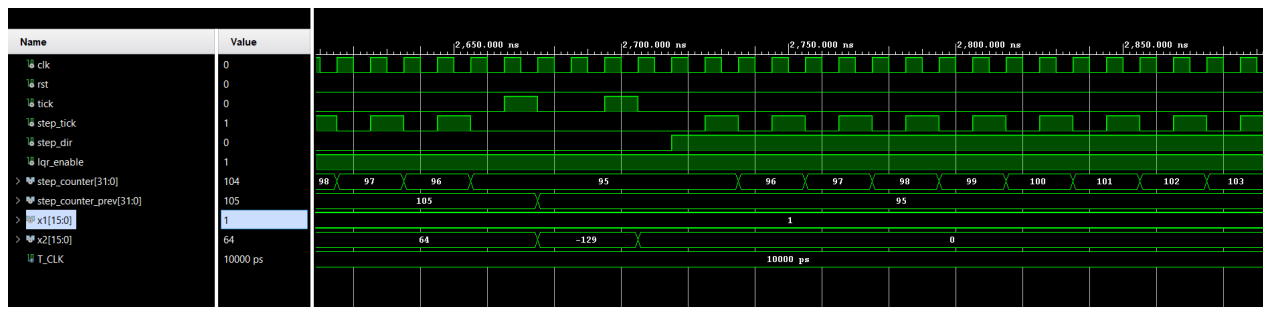


Figura 46: Simulación funcional del módulo `cart_estimator II`.

tb_simple_integrator

En la Figura 47 se verifica la integración discreta de Euler que realiza el bloque `simple_integrator`. La simulación muestra como x_2 incrementa acumulando el producto $u \cdot K_1$ confirmando el comportamiento esperado del integrador.

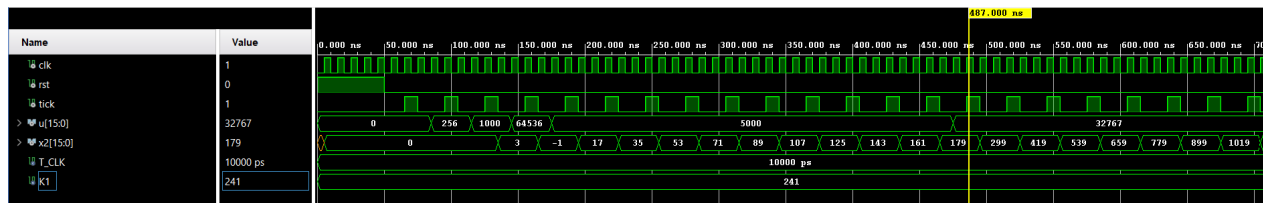


Figura 47: Simulación funcional del módulo `simple_integrator`.

tb_step_generator

La simulación del módulo `step_generator` se muestra en la Figura 48. Se observa que la generación de pasos es la esperada, con `period=1000` y `enable=1`, se generan pulsos pe-

riódicos. Cuando el periodo cambia, el tiempo entre pulsos lo hace también verificando el comportamiento correcto del bloque.

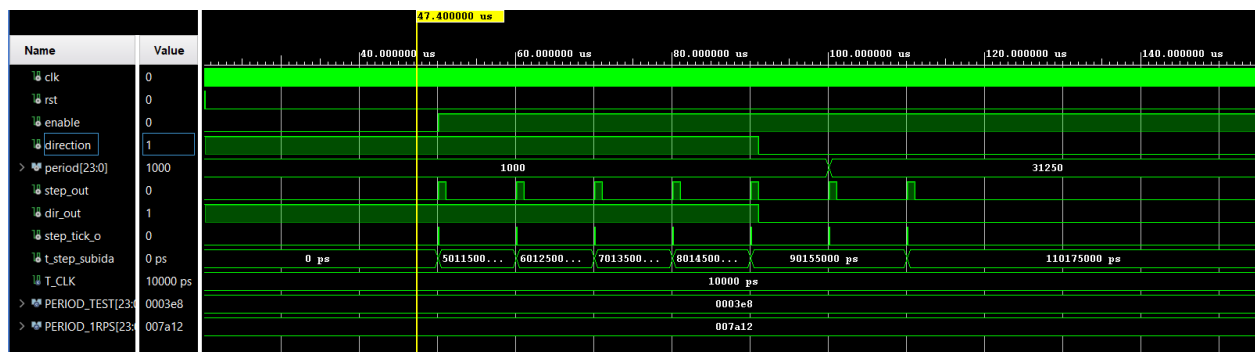


Figura 48: Simulación funcional del módulo `step_generator`.