UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

"DISEÑO DE UN SISTEMA ELECTRÓNICO DE CONTROL PARA ACCIONAR MOTORES PASO A PASO NEMA 34 MEDIANTE EL DRIVER DM860T EN APLICACIONES INDUSTRIALES"

TRABAJO FIN DE GRADO

Febrero –2026

AUTOR: FCO. JOSÉ HIDALGO GUILABERT

DIRECTOR/ES: DAVID MARROQUÍ SEMPERE

CRISTÓBAL PARRA SORIANO

Agradecimientos

Dr. David Marroquí Sempere / Profesor titular de la Universidad Miguel Hernández
Me gustaría agradecer toda la confianza que has depositado en mí desde el inicio del proyecto. Tu apoyo y orientación me han permitido crecer como estudiante y como profesional. Ha sido un placer trabajar juntos y aprender de tus conocimientos en el sector de la electrónica y del diseño hardware.

Quiero hacer una mención especial al laboratorio del SPES de la UMH, que me han facilitado un puesto de trabajo y todo el material necesario en sus instalaciones.

Cristóbal Parra Soriano / Head of Electrical Engineering
Quiero expresar mi agradecimiento a Cristóbal por proponerme este proyecto y por la confianza depositada en mí para su desarrollo. Su apoyo y visión han sido clave para mi crecimiento profesional. También agradezco a ***Vector Conveyors*** la cesión del material y los recursos facilitados, que han hecho posible la realización de este trabajo.

Por último, a mis amigos de la carrera, por compartir el camino cuando parecía más empinado, y por recordarme que no se avanza solo. Y sobre todo a mi familia, por ser raíz firme incluso cuando el viento soplaba con fuerza.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

ÍNDICE DE ILUSTRACIONES

1. Introducción.

En este primer capítulo del proyecto final se busca responder a la siguiente pregunta: *¿Por qué se ha llevado a cabo el proyecto?* Para ello, se expone la necesidad industrial que motiva el desarrollo del sistema, así como los objetivos que debe cumplir esta solución para garantizar una implementación técnica adecuada. Adicionalmente, el diseño debe estar alineado con el entorno y los objetivos de la empresa colaboradora.

En capítulos posteriores se describe **cómo se ha materializado la solución** planteada en este primer capítulo. Se detalla su diseño conceptual, qué materiales se utilizaron para su desarrollo, su implementación hardware-software y su validación.

DESTACADO

Esta memoria se ha organizado con el objetivo de exponer, en primer lugar, el motivo que origina el proyecto y, posteriormente, detallar las decisiones adoptadas durante el proceso de diseño e implementación del sistema electrónico.

1.1. Antecedentes y contexto industrial del proyecto.

La electrónica está presente en la práctica totalidad de los sectores que influyen en la sociedad. Es posible encontrar electrónica en ámbitos como el aeroespacial, las telecomunicaciones, la robótica, la computación o la inteligencia artificial, entre otros.

Dentro de este amplio abanico de aplicaciones, este trabajo se enmarca en el ámbito industrial. En concreto, se centra en el diseño de un producto electrónico destinado a automatizar el control de motores industriales empleados en cintas transportadoras.

INFORMACIÓN

El ámbito de desarrollo del proyecto guarda una estrecha relación con los estudios cursados por su autor en el Grado en Ingeniería Electrónica y Automática Industrial.

Este trabajo desarrolla un **sistema electrónico** aplicando conceptos de "*Ingeniería Electrónica*", para automatizar procesos industriales en lo referente al control de motores NEMA, garantizando la "*Automática Industrial*".

Vector Conveyors: empresa colaboradora con el proyecto.

Vector Conveyors es una empresa dedicada al diseño e implementación de sistemas de transporte y manipulación de materiales en entornos industriales o de producción. Su actividad se centra en el diseño, fabricación y montaje de sistemas mecánicos y electrónicos de transporte intralogístico, proporcionando soluciones personalizadas a sus clientes de diversos sectores como el industrial, el logístico y la manufactura.

Vector Conveyors colabora ocasionalmente con otras empresas en el desarrollo de soluciones de transporte, garantizando la estructura física y mecánica sobre la que se montan los diversos módulos funcionales diseñados y fabricados por terceros.

Un ejemplo de éxito colaborativo es el **Celluveyor**, desarrollado por *Cellumation GmbH,* un módulo innovador hexagonal que permite a los sistemas de transporte realizar movimientos omnidireccionales, capaz de desplazar productos en cualquier dirección (véase Ilustración 1.1 - Sistema Celluveyor y módulo hexagonal omnidireccional). En este caso, *Cellumation GmbH* se encarga del diseño modular y *Vector Conveyors* se encarga de su montaje en un sistema completo de transporte**.**

Ilustración 1.1 - Sistema Celluveyor y módulo hexagonal omnidireccional

DESTACADO

Esta colaboración pretende replicar el caso de éxito mencionado anteriormente. En lo que respecta al trabajo, se ha diseñado un módulo electrónico que resuelve las necesidades actuales de la empresa. El componente es replicable, por lo que permite desarrollar un sistema completo a partir de su duplicación. (véase Ilustración 1.2 - Esquema conceptual modular para un sistema completo de transporte).

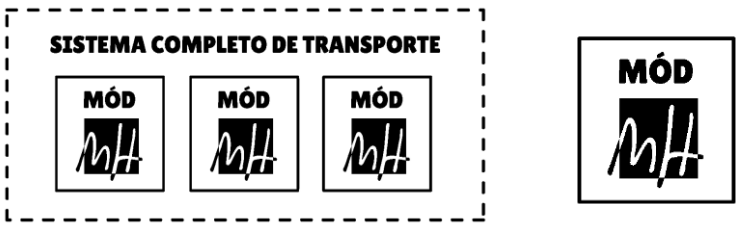


Ilustración 1.2 - Esquema conceptual modular para un sistema completo de transporte

1.2. Problemática detectada y motivación técnica.

En este apartado, se explica cuál es la aplicación industrial que desarrolla *Vector Conveyors* y sobre la que se ha identificado una posibilidad de automatización.

Dentro del campo de la manipulación y transporte de objetos en la industria, existe un subgrupo que permite la clasificación y desvío de objetos en sistemas intralogísticos. Este trabajo se centra en el **transfer de correas**, un mecanismo electromecánico de accionamiento vertical, utilizado para desviar objetos dentro de las líneas productivas, véase Ilustración 1.3 - Transfer de correas con identificación de sus principales elementos estructurales.

Ilustración 1.3 - Transfer de correas con identificación de sus principales elementos estructurales

Su funcionamiento se basa en el ajuste de la altura de unas bandas transportadoras integradas entre los rodillos de transporte, véase Ilustración 1.4 - Esquema funcional del transfer de correas y direcciones de movimiento. Para lograr este movimiento vertical, se emplean motores paso a paso de alta precisión en formato NEMA. Gracias a ellos, es posible direccionar la mercancía según:

- **Movimiento habitual:** accionado mediante las bandas transportadoras.
- **Movimiento perpendicular:** accionado mediante los rodillos de transporte.

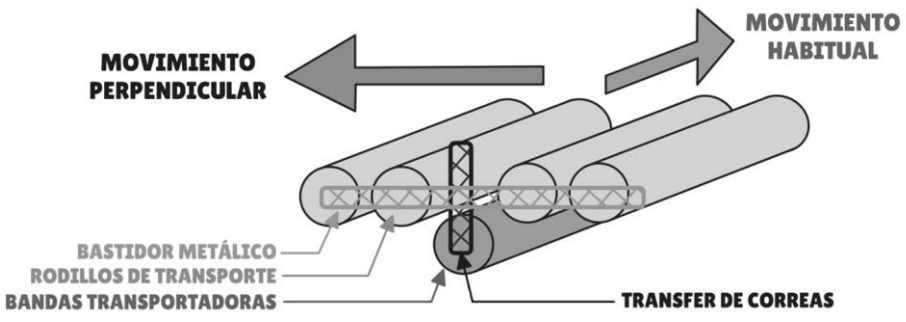


Ilustración 1.4 - Esquema funcional del transfer de correas y direcciones de movimiento.

Para facilitar la comprensión del funcionamiento del *transfer de correas*, se recurre a esquemas explicativos. Cuando las bandas transportadoras se sitúan por encima de los rodillos de transporte, la caja es impulsada por la dirección impuesta por dichas bandas, produciéndose el denominado **movimiento habitual**, véase Ilustración 1.5 - Funcionamiento del transfer de correas en modo de movimiento habitual.

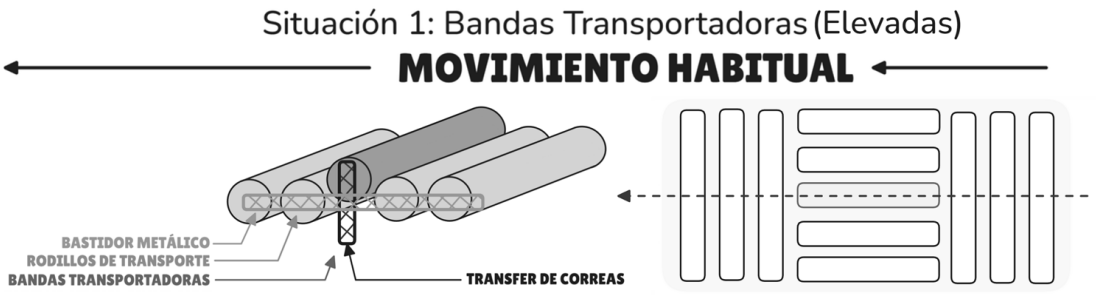


Ilustración 1.5 - Funcionamiento del transfer de correas en modo de movimiento habitual.

Cuando se desea clasificar un objeto, el sistema detiene provisionalmente las bandas transportadoras para que la caja quede posicionada en la parte superior de los rodillos de transporte a 90º. En ese instante, el transfer de correas se acciona para descender dichas bandas, hasta situarlas por debajo de los rodillos de transporte (situación 2).

De este modo, el objeto es desplazado con la dirección que imprimen los rodillos sobre los que la caja se sitúa en ese instante, generando lo que se conoce en aplicaciones industriales como **movimiento perpendicular clasificatorio** (véase Ilustración 1.6 - Funcionamiento del transfer de correas en modo de movimiento perpendicular).

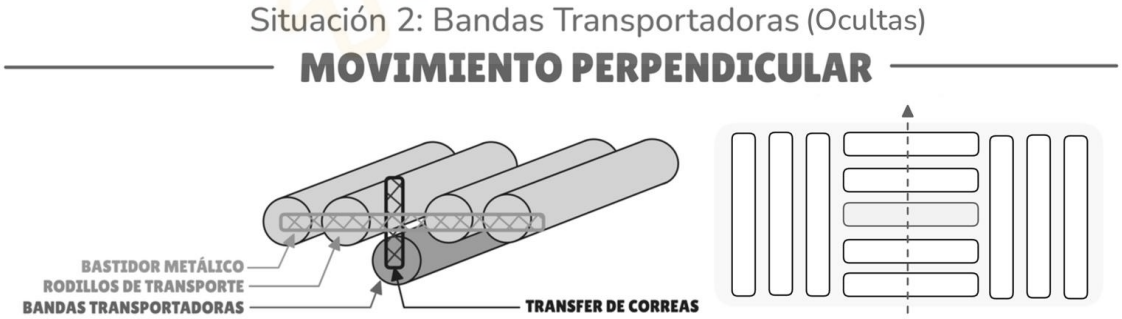


Ilustración 1.6 - Funcionamiento del transfer de correas en modo de movimiento perpendicular

En los esquemas anteriores se visualizan dos casos en los que el transfer de correas es necesario para realizar una aplicación industrial de clasificación de paquetería.

Tras la explicación de su funcionamiento mediante esquemas, se presenta un caso práctico de un sistema de clasificación a 90º mediante transfer de correas.

Este ejemplo práctico reproduce el funcionamiento descrito anteriormente, aplicado a un caso real. Se detalla el uso del transfer de correas en entornos industriales.

En primer lugar, se identifica el transfer de correas dentro del sistema de transporte, como elemento central del proceso logístico (véase Ilustración 1.7 - Identificación del transfer de correas en instalación real).

Ilustración 1.7 - Identificación del transfer de correas en instalación real

Si el objeto no debe clasificarse, las bandas transportadoras permanecen en una posición más elevada que los rodillos de transporte, por lo que la caja continúa su desplazamiento, denominado como **movimiento habitual** (véase Ilustración 1.8 – Funcionamiento del transfer de correas en situación 1).

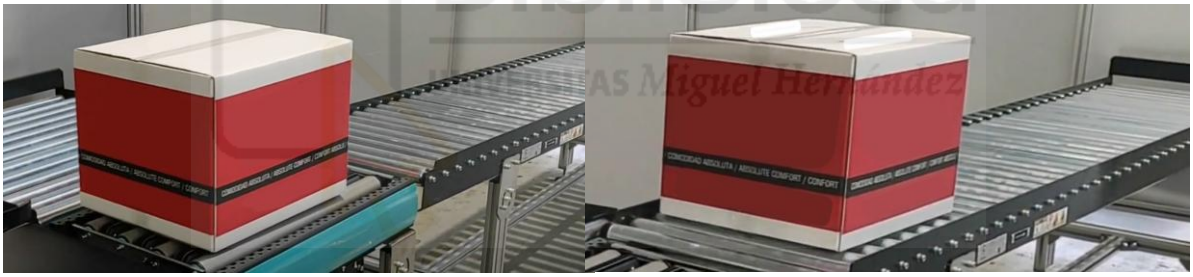


Ilustración 1.8 – Funcionamiento del transfer de correas en situación 1

Si el objeto debe clasificarse, el autómata detiene las bandas transportadoras hasta posicionar la caja sobre los rodillos de transporte, dispuestos perpendicularmente al flujo principal. En ese momento, el **transfer de correas** se activa para descender las bandas. Como consecuencia, el objeto pasa a ser impulsado por el **movimiento perpendicular** de los rodillos de transporte (véase Ilustración 1.9 – Funcionamiento del transfer de correas en situación 2).

Ilustración 1.9 – Funcionamiento del transfer de correas en situación 2

1.3. Propuesta técnica: Interfaz Electrónica de Control

Debido a que el sistema electromecánico es un elemento complejo, Vector Conveyors proporciona un documento técnico en el que se recogen de forma preliminar todos los requisitos y especificaciones del hardware que desean recibir para su desarrollo.

En referencia a capítulos anteriores, se pretende diseñar un componente electrónico que permita automatizar un proceso relacionado con un sistema mecánico, conocido como transfer de correas, véase Ilustración 1.3 - Transfer de correas con identificación de sus principales elementos estructurales.

INFORMACIÓN

El documento proporcionado por Vector Conveyors se encuentra disponible en los anexos de este proyecto, véase Anexo I – Bases y especificaciones técnicas.

En el documento, la empresa indica los motores de los que dispone en su taller para la construcción de maquinaria industrial. Actualmente, uno de los motores que desea automatizar la compañía es un motor sin escobillas de DC con driver integrado, con un precio aproximado de 140.97 €, disponible en el proveedor Stepper-online, véase a continuación una representación mediante la Ilustración 1.10 - Motor BLDC .

Ilustración 1.10 - Motor BLDC con driver integrado

Tras el análisis de la información recibida, se definen una serie de objetivos que debe cumplir el sistema electrónico para la validación del diseño por parte de la empresa.

> **El presente Trabajo Fin de Grado pretende resolver las directrices indicadas en el Objetivo I del documento Anexo I – Bases y especificaciones técnicas**

Con esta información, el proyecto cuenta con un manual de instrucciones para el usuario que recoge el funcionamiento esperado del dispositivo. Este manual resulta de utilidad para las capacidades del producto y la verificación del funcionamiento previsto. Dicho manual de instrucciones se encuentra disponible en los Anexos del presente proyecto, véase Anexo II – Manual de instrucciones.

1.4. Objetivos generales y específicos del sistema

Objetivos generales del proyecto

Los objetivos generales enmarcan la finalidad última del proyecto, orientada a resolver una necesidad industrial mediante diseño hardware y software:

- **Diseñar, fabricar y validar un producto electrónico** que actúe como interfaz electrónica de control para motores en entornos industriales.
- **Automatizar el accionamiento y control cinemático** de los sistemas electro-mecánicos industriales, enfocándose específicamente en su aplicación para módulos de transferencia mecánica (transfer de correas).
- **Dotar al sistema de autonomía operativa** mediante la generación de señales digitales, eliminando la dependencia de utilizar autómatas programables (PLC) para tareas de procesamiento y control.
- **Reducir la dependencia tecnológica y económica** frente a suministradores externos de drivers para motores industriales, creando una solución a medida que se integre de forma directa y eficiente con los motores de la planta.

Objetivos específicos del proyecto

Los objetivos específicos que se detallan a continuación han sido definidos de forma iterativa a lo largo de las reuniones de seguimiento mantenidas con la empresa. Estas metas, derivadas de los objetivos generales del proyecto, acotan el alcance técnico del diseño y establecen los requisitos funcionales exactos que deben cumplir el hardware y el software desarrollado para satisfacer la demanda industrial:

- **Sintetizar señales digitales de control:** Generar las señales de temporización y control con la frecuencia y latencia exigidas por el driver industrial de estudio DM860T para un funcionamiento fluido del motor.
- **Integración y acondicionamiento de sensores:** Adaptar el hardware para el procesamiento de forma segura de las lecturas provenientes de un sensor de tipo inductivo externo, permitiendo al sistema reaccionar ante eventos.
- **Gestión del sentido de giro:** Implementar el control bidireccional del motor mediante la lectura y procesado de un selector físico integrado en la interfaz.
- **Gestión de la alimentación del sistema:** Incorporar un pulsador para el encendido y apagado que controle de forma segura el motor.
- **Diseño de una arquitectura de modos de operación:** Programar el núcleo del microcontrolador utilizando técnicas software para permitir alternar de forma segura entre distintos modos de operación: Run, Sleep, Wait.
- **Persistencia de datos del sistema:** Garantizar que el dispositivo recuerde su última configuración (como el modo de operación actual) almacenando estos datos en la memoria no volátil del microcontrolador (EEPROM).
- **Regulación de velocidad en tiempo real**: Permitir el ajuste de la velocidad del motor mediante la lectura de un encoder rotativo dispuesto en la propia interfaz.

2. Materiales y Métodos.

En este apartado se describe el conjunto de herramientas, tanto físicas como digitales, empleadas en el desarrollo del proyecto VCON-PULSE-LINK. Se detallan las distintas plataformas y aplicaciones software utilizadas para el diseño del prototipo, así como los materiales y equipos, tanto del laboratorio como de la planta industrial, utilizados para la verificación y validación del sistema funcional.

Asimismo, se exponen algunas de las tecnologías disponibles en el mercado para la ejecución de tareas de producción, analizando sus ventajas e inconvenientes con el fin de justificar la selección de las herramientas empleadas en el proyecto.

DESTACADO

El objetivo de este apartado es detallar todos los recursos técnicos que han permitido materializar la propuesta técnica presentada en la sección introductoria anterior.

2.1. Equipos y materiales empleados en el proyecto

Los materiales empleados en el proyecto comprenden el conjunto de componentes y equipos de carácter industrial empleados para la verificación completa del sistema.

Estos elementos han permitido reproducir en un entorno controlado todas las pruebas, simulando las condiciones del sistema en un entorno real. De este modo, se valida el diseño sin comprometer la integridad de una instalación de mayor coste, como es el transfer de correas. A continuación, se desglosan todos los elementos que han intervenido en estas pruebas del diseño:

2.1.1. Driver industrial para motor paso a paso

Definición: un driver industrial para motor paso a paso es un dispositivo electrónico de potencia y control encargado de excitar el motor mediante la regulación de corriente en sus bobinados, a partir de señales digitales de mando externas.

Driver empleado: en el presente proyecto, se ha utilizado un driver industrial DM860T, empleado durante las pruebas de funcionamiento del dispositivo electrónico.

Justificación de la elección: se ha seleccionado el driver industrial DM860T debido a que es el sistema que proporciona la empresa para control. Este driver se considera objeto de estudio para las pruebas, por lo que debe usarse ese modelo específico.

Véase Ilustración 2.1 – Driver digital DM860T: vista real y esquema técnico del dispositivo para obtener una representación gráfica sobre el dispositivo.

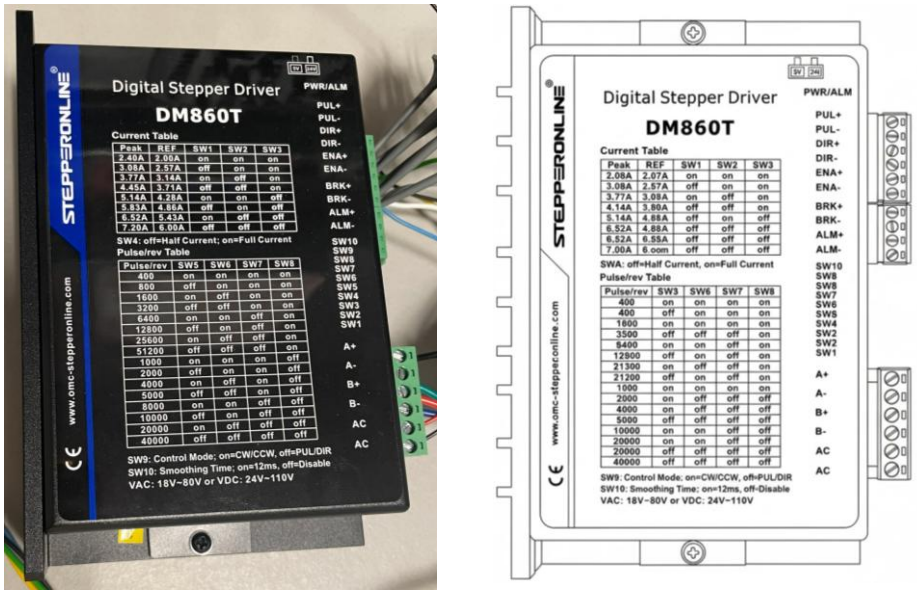


Ilustración 2.1 – Driver digital DM860T: vista real y esquema técnico del dispositivo

2.1.2. Motor paso a paso industrial

Definición: un motor paso a paso es un actuador electro-magnético que transforma señales eléctricas discretas en movimiento angular controlado, de forma que el eje del motor (rotor) avanza una cantidad fija por cada impulso eléctrico recibido.

A diferencia de otros motores eléctricos de giro continuo, el motor paso a paso no gira libremente al aplicarle tensión, sino que su movimiento es discreto. Esta característica permite obtener un control preciso de la posición, la velocidad y el sentido de giro.

Este motor resulta especialmente adecuado en aplicaciones en las que se requiere un posicionamiento preciso, ya que cada impulso eléctrico enviado se corresponde con un desplazamiento angular fijo. Véase Ilustración 2.2 – Motor paso a paso NEMA 34 para obtener una representación más detallada.

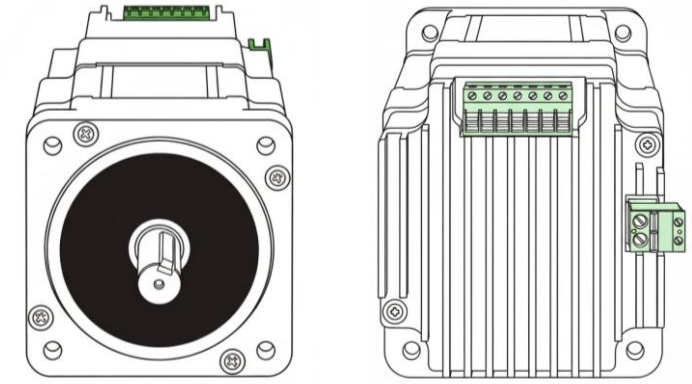


Ilustración 2.2 – Motor paso a paso NEMA 34

Desde el punto de vista constructivo, un motor paso a paso está formado por un conjunto de elementos electro-mecánicos. Algunos de ellos son:

- **Front Cover (Tapa Frontal)**
  Elemento mecánico que cierra el conjunto por la parte frontal del motor y le proporciona soporte estructural al eje de salida y a los rodamientos.
- **Screw (Tornillería)**
  Conjunto de tornillos para unir las distintas partes mecánicas del motor, con el objetivo de asegurar la rigidez estructural necesaria.
- **Bearing (Rodamiento)**
  Componente mecánico que permite el giro del eje con un rozamiento mínimo, soportando cargas axiales y radiales, garantizando un movimiento suave.
- **Central Axis (Eje Central)**
  Eje solidario al rotor que transmite el movimiento rotatorio generado hacia el exterior del motor, permitiendo su acoplamiento con otros elementos.
- **Rotor Core (Núcleo del rotor)**
  Parte móvil formada por un núcleo ferromagnético dentado, que interactúa con los campos magnéticos generados por el estator para producir el movimiento.
- **Al-Ni-Co (Aleación)**
  Imán permanente integrado en el rotor (en motores híbridos), encargado de reforzar el campo magnético del conjunto y mejorar el par o estabilidad.
- **Stator Module (Módulo del estator)**
  Parte fija del motor que alberga las bobinas distribuidas por fases. Al excitarse eléctricamente de forma secuencial, generan campos magnéticos.
- **Rear Cover**
  Elemento de cierre posterior del motor que protege el conjunto interno y contribuye a la rigidez estructural, además de servir como soporte.

Todos estos elementos del motor paso a paso se visualizan en el conjunto explotado, véase Ilustración 2.3 – Vista explotada del motor paso a paso.

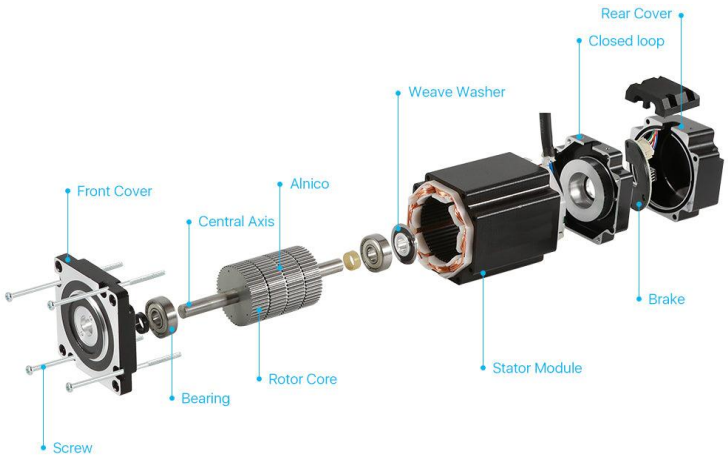


Ilustración 2.3 – Vista explotada del motor paso a paso

En cuanto al funcionamiento interno del motor paso a paso, este dispone de bobinas distribuidas en fases, siendo habitual la existencia de dos fases denominadas **fase A** (bobina A–A') y **fase B** (bobina B–B'). Cada fase está formada por devanados situados en polos opuestos del estator y conectados entre sí.

Al excitar eléctricamente una de estas fases, se genera un **campo magnético** en el estator que provoca la alineación del rotor con dicho campo, adoptando una posición estable determinada. Si posteriormente se modifica la fase excitada, para realizar un nuevo movimiento, el campo magnético resultante cambia de orientación, obligando al rotor a desplazarse hasta una nueva posición.

De este modo, el movimiento del motor no es continuo, sino que se produce mediante incrementos angulares fijos, denominados pasos. Cada cambio en la secuencia de excitación de las fases provoca un desplazamiento angular determinado del rotor. Esto permite controlar con precisión la posición y la velocidad del eje simplemente con la regulación y la frecuencia de los pasos aplicados (véase Ilustración 2.4 – Secuencia de excitación de fases en un motor paso a paso).

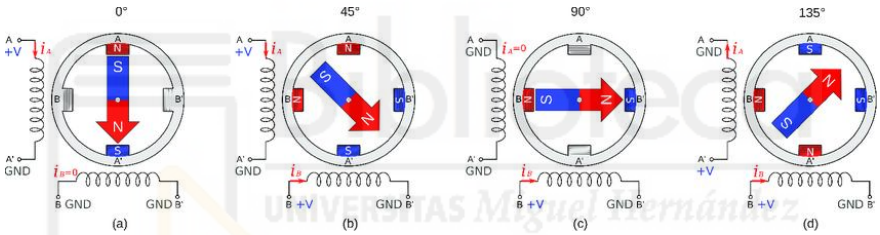


Ilustración 2.4 – Secuencia de excitación de fases en un motor paso a paso

En el presente proyecto, se ha utilizado un motor paso a paso bipolar NEMA 34 (Serie E), durante las pruebas. Véase Ilustración 2.5 – Motor paso a paso NEMA 34 sin driver integrado.

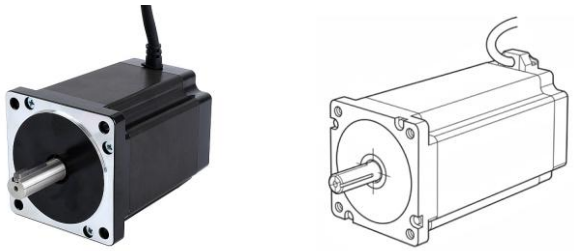


Ilustración 2.5 – Motor paso a paso NEMA 34 sin driver integrado

2.1.3. Sensor inductivo de proximidad

Definición: un sensor inductivo de proximidad es un dispositivo electrónico capaz de detectar la presencia de objetos metálicos sin contacto físico, mediante la generación de un campo electromagnético.

Se ha empleado un sensor inductivo, BES060U de Balluff, con código de producto equivalente a **BES M12ZI-PSC40B-S04G**.

El sensor inductivo presenta codificación A, siguiendo el estándar industrial para los conectores de tipo M12. Esta codificación define una asignación normalizada de pines, utilizada ampliamente en sensores de proximidad.

Además, el sensor dispone de una salida PNP de tipo normalmente abierta (NO), lo que implica que, al detectar un objeto metálico, la salida del sensor se conecta a la tensión de alimentación del sistema (+VCC).

Para más información sobre las dimensiones del dispositivo, véase Ilustración 2.6 – Sensor inductivo PNP con conexión M12.

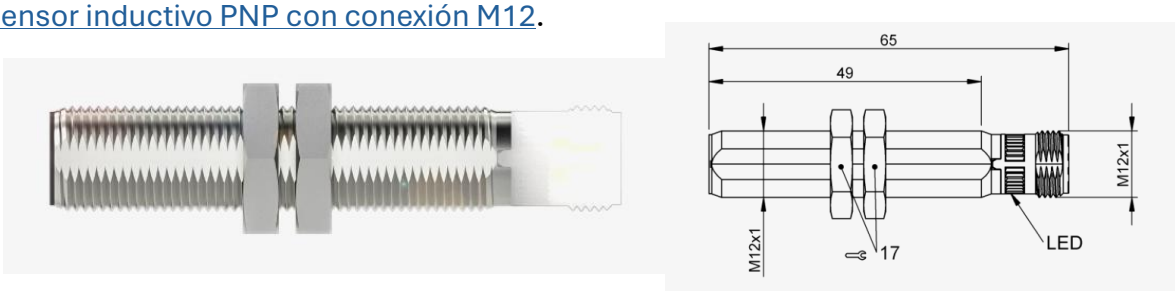


Ilustración 2.6 – Sensor inductivo PNP con conexión M12

2.2. Herramientas de diseño electrónico y CAD (EDA & CAD Tools).

Las herramientas de diseño electrónico asistido por ordenador, conocidas como EDA (Electronic Design Automation), constituyen el conjunto de aplicaciones de software para diseño, análisis y validación de los sistemas electrónicos modernos.

Su uso permite a los ingenieros la automatización de tareas repetitivas en el proceso de diseño hardware. Estas herramientas generan múltiples archivos de fabricación como Gerber, BOM y Pick & Place, entre otros, por lo que el entorno de desarrollo dispone de funcionalidades tanto para el diseño del layout como para el diseño digital.

En el contexto de este proyecto, estas aplicaciones de diseño hardware son esenciales para desarrollar la propuesta técnica definida en la sección de introducción. Sin estas herramientas no sería posible el desarrollo de la solución electrónica.

Una correcta selección de la herramienta EDA/CAD influye directamente en la calidad del diseño final, sobre todo a la hora del montaje y supervisión del prototipo diseñado.

DESTACADO

Al automatizar tareas repetitivas, estas herramientas aceleran el proceso de diseño, permitiendo a los ingenieros centrarse en los aspectos más críticos y reducir el tiempo de comercialización. También ayudan a identificar problemas con antelación y cumplir con los estándares de la industria, cruciales para la certificación y homologación. [1]

La automatización del diseño electrónico (EDA) mediante herramientas de software es necesaria para gestionar la complejidad de los sistemas electrónicos modernos, que a menudo involucran millones de componentes. [1]

En el ámbito del diseño electrónico existen diversas herramientas EDA/CAD utilizadas tanto en entornos académicos como profesionales. Estas soluciones difieren sobre todo en su modelo de licencia, facilidad de uso y capacidades técnicas.

A continuación, se presentan algunos de los programas más importantes del mercado, analizando de forma general sus características en lo referente a usabilidad del SW.

KiCad es una suite gratuita (open-source) para la automatización del diseño electrónico (EDA). Ofrece captura de esquemas, simulación de circuitos integrados, diseño de placas de circuito impreso (PCB), renderizado 3D y exportación del trazado a numerosos formatos. [2]

Incluye una biblioteca de componentes de calidad con miles de símbolos, huellas y modelos 3D. KiCad tiene requisitos de sistema mín. y es compatible con Linux, Windows y macOS. [2]

Para visualizar un ejemplo de la interfaz que presenta el programa, véase Ilustración 2.7 - Entorno de trabajo de KiCad: renderizado 3D y editor de PCB.

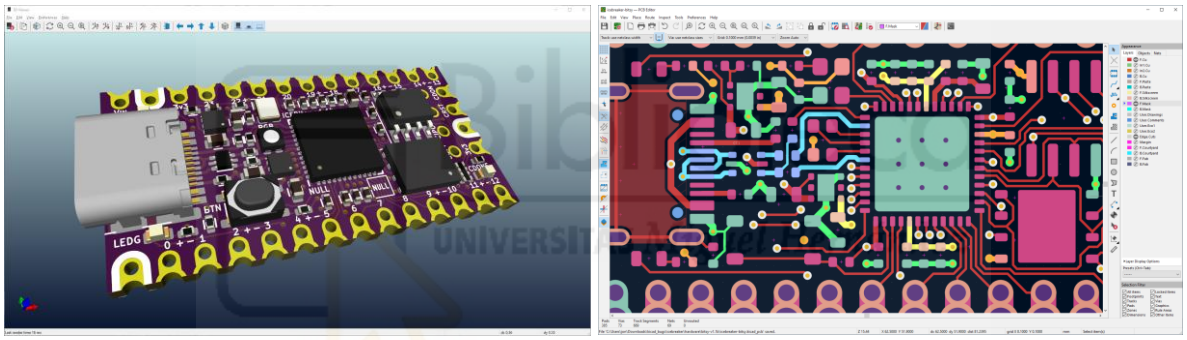


Ilustración 2.7 - Entorno de trabajo de KiCad: renderizado 3D y editor de PCB

Evaluación de la herramienta KiCad en el contexto del proyecto

A continuación, se exponen los principales criterios para la evaluación:

- **Usabilidad (Media)** – Ofrece un entorno de trabajo completo y potente, aunque requiere de un periodo inicial de aprendizaje, especialmente en el diseño del hardware y en la gestión de la biblioteca de componentes y huellas.
- **Licencia (Gratuita) –** Se trata de una herramienta de código abierto sin coste de licencia, lo que permite su uso sin limitaciones funcionales en entornos de tipo académico y/o profesional, muy válido para proyectos sin presupuesto.
- **Adecuación al proyecto (Alta) –** Dispone de las funcionalidades necesarias para el desarrollo del prototipo electrónico, incluyendo diseño de la PCB de tipo multicapa y generación de archivos de fabricación estándar.

Véase **Tabla 2.1 - Criterios de evaluación KiCad**.

Tabla 2.1 - Criterios de evaluación KiCad

EDA TOOL	Usabilidad	Licencia	Adecuación
KiCad	Media	Gratuita	Alta

Altium Designer es una solución integral diseñada para todo el espectro de proyectos de PCB, desde un circuito básico hasta complejos sistemas multiplaca. [3]

Cuenta con las herramientas adecuadas para gestionar: reglas de diseño, enrutamiento más avanzado, impedancia controlada, etc. Altium ahorra costes adicionales en el proceso de desarrollo, facilitando el trabajo colaborativo entre varios diseñadores hardware. [3]

Para visualizar un ejemplo de la interfaz que presenta el programa, véase Ilustración 2.8 – Entorno del editor PCB en Altium Designer

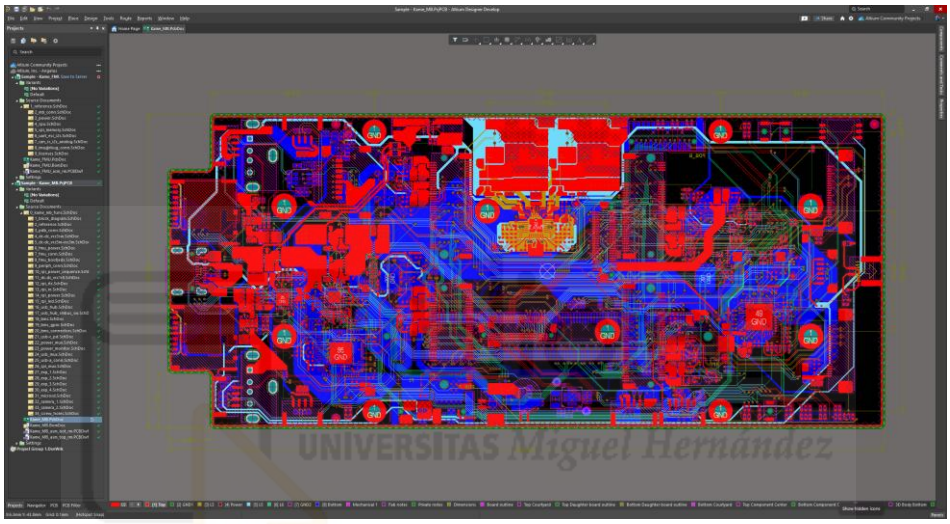


Ilustración 2.8 – Entorno del editor PCB en Altium Designer

Evaluación de la herramienta Altium en el contexto del proyecto

A continuación, se exponen los principales criterios para la evaluación:

- **Usabilidad (Alta)** – Entorno de trabajo altamente integrado e intuitivo para todo tipo de usuarios con experiencia en diseño electrónico. Facilita la gestión de proyectos complejos entre varios participantes, lo que es una gran ventaja.
- **Licencia (Comercial) –** Se trata de una herramienta profesional con licencia de pago, lo que implica un coste elevado en comparación con soluciones open source. Supone una limitación en entornos donde el presupuesto es reducido.
- **Adecuación al proyecto (Media) –** Aunque ofrece prestaciones avanzadas y un alto nivel de control del diseño, sus capacidades superan las necesidades del proyecto con creces, por lo que es demasiado potente para el mismo.

Véase **Tabla 2.2 - Criterios de evaluación Altium**.

Tabla 2.2 - Criterios de evaluación Altium

EDA TOOL	Usabilidad	Licencia	Adecuación
Altium	Alta	Comercial	Media

AUTODESK® EAGLE

EAGLE es un software de automatización de diseño electrónico (EDA) que permite a los diseñadores de placas de circuito impreso (PCB) conectar sin problemas diagramas esquemáticos, ubicación de componentes y enrutamiento de la PCB. [4]

Para visualizar un ejemplo de la interfaz que presenta el programa, véase Ilustración 2.9 – Entorno de desarrollo de Eagle: esquema, editor PCB y vista 3D.

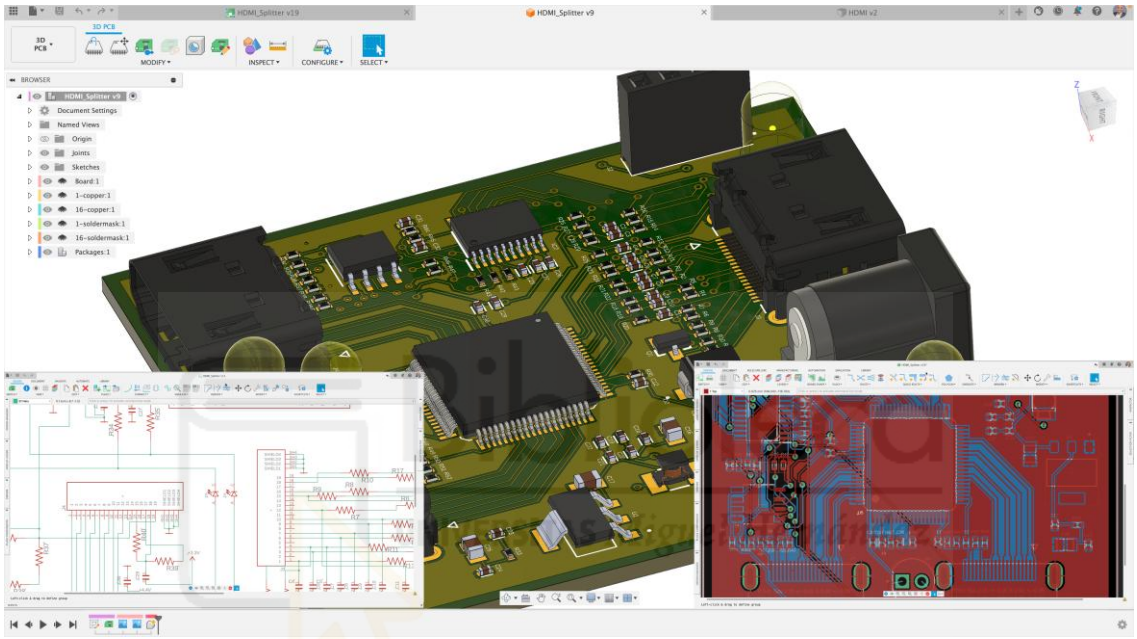


Ilustración 2.9 – Entorno de desarrollo de Eagle: esquema, editor PCB y vista 3D

Evaluación de la herramienta Eagle en el contexto del proyecto

A continuación, se exponen los principales criterios para la evaluación:

- **Usabilidad (Media)** – Presenta una interfaz relativamente intuitiva y orientada al prototipado rápido, lo que facilita su uso en proyectos de pequeña y mediana complejidad. Sus limitaciones funcionales dependen del tipo de licencia.
- **Licencia (Comercial) –** Herramienta de pago integrada en Autodesk. También pueden adquirirse versiones educativas con determinadas limitaciones.
- **Adecuación al proyecto (Media) –** Aunque permite desarrollar esquemáticos y PCB completamente funcionales, Eagle dispone de módulos y servicios de pago que pueden limitar el acceso completo a la herramienta.

Véase **Tabla 2.3 - Criterios de evaluación para Eagle**.

Tabla 2.3 - Criterios de evaluación para Eagle

EDA TOOL	Usabilidad	Licencia	Adecuación
Eagle	Media	Comercial	Media

Tras el análisis y evaluación de las diferentes herramientas EDA/CAD disponibles en el mercado, **se ha seleccionado KiCad** como herramienta principal para el desarrollo del proyecto. Esta decisión se ha basado en los **criterios de evaluación** citados (véase Tabla 2.4 - Criterios de evaluación para seleccionar EDA).

Tabla 2.4 - Criterios de evaluación para seleccionar EDA

EDA TOOL	Usabilidad	Licencia	Adecuación
KiCad	Media	Gratuita	Alta
Altium	Alta	Comercial	Media
Eagle	Media	Comercial	Media

2.3. Herramientas de desarrollo software (IDEs & Utilities)

Las herramientas de desarrollo software y las Utilities (o utilidades) son el conjunto de aplicaciones software empleadas para la creación, compilación, carga y verificación del software/firmware que gobierna el funcionamiento del sistema electrónico.

Dentro de este grupo de aplicaciones para la producción del software se incluyen: los entornos de desarrollo integrados (IDEs) y diversas utilidades auxiliares. Proporcionan un entorno unificado que integra: editor de código, compilador y algunas herramientas para depuración (según el sistema embebido seleccionado).

En el contexto del presente proyecto, las aplicaciones de producción software han sido fundamentales para implementar la lógica de control del sistema. Se ha desarrollado un firmware capaz de interactuar con el hardware utilizando dichas herramientas.

DESTACADO

Un ejemplo de utilidad empleada en este proyecto son las CLI, también conocidas como “Command Line Interface”. Este tipo de aplicaciones se ejecutan desde la terminal y permiten interactuar con el sistema operativo mediante comandos.

En este proyecto se ha utilizado el comando tree, que se ejecuta desde la interfaz de línea de comandos y permite visualizar de forma jerárquica la estructura de directorios de una ruta dada en forma de árbol (véase Ilustración 2.10 - Visualización jerárquica de directorios mediante el comando tree en CLI).

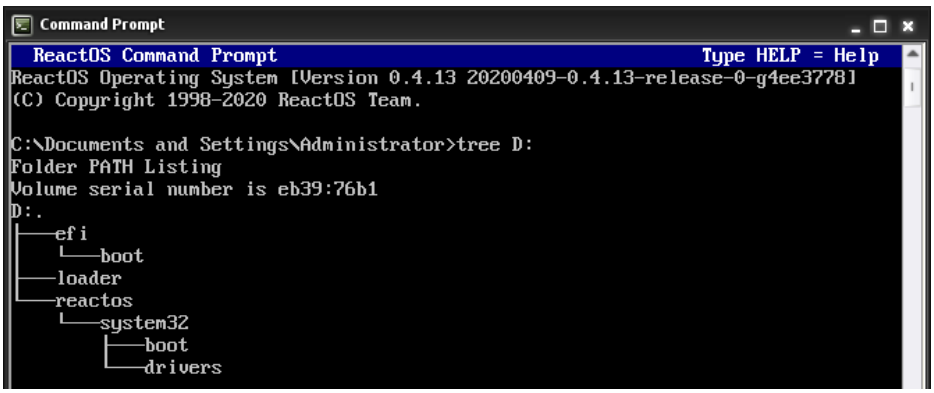


Ilustración 2.10 - Visualización jerárquica de directorios mediante el comando tree en CLI

Visual Studio Code, al que también se conoce como VSCode, es un editor de código para programadores gratuito, de código abierto y multiplataforma. Está desarrollado por Microsoft, una compañía con mucha experiencia en la creación de IDEs. [5]

Para visualizar un ejemplo de la interfaz que presenta el programa, véase Ilustración 2.11 - Interfaz del entorno de desarrollo Visual Studio Code.

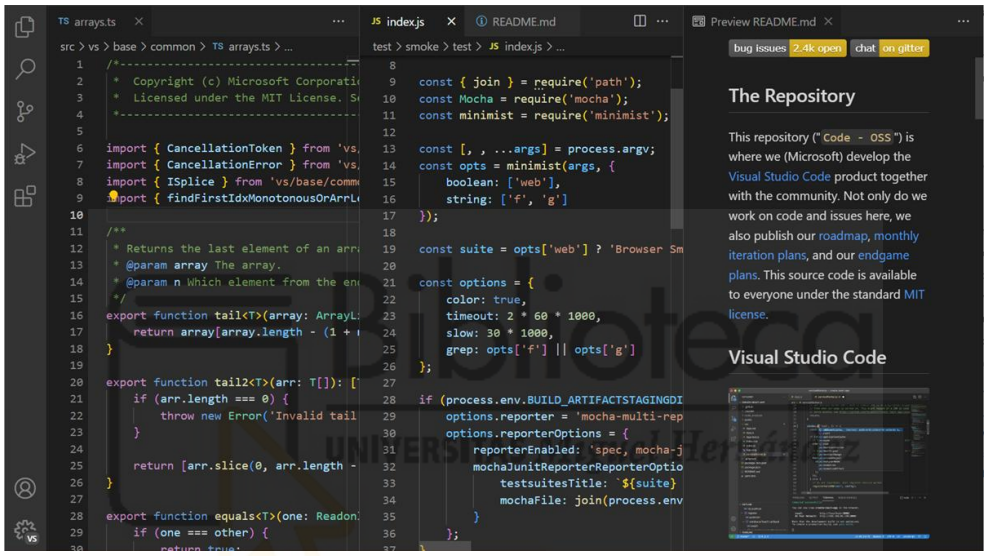


Ilustración 2.11 - Interfaz del entorno de desarrollo Visual Studio Code

Evaluación de la herramienta VSCode en el contexto del proyecto

A continuación, se exponen los principales criterios para la evaluación:

- **Usabilidad (Alta)** – Proporciona un entorno de desarrollo moderno, flexible y altamente configurable mediante diversas extensiones. Aunque su curva de aprendizaje es mayor que la de entornos más simples, es mucho más potente.
- **Licencia (Gratuita) –** Se trata de una herramienta de código abierto, gratuita y multiplataforma, ampliamente adoptada en entornos puramente académicos y profesionales. Permite uso sin restricciones funcionales para proyectos.
- **Adecuación al proyecto (Alta) –** Resulta adecuada como entorno principal de desarrollo, ya que permite integrar herramientas específicas para sistemas embebidos, además de contar con Copilot, IA que ayuda a desarrollar código.

Véase Tabla 2.5 - Criterios de evaluación para VSCode.

Tabla 2.5 - Criterios de evaluación para VSCode

TOOL	Usabilidad	Licencia	Adecuación
VSCode	Alta	Gratuita	Alta

Arduino IDE es un conjunto de herramientas software que permiten a los programadores desarrollar y grabar todo el código necesario para hacer que Arduino funcione. [6]

El IDE de Arduino permite escribir, depurar, editar y grabar programas (llamados «sketches» en el mundo Arduino) de una manera sumamente sencilla. [6]

Para visualizar un ejemplo de la interfaz que presenta el programa, véase Ilustración 2.12 - Entorno de desarrollo del IDE Arduino.

Ilustración 2.12 - Entorno de desarrollo del IDE Arduino

Evaluación de la herramienta IDE Arduino en el contexto del proyecto

A continuación, se exponen los principales criterios para la evaluación:

- **Usabilidad (Alta)** – Presenta un entorno de desarrollo sencillo, orientado al desarrollo de firmware para microcontroladores. Su curva de aprendizaje es reducida, lo que facilita una rápida puesta en marcha del entorno de trabajo.
- **Licencia (Gratuita) –** Se trata de una herramienta de código abierto y gratuita, sin restricciones funcionales, adecuada para entornos académicos y proyectos de carácter personal.
- **Adecuación al proyecto (Alta) –** Resulta apropiada para el desarrollo del firmware debido a su estructura basada en *sketches* y a su enfoque orientado al prototipado, alineado con los objetivos y el alcance del proyecto.

Véase Tabla 2.6 - Criterios de evaluación Arduino.

Tabla 2.6 - Criterios de evaluación Arduino

TOOL	Usabilidad	Licencia	Adecuación
Arduino	Alta	Gratuita	Alta

Ambas herramientas son adecuadas para el proyecto. Por un lado, el IDE de Arduino ofrece una capa de abstracción hardware junto con un entorno de desarrollo potente. En el otro extremo, se encuentra VSCode que, si bien no es una herramienta específica para diseño de firmware, dispone de varias extensiones para la producción de código más eficiente (*PlatformIO* ofrece una extensión en VSCode para firmware).

Por todo ello, se ha decidido aprovechar las capacidades que ofrecen ambas herramientas. Se utiliza VSCode como entorno principal para el desarrollo del firmware y se añaden algunas herramientas de terminal (CLI) que ofrece el entorno de *Arduino*.

INFORMACIÓN

El planteamiento híbrido adoptado implica que se heredan todos los comandos que se ejecutan al activar la opción de carga de firmware desde el IDE de Arduino. En este proceso se lanza una secuencia de rutinas preestablecidas y, para ello, se utiliza su propia herramienta de línea de comandos, conocida como **Arduino IDE CLI**.

De hecho, el propio IDE de Arduino expone en su consola integrada la salida estándar y de error de la herramienta (véase Ilustración 2.13 – Ejemplo terminal IDE Arduino).

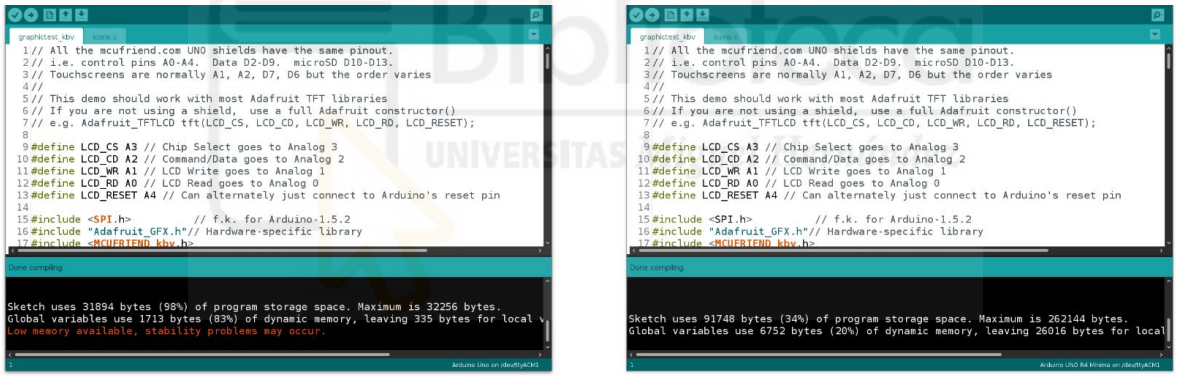


Ilustración 2.13 – Ejemplo terminal IDE Arduino

arduino/arduino-cli

Arduino command line tool

ARDUINO

Arduino CLI es una solución integral que ofrece gestores de placas/bibliotecas, generador de bocetos, detección de placas, cargador de archivos y otras herramientas necesarias para usar cualquier placa y plataforma compatible. [7]

Además de ser una herramienta independiente, Arduino CLI es el núcleo de todo el software oficial de desarrollo de Arduino (Arduino IDE, Arduino Web Editor). [7]

Un ejemplo de un comando que se ejecuta en la terminal al presionar el botón *Verificar* en el IDE de Arduino es el siguiente:

```
Verificar errores y generar el archivo binario (.hex)
>> arduino-cli compile -fqbn arduino:avr:uno name_of_file.ino
```

DESTACADO

Se utiliza el entorno de desarrollo de VSCode para generar el código y los ficheros, pero para compilar y cargar el firmware se utiliza la herramienta CLI de Arduino.

2.4. Sistemas operativos y entornos de terminal

Al disponer de un sistema operativo Windows 11 Pro, se emplea Windows PowerShell como herramienta de línea de comandos nativa del sistema. PowerShell se utiliza para interactuar con los puertos de comunicación serie (COM) del ordenador y programar directamente el MCU, véase Ilustración 2.14 – Herramienta de terminal PowerShell.

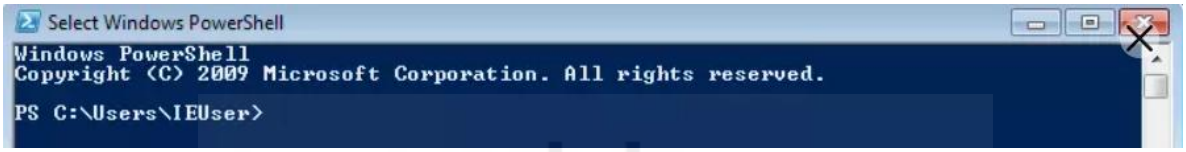


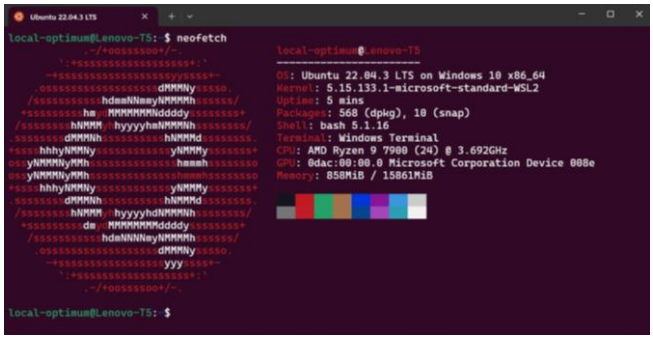
Ilustración 2.14 – Herramienta de terminal PowerShell

PowerShell ejecuta las utilidades del Arduino CLI, permitiendo la compilación, carga y monitorización del firmware directamente desde la terminal.

WSL2 (Windows Subsystem for Linux 2) se ha utilizado como entorno de terminal, integrado en Windows, permitiendo ejecutar herramientas de desarrollo y utilidades de línea de comandos que solo estarían disponibles en Linux de forma nativa.

El uso de esta herramienta se justifica debido a que los sistemas Linux, en particular Ubuntu, disponen de un sistema de gestión de paquetes robusto, lo que facilita mucho el acceso a versiones actualizadas de las herramientas de desarrollo. Por ese motivo, la ejecución del **Arduino CLI** se ha realizado mediante este subsistema Linux (WSL2).

Véase Ilustración 2.15 – Sistema operativo Linux (Ubuntu) WSL2.



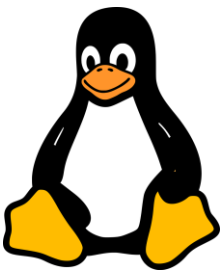


Ilustración 2.15 – Sistema operativo Linux (Ubuntu) WSL2

3. Diseño y Desarrollo del Hardware

El presente apartado tiene como objetivo desglosar y justificar las decisiones técnicas adoptadas durante el proceso de diseño electrónico, abarcando desde la concepción lógica del esquemático hasta el rutado físico del circuito impreso (PCB).

DESTACADO

La finalidad de este capítulo no es solo describir la implementación del hardware, sino aportar una perspectiva necesaria para entender la elección de cada componente en términos de funcionalidad, integridad de la señal y disponibilidad del componente.

Para ello, la exposición se estructura siguiendo la misma **arquitectura modular** que se define durante todo el proyecto. En primera instancia, se analizará el diseño a nivel esquemático, en el que la abstracción por bloques permite simplificar la complejidad de todo el sistema electrónico. Avanzando en los siguientes apartados, se detalla la implementación de estos elementos en un circuito impreso.

Véase Ilustración 3.1 – Layout del circuito impreso (PCB) en KiCad, para visualizar con mayor detalle el resultado final en la herramienta seleccionada en el apartado anterior.

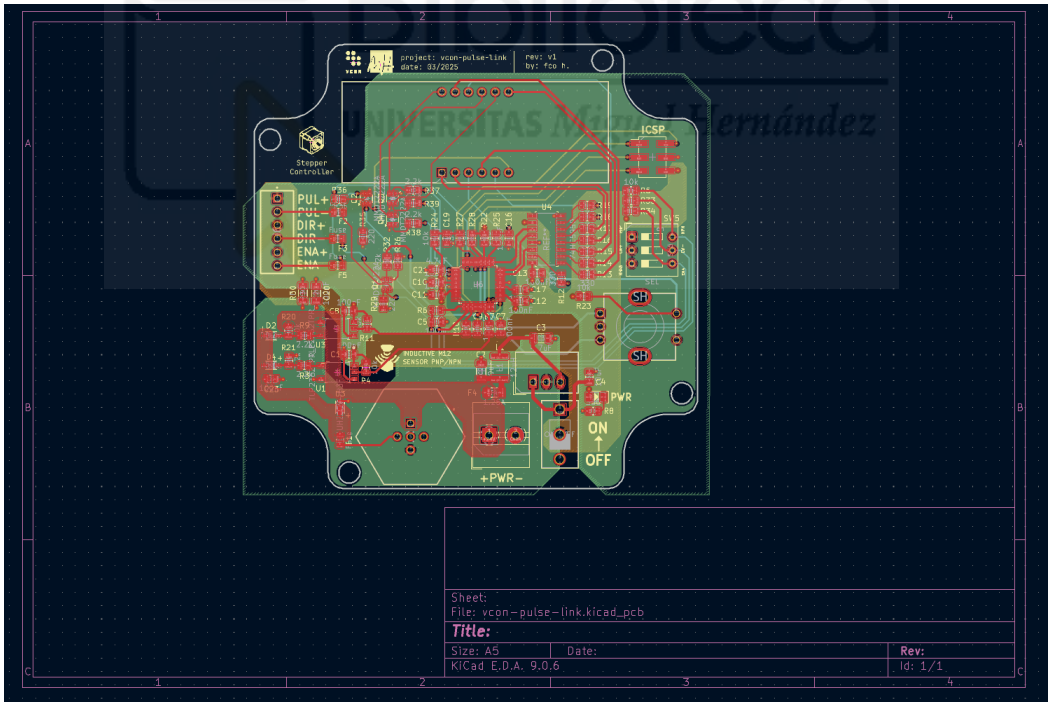


Ilustración 3.1 – Layout del circuito impreso (PCB) en KiCad

A continuación, se exponen los apartados que permiten justificar todas las decisiones tomadas durante el proyecto. Se incluye el esquemático final y los planos de la PCB en los anexos de este proyecto, véase Anexo III - Esquemático del circuito diseñado con KiCad, Anexo IV – Planos del circuito impreso (PCB) y Anexo V – Planos de acotación del circuito impreso (PCB).

3.1.1. Módulo de Alimentación (Power Supply)

El módulo de alimentación tiene como objetivo transformar la tensión de entrada +24V DC, típica en entornos industriales, a una tensión regulada +5V DC adecuada para alimentar el resto de la electrónica del sistema.

POWER SUPPLY

SELECCIÓN DEL COMPONENTE

El regulador de tensión seleccionado es **RECOM R-78E5.0-0.5**, un módulo convertidor DC-DC de bajo ruido y alta eficiencia. Véase Ilustración 3.2 – Características del regulador de tensión R-78E para visualizar sus características básicas.

Selection Guide

Part Number	Input Voltage Range [VDC]	Output Voltage [VDC]	Output Current [A]	Efficiency (1) @ min. Vin [%]	Efficiency (1) @ max. Vin [%]	max. Capacitive Load (2) [µF]
R-78E3.3-0.5	6-28	3.3	0.5	88	75	220
R-78E5.0-0.5	7-28	5	0.5	92	82	220
R-78E9.0-0.5	12-28	9	0.5	94	89	220
R-78E12-0.5	15-28	12	0.5	95	92	220
R-78E15-0.5	18-28	15	0.5	95	93	220

Ilustración 3.2 – Características del regulador de tensión R-78E

Según la información ofrecida por el fabricante, los módulos de la serie R-78E se han diseñado para ofrecer grandes ventajas: alta eficiencia, amplio rango de tensión de entrada y regulación precisa en la salida, manteniendo un bajo coste. [8]

El módulo utiliza un encapsulado SIP-3 compatible con TO-220. Las dimensiones son: 11,6 × 8,5 × 10,4 mm, lo que permite optimizar el espacio disponible en la PCB. [8]

Eficiencia del regulador de tensión

El fabricante ofrece una curva en la que se compara la eficiencia del regulador frente a la corriente de salida, véase Ilustración 3.3 – Curva de eficiencia en función de la carga del regulador R-78E5.0-0.5.

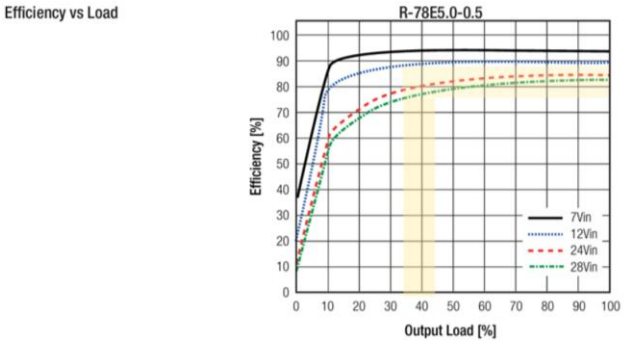


Ilustración 3.3 – Curva de eficiencia en función de la carga del regulador R-78E5.0-0.5

Este diseño electrónico consume, en condiciones normales, un 40.0 % de la corriente máxima que es capaz de suministrar el regulador de tensión. El MCU es el componente que consume la mayor parte (200 mA) frente a los 500 mA que es capaz de suministrar el regulador R-78E.

Del gráfico anterior se extrae la conclusión de que el regulador de tensión nunca tendrá una eficiencia inferior al 80%, por lo que no es necesaria la instalación de un disipador externo para reducir la temperatura. En cualquier caso, los disipadores suelen añadirse cuando la eficiencia del regulador se encuentra por debajo del 60%.

Metodología empleada para diseñar el circuito impreso

Para diseñar el circuito electrónico adyacente al regulador de tensión, se han seguido las directrices que marca la normativa **EN55032**, específica de emisiones (EMI).

La normativa define dos clases de emisión, denominadas *Class A y Class B,* en función del entorno en el que se prevé utilizar el equipo electrónico. *Class A* está orientada a entornos técnico-industriales, mientras que *Class B* está definida para entornos más domésticos y de oficinas, por lo que la *Class B* es más restrictiva que la *Class A*.

Este diseño electrónico cumple las recomendaciones correspondientes a la *Class B*, con el objetivo de minimizar al máximo las interferencias. Para ello, se debe disponer de un filtro pasivo como etapa previa a la alimentación, véase el circuito adyacente mediante la Ilustración 3.4 – Filtro EMC recomendado para el regulador R-78Exx-0.5 según EN55032.

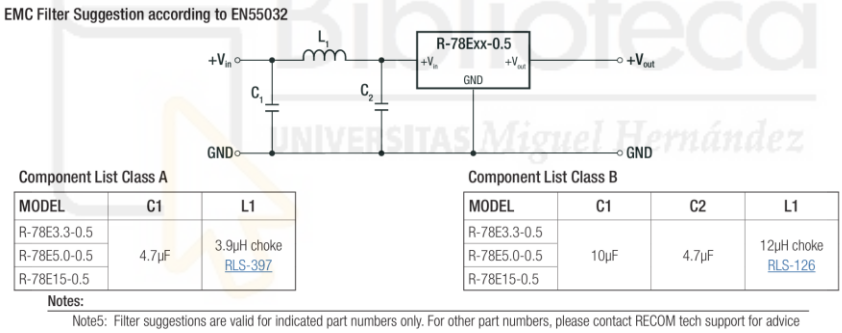


Ilustración 3.4 – Filtro EMC recomendado para el regulador R-78Exx-0.5 según EN55032

El esquemático implementado en KiCad contiene un leve error al confundir el pin de alimentación (Vin) con la salida regulada (Vout). Dicho error se produjo al importar el símbolo desde la biblioteca oficial de Digikey, que por defecto aparecía en sentido contrario al tradicional. Este error se corrigió invirtiendo físicamente la orientación del regulador durante el proceso de soldadura, véase Ilustración 3.5 – Esquema del bloque de alimentación con regulador DC-DC R-78E5.0-0.5.

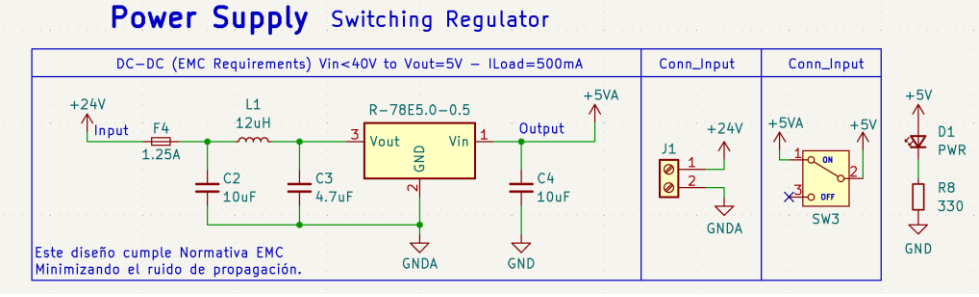


Ilustración 3.5 – Esquema del bloque de alimentación con regulador DC-DC R-78E5.0-0.5

3.1.2. Módulo de Control (Microcontroller)

El módulo de control constituye el elemento central de procesamiento del sistema electrónico diseñado. Su función principal consiste en coordinar el funcionamiento del conjunto de módulos hardware que componen la unidad.

MICROCONTROLLER

SELECCIÓN DEL COMPONENTE

Para ello, se ha seleccionado como unidad principal de procesamiento de datos el **microcontrolador ATmega**, perteneciente a la familia AVR de 8 bits. Esta elección se fundamenta en su idoneidad para llevar a cabo las aplicaciones de control.

Dentro de esta familia, se ha seleccionado el modelo ATmega328P principalmente por su superior capacidad de memoria. A diferencia de otros modelos como el 48A, 88A o 168A, el 328P dispone de 32 KB de memoria Flash y 2 KB de memoria RAM, véase una tabla comparativa mediante la Ilustración 3.6 – Comparativa de memoria entre microcontroladores ATmega48/88/168/328.

2.2 Comparison Between Processors

The ATmega48A/PA/88A/PA/168A/PA/328/P differ only in memory sizes, boot loader support, and interrupt vector sizes. Table 2-1 summarizes the different memory and interrupt vector sizes for the devices.

Table 2-1. Memory Size Summary

Device	Flash	EEPROM	RAM	Interrupt Vector Size
ATmega48A	4KBytes	256Bytes	512Bytes	1 instruction word/vector
ATmega48PA	4KBytes	256Bytes	512Bytes	1 instruction word/vector
ATmega88A	8KBytes	512Bytes	1KBytes	1 instruction word/vector
ATmega88PA	8KBytes	512Bytes	1KBytes	1 instruction word/vector
ATmega168A	16KBytes	512Bytes	1KBytes	2 instruction words/vector
ATmega168PA	16KBytes	512Bytes	1KBytes	2 instruction words/vector
ATmega328	32KBytes	1KBytes	2KBytes	2 instruction words/vector
ATmega328P	32KBytes	1KBytes	2KBytes	2 instruction words/vector

Ilustración 3.6 – Comparativa de memoria entre microcontroladores ATmega48/88/168/328

Según el datasheet oficial del fabricante, el ATmega328P está diseñado para funcionar con tensiones de alimentación en un rango que abarca desde 1.8V (4 MHz) hasta 5.5V (20 MHz), lo que permite su operación estable a 5V DC. [9]

Véase Ilustración 3.7 – Área segura de operación (SOA) del microcontrolador en función de tensión y frecuencia.

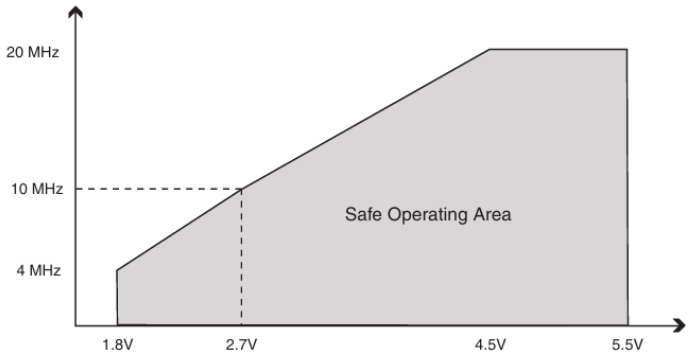


Ilustración 3.7 – Área segura de operación (SOA) del microcontrolador en función de tensión y frecuencia

Tras las justificaciones aportadas para la selección del componente ATMEGA328P-AU, se detalla el conjunto de componentes que rigen el funcionamiento del MCU.

Oscilador externo de cuarzo de 16 MHz (Clock)
Para este diseño se ha tomado la decisión de utilizar el oscilador RC interno de 8 MHz que incorpora el MCU. Por eso mismo, en el esquemático la parte del oscilador externo se visualiza con un rectángulo de color rojo, ya que no será implementada.

Filtrado mediante condensadores de desacoplo (Bypass Capacitance)
Para garantizar estabilidad en la señal de alimentación, se utilizan condensadores de desacoplo de tipo cerámicos con un valor de 100 nF. Se han dispuesto lo más cercanos a los pines de alimentación con el objetivo de filtrar ruido de alta frecuencia. Además, se añade un condensador extra que actúa como reserva de energía (4.7 µF).

Gestión del reset del microcontrolador y programación mediante ICSP
El RESET está protegido contra reinicios aleatorios mediante un filtro RC (paso bajo). Por otro lado, el puerto ICSP (J3) es vital para configurar los fusibles del MCU y cargar el binario / hexadecimal con el firmware completo en la memoria flash.

Véase el esquemático implementado en KiCad mediante la Ilustración 3.8 – Esquema del bloque de control con microcontrolador ATMEGA328P-AU.

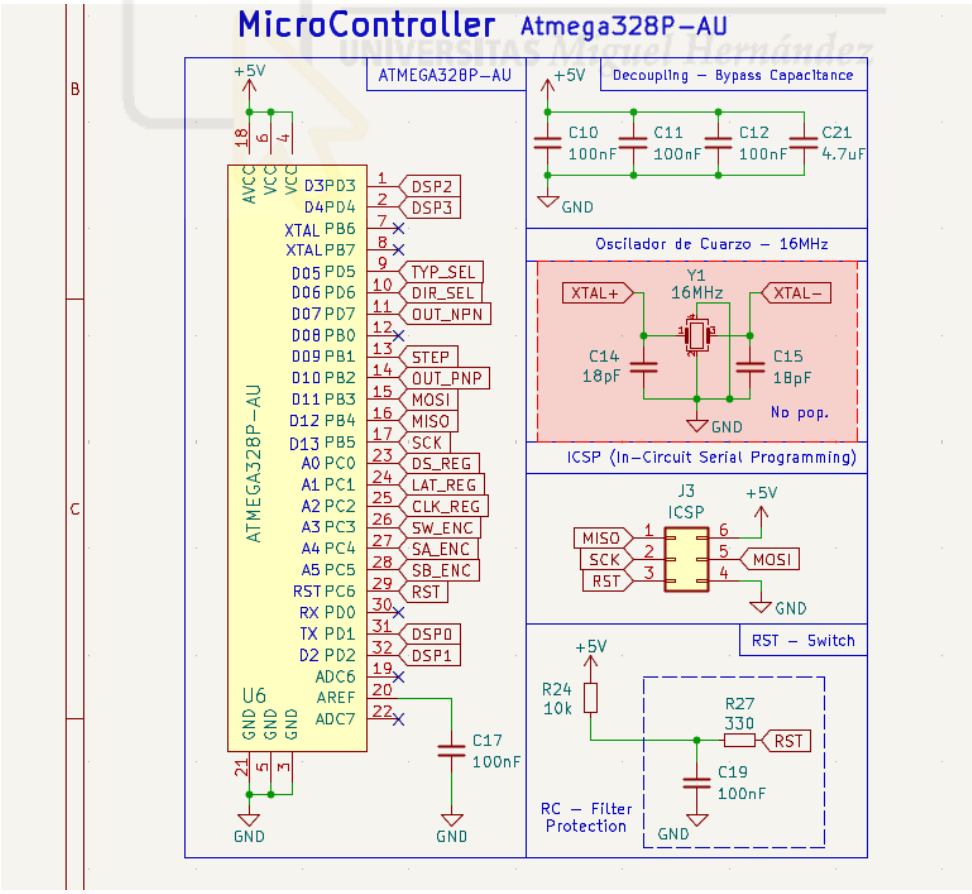


Ilustración 3.8 – Esquema del bloque de control con microcontrolador ATMEGA328P-AU

3.1.3. Módulo de Visualización (ShiftRegister Display)

El módulo de visualización tiene como objetivo proporcionar una interfaz de salida simple y directa que permita al usuario conocer en tiempo real el estado y parámetros de funcionamiento del sistema.

SHIFTREGISTER

SELECCIÓN DEL COMPONENTE

Para optimizar el uso de los pines del microcontrolador, se ha seleccionado el registro de desplazamiento 74HC595D junto con un display 5641AS. Esta combinación permite el control total de la visualización utilizando únicamente tres líneas de datos.

Justificación del uso del componente en el diseño

En caso de no utilizar el registro de desplazamiento, deberían conectarse los pines del MCU directamente al display. Esto supondría emplear un total de 12 pines, contando los necesarios para activar cada dígito (D1, D2, D3, D4) junto con los pines de cada segmento que se pretende iluminar (A, B, C, D, E, F, G, DP).

Esta configuración se simplifica mediante el registro de desplazamiento **74HC595D**, enviando datos en serie a los dígitos mediante un canal serie que utiliza 3 pines para realizar la comunicación (DS, STCP, SHCP), véase Ilustración 3.9 – Arquitectura de control del display mediante registro de desplazamiento 74HC595.

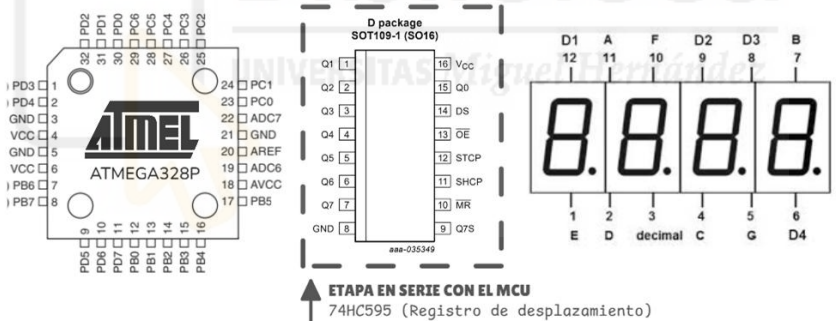


Ilustración 3.9 – Arquitectura de control del display mediante registro de desplazamiento 74HC595

El registro de desplazamiento utiliza los pines citados para convertir datos enviados en serie hacia una salida en paralelo. Los bits se registran mediante una señal de reloj (SHCP) y se transfieren a la salida mediante una señal de almacenamiento (STCP), véase Ilustración 3.10 – Descripción de pines del registro de desplazamiento 74HC595.

Table 2. Pin description

Symbol	Pin	Description
Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7	15, 1, 2, 3, 4, 5, 6, 7	parallel data output
GND	8	ground (0 V)
Q7S	9	serial data output
$\overline{MR}$	10	master reset (active LOW)
SHCP	11	shift register clock input
STCP	12	storage register clock input
$\overline{OE}$	13	output enable input (active LOW)
DS	14	serial data input
Q0	15	parallel data output 0
V_{CC}	16	supply voltage

Ilustración 3.10 – Descripción de pines del registro de desplazamiento 74HC595

Según el datasheet oficial del fabricante del display, los diodos LED tienen una caída de tensión Forward Voltage (Vf) = 1.9 V cuando circula una corriente de 10 mA. [10]

Se utilizará un valor de resistencia de 330 Ω para garantizar la integridad del sistema en cada segmento, asegurando que circula una corriente de 9.4 mA por LED. Además, esta corriente nunca superará la corriente máxima admisible en cada pin del registro de desplazamiento y tendrá un valor cercano a las condiciones óptimas del display.

Véase el cálculo, Ecuación 3.1 – Cálculo de la corriente que circula por el LED.

$$I = \frac{VCC - Vf}{R} = \frac{5\,[V] - 1.9\,[V]}{330\,[\Omega]} = 9.4\,[mA]$$

Ecuación 3.1 – Cálculo de la corriente que circula por el LED

Según el fabricante del registro de desplazamiento, los segmentos o pines del registro tienen limitaciones de corriente, nunca deberán superar los ±20-25 mA, lo que justifica el uso de resistencias para limitar la corriente. [11] Véase Ilustración 3.11 – Valores eléctricos máximos absolutos del 74HC595.

Table 4. Limiting values

In accordance with the Absolute Maximum Rating System (IEC 60134). Voltages are referenced to GND (ground = 0 V).

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	supply voltage		-0.5	+7	V
I_{IK}	input clamping current	$V_I < -0.5$ V or $V_I > V_{CC} + 0.5$ V	-	±20	mA
I_{OK}	output clamping current	$V_O < -0.5$ V or $V_O > V_{CC} + 0.5$ V	-	±20	mA
I_O	output current	$V_O = -0.5$ V to ($V_{CC} + 0.5$ V)			
		pin Q7S	-	±25	mA
		pins Qn	-	±35	mA

Ilustración 3.11 – Valores eléctricos máximos absolutos del 74HC595

Se han utilizado transistores NPN para no utilizar los pines del MCU como drenadores de corriente. Véase el esquemático en KiCad mediante la Ilustración 3.12 – Esquema del subsistema de visualización con display de 7 segmentos y 74HC595.

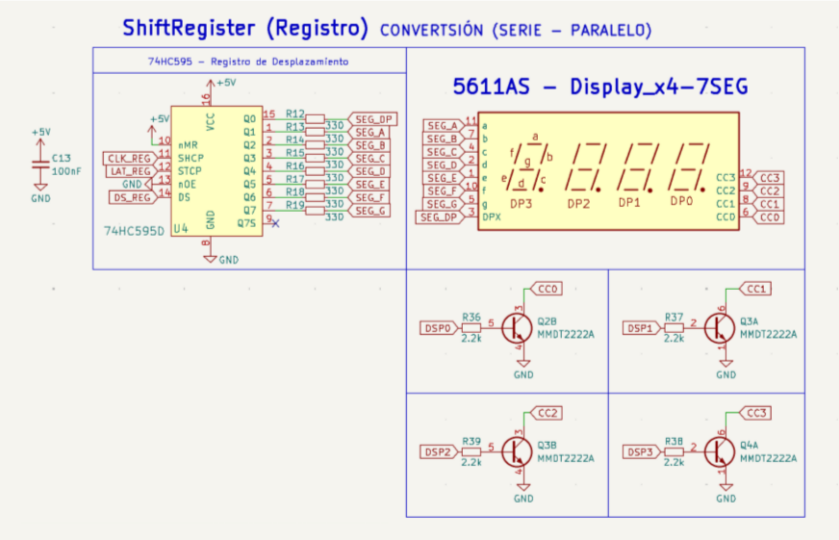


Ilustración 3.12 – Esquema del subsistema de visualización con display de 7 segmentos y 74HC595

3.1.4. Interfaz Electrónica de entrada (Inductive Sensor)

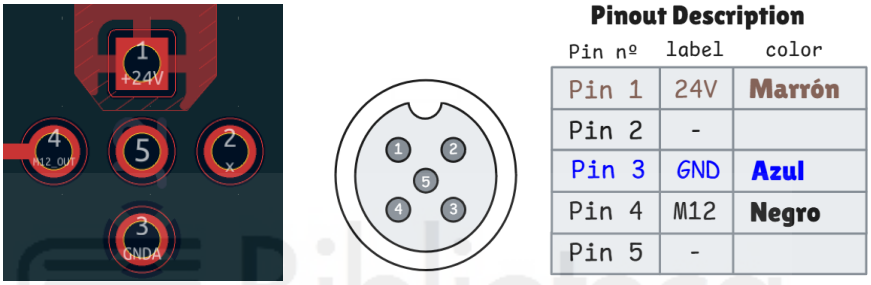
La interfaz electrónica de entrada constituye la primera etapa para introducir datos al sistema. Su función es captar la presencia de los objetos metálicos utilizando el sensor inductivo conectado a la interfaz.

INDUCTIVE SENSOR

SELECCIÓN DEL COMPONENTE

Con el fin de asegurar una conexión correcta y estandarizada del sensor inductivo, el diseño incorpora un único conector hembra M12 con codificación A, exclusivo para este tipo de sensores, garantizando compatibilidad y evitando errores de conexión.

El pinout sigue la asignación de colores normalizada para conectores M12 de 4 pines (véase Ilustración 3.13 – Asignación de pines del conector).



Pinout Description

Pin nº	label	color
Pin 1	24V	Marrón
Pin 2	-	
Pin 3	GND	Azul
Pin 4	M12	Negro
Pin 5	-	

Ilustración 3.13 – Asignación de pines del conector

Funcionamiento y lógica del sensor inductivo de métrica 12

Este apartado se centra en la implementación del sensor PNP, aunque el esquemático en realidad esté diseñado para conectar sensores de tipo NPN. Tal y como se detalla en la documentación técnica del sensor inductivo (BES M12ZI-PSC40B-S04G), su lógica de funcionamiento (PNP) facilita que cuando se aproxima un objeto metálico, la entrada del sensor se conecte directamente con la señal de datos (véase Ilustración 3.14 – Lógica PNP y asignación de pines del conector M12).

Ilustración 3.14 – Lógica PNP y asignación de pines del conector M12

Dentro de la problemática que tiene esta implementación, se destaca:

- Al conectar la salida (Pin 4) con la entrada (Pin 1), se recibe una señal de +24V DC en el pin de datos, por lo que no se puede conectar a un MCU.
- Si no se detecta un objeto, la salida del sensor inductivo quedará en un estado de alta impedancia, lo que puede causar lecturas erróneas en el MCU.

Resolver el conflicto del estado de alta impedancia (Z)

Para solventar el problema del estado de alta impedancia, se colocaron resistencias Pull-Up y Pull-Down de acuerdo con el comportamiento esperado para cada sensor.

Adaptar el nivel de tensión de la señal de datos

En el siguiente apartado se muestra cómo se utilizaron optoacopladores con el fin de reducir el nivel de tensión a un valor compatible con las lecturas del MCU.

Las medidas del conector son relevantes para este diseño, ya que este elemento es el más sobresaliente del diseño electrónico y puede ser un factor clave en espacios muy reducidos o en algunos entornos específicos (maquinaria con poco espacio). Véanse las medidas del conector mediante la Ilustración 3.15 – Dimensiones mecánicas y vista 3D del conector M12.

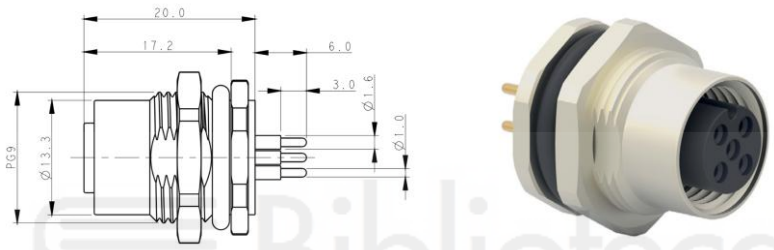


Ilustración 3.15 – Dimensiones mecánicas y vista 3D del conector M12

También se introduce por seguridad en todas las entradas externas del diseño un **fusible**, con el objetivo de limitar la corriente eléctrica en caso de un defecto. Se utiliza un PolyFuse (TVS) con el objetivo de absorber picos de tensión transitorios.

Véase el esquemático implementado en KiCad mediante la Ilustración 3.16 – Interfaz Electrónica de entrada (Sensor inductivo).

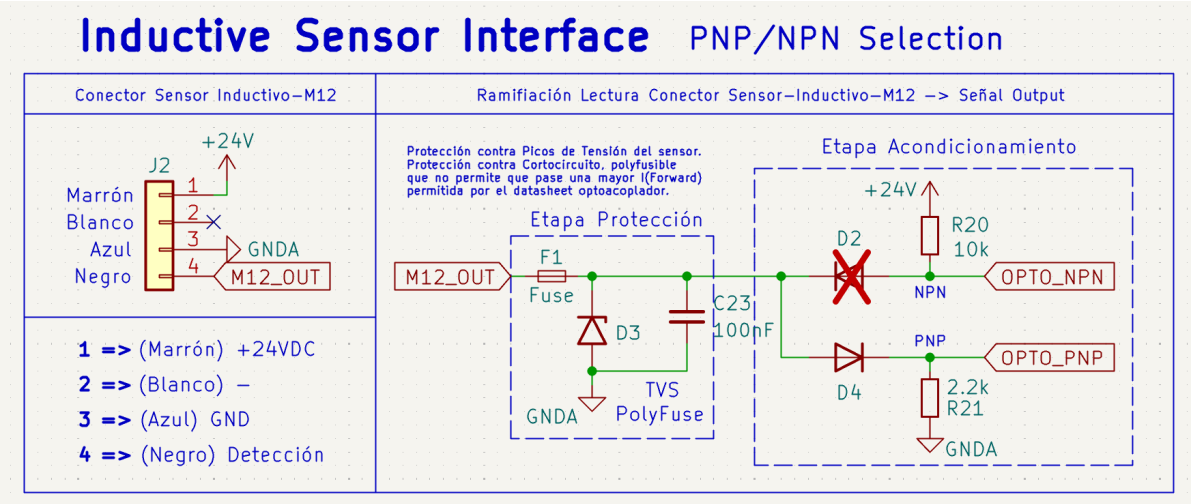


Ilustración 3.16 – Interfaz Electrónica de entrada (Sensor inductivo)

3.1.5. Módulo de Aislamiento (Optocoupler Conversor)

Este módulo tiene como objetivo realizar la conversión de niveles de tensión desde 24V a 5V (TTL). Simultáneamente se garantiza un aislamiento de tipo galvánico entre la etapa de potencia y la de control.

OPTOACOPLADOR

SELECCIÓN DEL COMPONENTE

Para esta función crítica se ha descartado el uso de optoacopladores genéricos de baja velocidad (como la familia PC817). Se ha seleccionado específicamente el **Toshiba TLP2362** al ser el componente que introduce el menor retardo posible.

La justificación principal de esta elección reside en la velocidad de conmutación de los transistores internos del componente. La electrónica digital establece que, a menor tiempo de conmutación, mayor será la velocidad con la que se envían los datos.

Según la hoja del fabricante, el diodo emisor interno del optoacoplador requiere de una corriente nominal de 10 mA para garantizar una conmutación robusta. Si se considera una tensión de entrada equivalente a 24 V y la caída de tensión típica del diodo interno Vf = 1.55 V, se calcula el valor de la resistencia necesaria para limitar la corriente. Véase Ecuación 3.2 – Cálculo de la corriente que circula por el diodo emisor interno.

$$R_{in} = \frac{V_{in} - V_F}{I_F} = \frac{24\ V - 1.55\ V}{0.010\ A} = 2245\Omega$$

Ecuación 3.2 – Cálculo de la corriente que circula por el diodo emisor interno

Véase el esquemático implementado en KiCad mediante la Ilustración 3.17 – Módulo de aislamiento (optoacoplador).

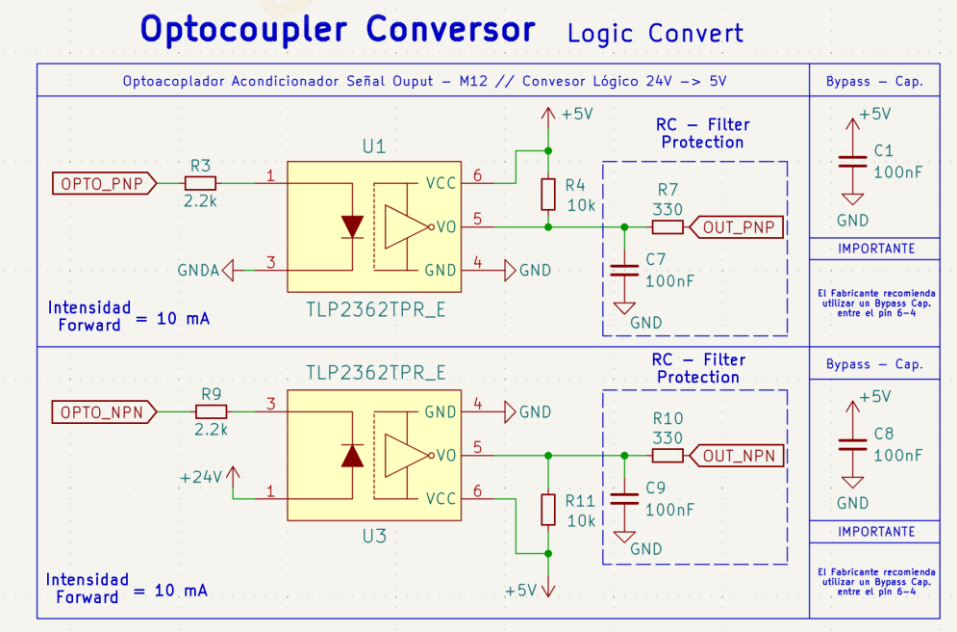


Ilustración 3.17 – Módulo de aislamiento (optoacoplador)

3.2. Características físicas y tecnológicas de la PCB.

Respecto al diseño físico y la geometría del circuito impreso, debido a los requisitos del proyecto, se ha definido un contorno ***edge-cuts*** a partir de la proyección ortogonal de la parte trasera del motor NEMA 34.

El objetivo de este contorno es aprovechar al máximo el espacio disponible dentro de la envolvente del motor sin exceder sus límites físicos, evitando de esta manera que se produzcan interferencias mecánicas con las vibraciones del motor.

Plano de acotación del contorno

El plano de acotación del contorno pretende establecer todas las medidas del equipo electrónico, véase Ilustración 3.18 - Cotas del contorno del circuito impreso. El diseño parte de una forma rectangular a la que se han aplicado recortes en las cuatro esquinas de forma simétrica.

Para este plano será suficiente con acotar un cuarto de la pieza, el resto de las cotas se consideran simétricas respecto a los planos de simetría, trazados en línea de tipo discontinua. Véase el plano completo en los anexos de este proyecto, Anexo V - Planos de acotación del circuito impreso (PCB).

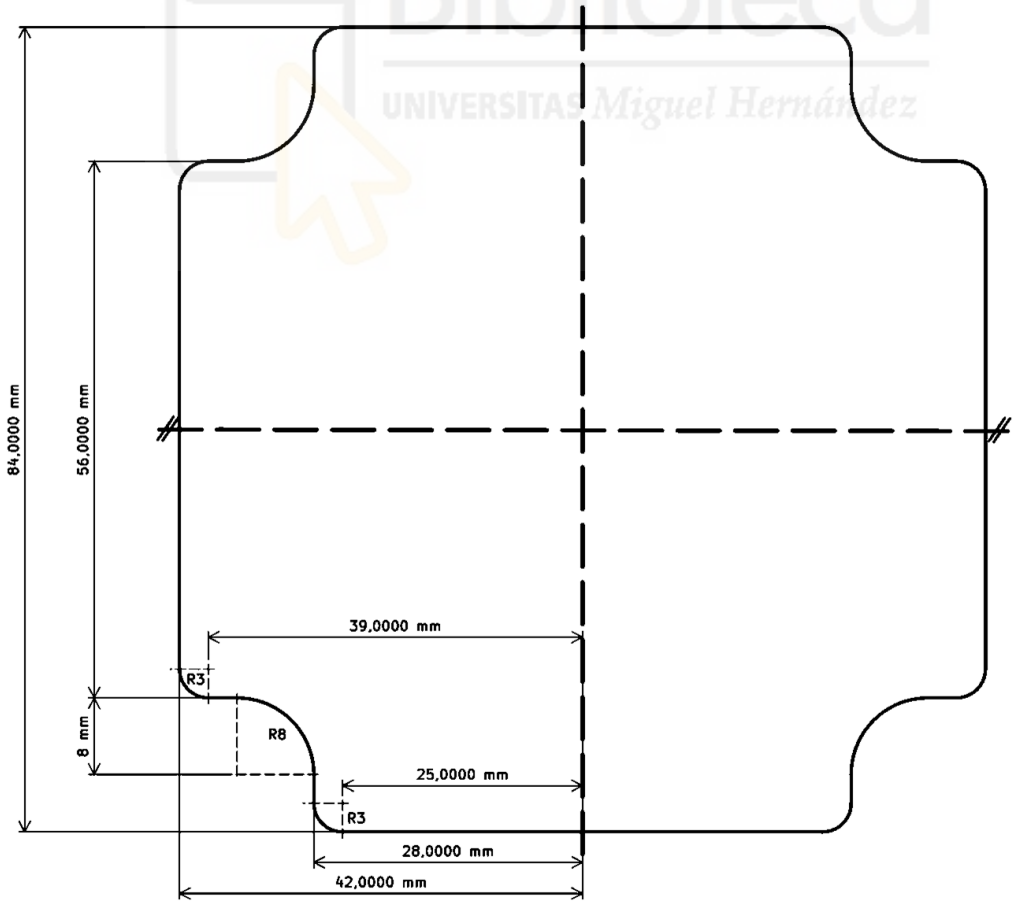


Ilustración 3.18 - Cotas del contorno del circuito impreso

Con el objetivo de reducir el número de cotas se ha desarrollado otro plano específico para los agujeros pasantes del diseño electrónico. A este plano se le ha denominado plano de acotación de los taladros, y se encuentra disponible en los anexos del proyecto. Véase Anexo V – Planos de acotación del circuito impreso (PCB).

Plano de acotación de los taladros

Para garantizar la fijación mecánica y estructural del circuito impreso, se colocarán los taladros en los extremos del diseño. La distribución de estos orificios pasantes sigue una estricta simetría respecto al eje diagonal del circuito impreso. Sin embargo, para este plano de acotación no se han trazado ejes de simetría para reducir el número de cotas, ya que no conviene simplificar en este caso la información.

Véase la Ilustración 3.19 - Cotas de los orificios pasantes del circuito impreso para obtener una mayor información sobre las cotas de los orificios pasantes.

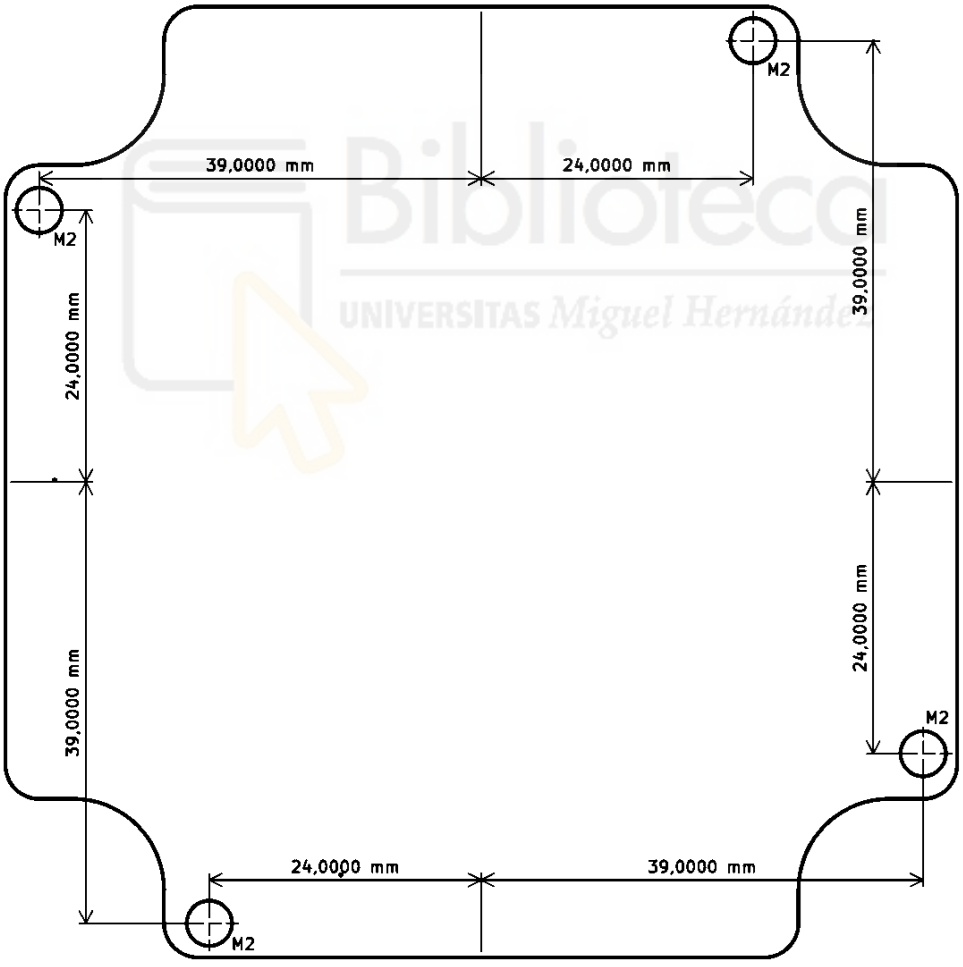


Ilustración 3.19 - Cotas de los orificios pasantes del circuito impreso

Esta disposición dota al sistema electrónico de un anclaje firme sobre el estator del motor y le permite tener un alineamiento preciso respecto a la carcasa.

Características tecnológicas y estrategias de apilado (Stackup)

Definida la geometría del diseño electrónico, se especifica la composición interna del circuito impreso. Para este diseño se ha seleccionado como material base **FR-4**, que es un tejido de fibra de vidrio con resina epoxi. En el software de KiCad, se han definido previamente los parámetros de fabricación, véase Ilustración 3.20 - Definición del apilado de capas (stackup) y materiales en KiCad.

En cuanto a la placa, se puede observar que se mantiene un grosor estándar de 1.6 mm.

Ilustración 3.20 - Definición del apilado de capas (stackup) y materiales en KiCad

Respecto a la arquitectura y al conjunto de capas de la PCB, debido a la complejidad de realizar un rutado completo y con el objetivo de minimizar las interferencias, se ha optado por una configuración multicapa de 4 capas. A diferencia de circuitos impresos de dos capas, esta arquitectura permite utilizar capas intermedias para el reparto de energía, desacoplando señales digitales internas. La distribución se divide en:

- **Capas externas (Top y Bottom Layer) –** Se han destinado exclusivamente al montaje de componentes y rutado de pistas digitales principales.
- **Capas internas (In1, In2) –** Se utilizan como planos de cobre, la capa interna superior (Inner Layer 1) actúa como plano de referencia GND y la capa inferior (Inner Layer 2) se utiliza como plano de alimentación VCC.

Véase Ilustración 3.21 - Esquema estructural de una PCB multicapa de 4 capas para obtener una representación gráfica de la estructura multicapa.

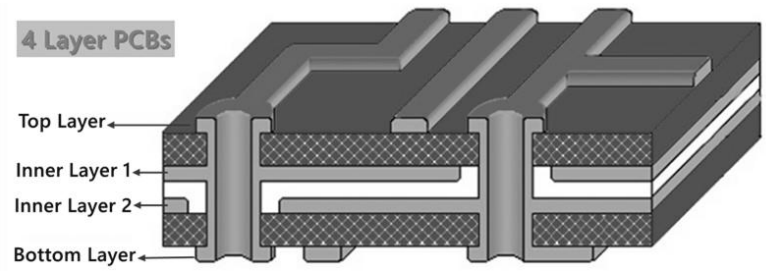


Ilustración 3.21 - Esquema estructural de una PCB multicapa de 4 capas

3.3. Consideraciones acerca del emplazado y rutado de la PCB

El proceso del diseño y rutado del circuito impreso no se ha limitado a la conexión de los nodos mediante la herramienta KiCad, sino que se ha seguido una serie de técnicas y metodologías profesionales con el objetivo de mejorar la integridad de la señal.

A continuación, se detallan algunas de las metodologías clave empleadas:

Estrategia de emplazamiento a partir de la distribución modular del esquemático
Para garantizar el rutado y trazar las pistas lo más cortas posibles se ha trasladado toda la arquitectura modular definida en el esquemático directamente a la distribución física de la PCB. Evitando que el diseño de las pistas capte ruido del entorno.

Esto implica que toda la abstracción realizada en el apartado anterior, realizada para simplificar el diseño y dotar al esquemático de una mayor limpieza visual, se mantiene también en esta fase del emplazamiento físico de la PCB. Véase la Ilustración 3.22 - Distribución modular de los componentes sobre el circuito impreso en la que los componentes respetan sus agrupaciones de origen.

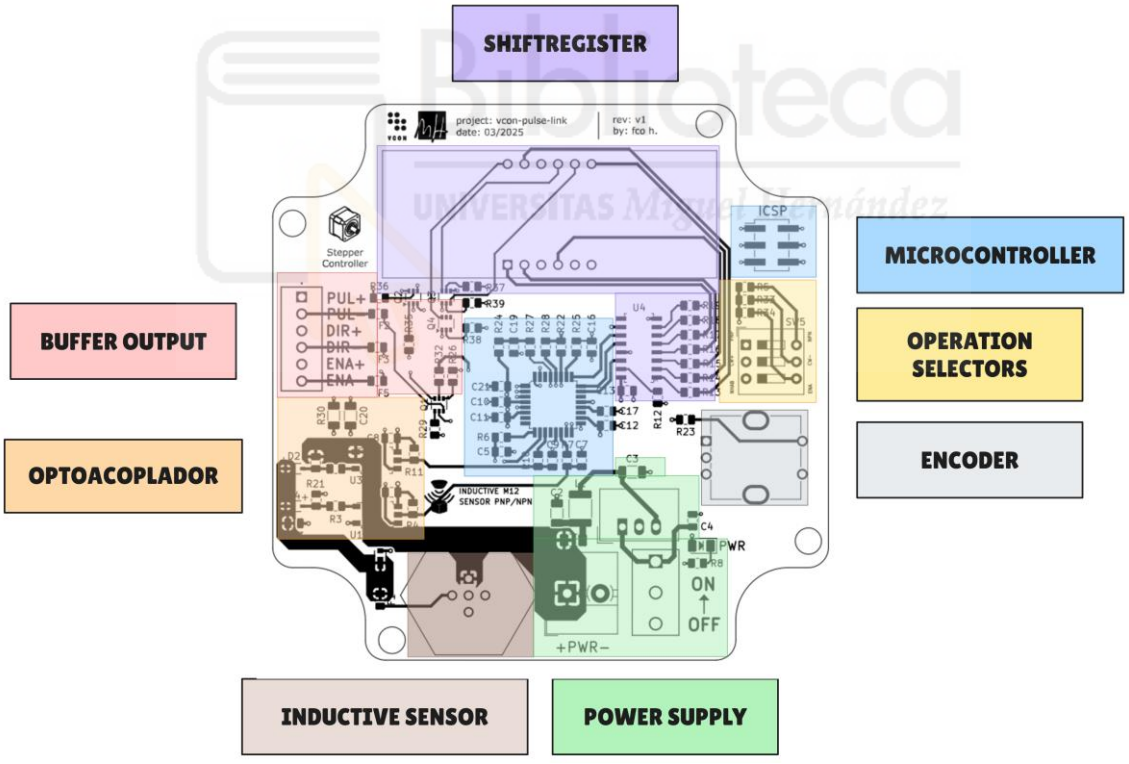


Ilustración 3.22 - Distribución modular de los componentes sobre el circuito impreso

Esta estrategia de agrupación por proximidad permite que los componentes pasivos (bobinas y condensadores) queden situados próximos a los pines de los (IC) circuitos integrados a los que están conectados, reduciendo el efecto de las **inductancias de tipo parásitas** y mejorando la estabilidad de la señal de cada módulo individualmente.

Gestión de los planos de masa y corrientes de retorno causadas por defectos

Uno de los principales problemas que han surgido durante el desarrollo de la PCB, es la coexistencia de una etapa de potencia ruidosa debido al sensor inductivo (+24 V DC) junto con una etapa de lógica digital sensible (+5 V DC).

Al compartir la tierra, la corriente que activa al sensor inductivo puede tener picos de corriente al conmutar. Estos defectos fluyen por el mismo plano de tierra que el MCU si no se controla de forma adecuada.

Para evitar precisamente ese fenómeno, se ha implementado una estrategia de **planos de masa divididos** conectados mediante un único punto. Como se aprecia en la Ilustración 3.23 - Segmentación de planos de masa y punto único de conexión, el plano de referencia no es continuo para toda la placa, sino que se ha segmentado en dos regiones físicas bien diferenciadas.

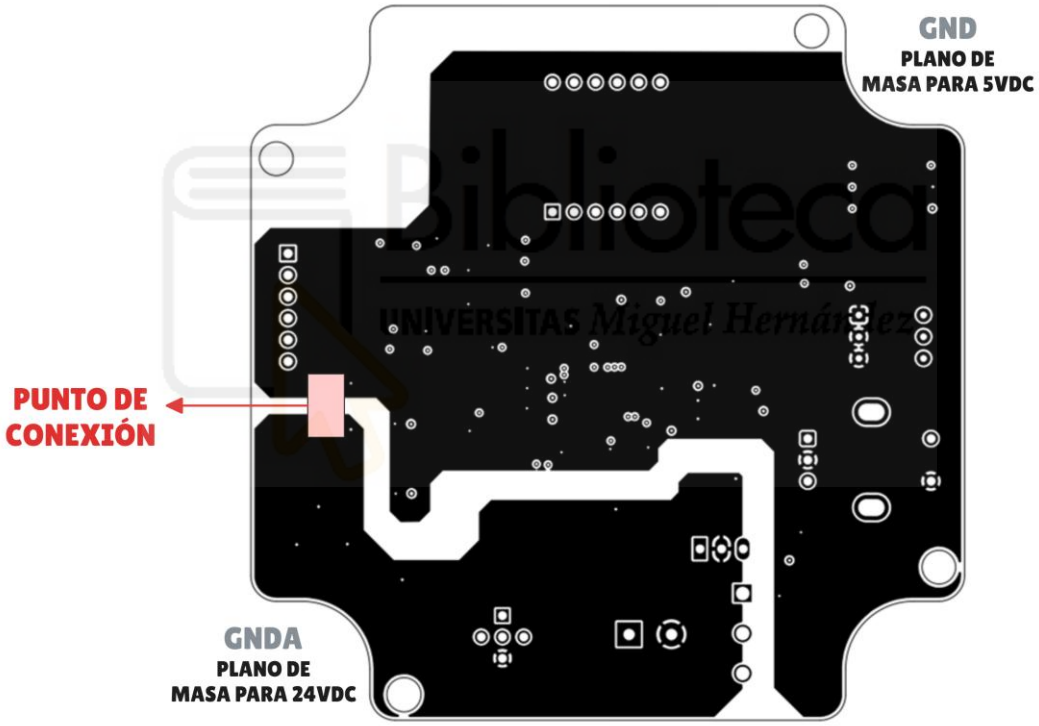


Ilustración 3.23 - Segmentación de planos de masa y punto único de conexión

Se visualizan dos regiones bien diferenciadas en el esquema anterior:

- **PLANO MASA +24V DC –** Dedicado exclusivamente al sensor inductivo y al regulador de tensión lineal que se alimenta también a esa tensión.
- **PLANO MASA +5V –** Dedicado exclusivamente a circuitos digitales del diseño.

Para mantener una referencia de potencial común, ambas zonas se unen en un único punto mediante una resistencia de valor 1 Ω en paralelo con un condensador, ambos ubicados de forma estratégica en la capa superior del diseño. (Layer In 1)

Gestión de los planos de masa y corrientes de retorno causadas por defectos

Una vez definida la referencia de masa, el siguiente punto crítico es la distribución de la alimentación +5 V DC de la salida del regulador de tensión, encargada de alimentar toda la lógica de control digital adyacente.

Para realizar esta implementación, se ha descartado el uso de pistas de señal estándar y en su lugar, se ha optado por el trazado de polígonos de cobre que permitan realizar una distribución uniforme de la alimentación. (Layer In 2)

Como se muestra en la Ilustración 3.24 - Plano interno de alimentación +5 V en Inner Layer 2, el plano de alimentación se ha implementado mediante un polígono de cobre dedicado que garantiza una menor impedancia y una mejor estabilidad.

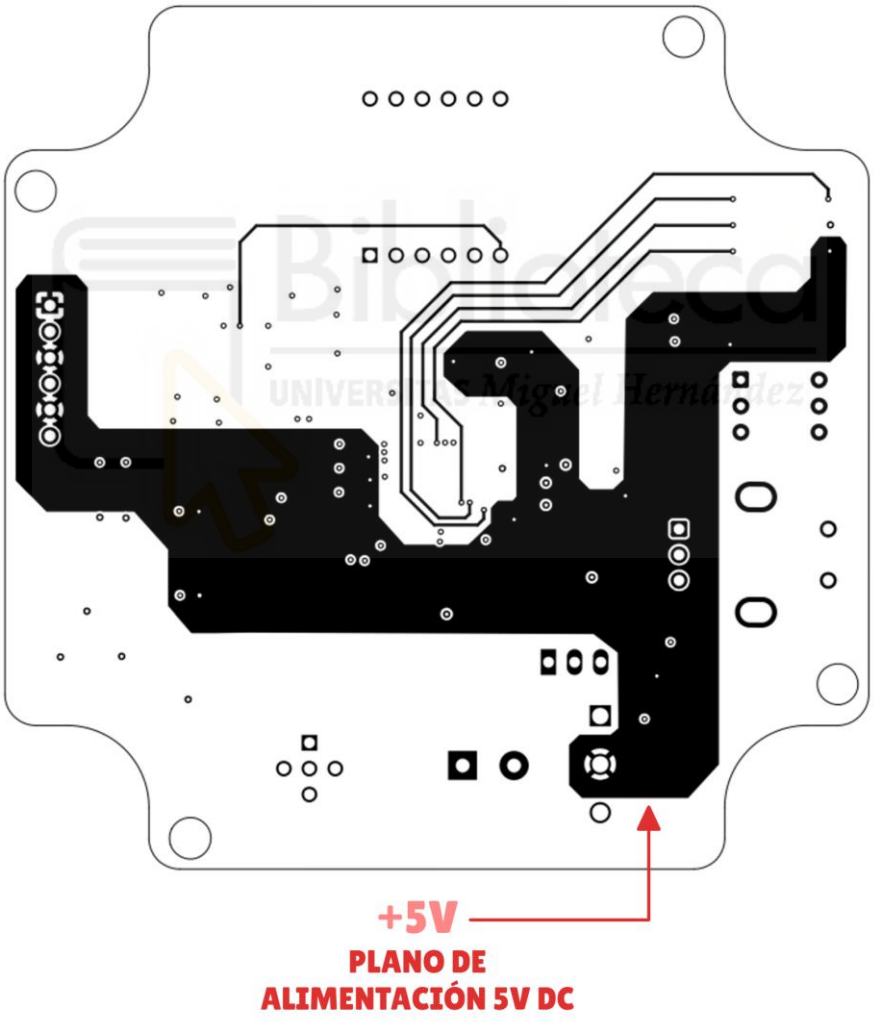


Ilustración 3.24 - Plano interno de alimentación +5 V en Inner Layer 2

Mediante esta técnica de distribución de la alimentación, se reduce la impedancia resultante por lo que la caída de tensión es menor y se garantiza la estabilidad térmica al aumentar la superficie de cobre.

3.4. Desglose de los costes de fabricación de una PCB.

La fabricación del producto electrónico se ha llevado a cabo a través de un fabricante de circuitos impresos, mientras que el proceso de soldadura y montaje se ha realizado en el laboratorio de la universidad, sin recurrir a un método de fabricación industrial y/o automatizado, como por ejemplo podrían ser las máquinas de pick & place.

Para aportar transparencia al presupuesto, los costes se han estructurado siguiendo una jerarquía de categorías y conceptos, tal y como se detalla a continuación:

- **Categoría 1 –** Costes de los materiales directos (Hardware)
  - Concepto 1.1.– Circuito impreso desnudo (PCB)
  - Concepto 1.2.– Componentes electrónicos (Bill Of Materials)

- **Categoría 2 –** Costes del ensamblaje y puesta en marcha del diseño
  - Concepto 2.1.– Mano de obra cualificada.
  - Concepto 2.2.– Consumibles de taller.

Descripción de las categorías que componen el presupuesto

Los costes de los materiales directos son todos los elementos que constituyen el diseño físico del circuito impreso, en esta categoría se encuentran conceptos como el coste de los componentes electrónicos o el coste de un circuito impreso.

Los costes de ensamblaje del prototipo engloban todo el proceso de fabricación en el que se incluye la mano de obra empleada y los consumibles de taller utilizados.

Se observa en la Ilustración 3.25 - Resultado final del producto PCB en comparación con Arduino UNO, que el diseño permite implementar una solución técnica específica mediante un proceso de prototipado, con resultados equiparables a los de un diseño electrónico profesional, como podría ser el Arduino UNO R3.

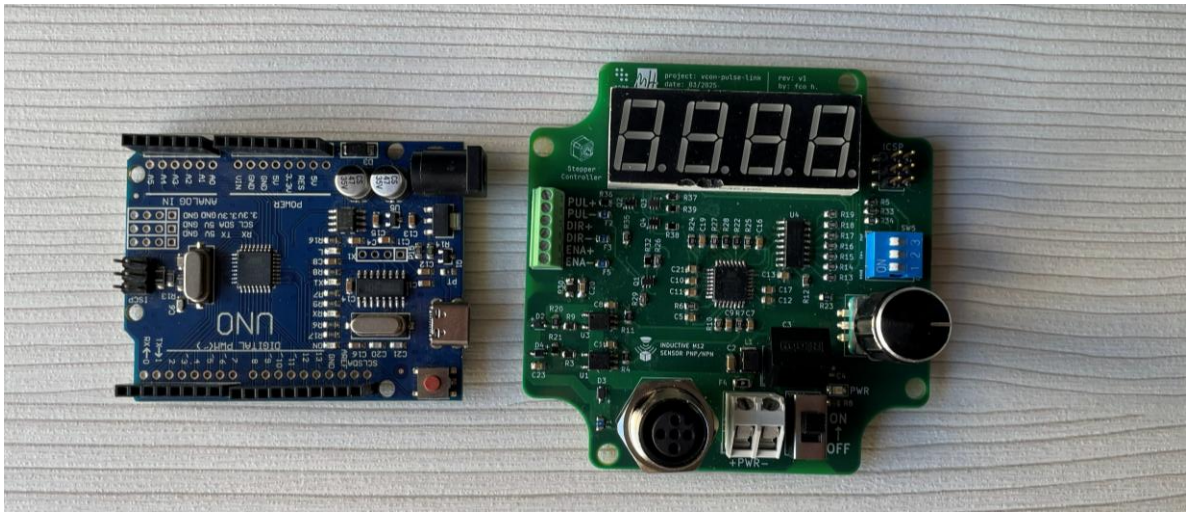


Ilustración 3.25 - Resultado final del producto PCB en comparación con Arduino UNO

Categoría 1 – Costes de los materiales directos (Hardware)

En este apartado se cuantifica el coste de los elementos físicos que componen el HW, adquiridos para la construcción del prototipo, diferenciando entre conceptos:

Categoría 1. Concepto 1.– Circuito impreso desnudo (PCB)

La fabricación de la placa se ha encargado al proveedor JLCPCB. A continuación, se detalla el coste del pedido, que incluye como mínimo un lote de 5 unidades y los gastos de transporte necesarios, dado que su fabricación se produce en el extranjero, véase Tabla 3.1 - Desglose de costes del pedido de PCB multicapa (Concepto 1.1).

Tabla 3.1 - Desglose de costes del pedido de PCB multicapa (Concepto 1.1)

Elemento (Descripción)	Qt	Ud	€	€/PCB
Fabricación del circuito impreso multicapa	5	Uds.	24,15 €	4,83 €
Gastos de envío del pedido desde el fabricante	1	Serv.	18,45 €	18,45 €
Stencil para soldadura SMD (no utilizado)	1	Serv.	- €	- €
Total, Concepto (1.1.- Circuito impreso desnudo)				**4,83 € ***

* Los pedidos de PCB se realizan habitualmente de forma conjunta con otros proyectos del departamento, lo que evita asumir gastos de envío individuales. Por ello, no se han incluido en el coste unitario calculado en el concepto anterior.

En cuanto al pedido, se han recibido 5 unidades del diseño electrónico tal y como se muestra en la Ilustración 3.26 - Lote recibido de PCB multicapa sin componentes. Se visualiza un circuito impreso desnudo sin ningún componente todavía soldado.

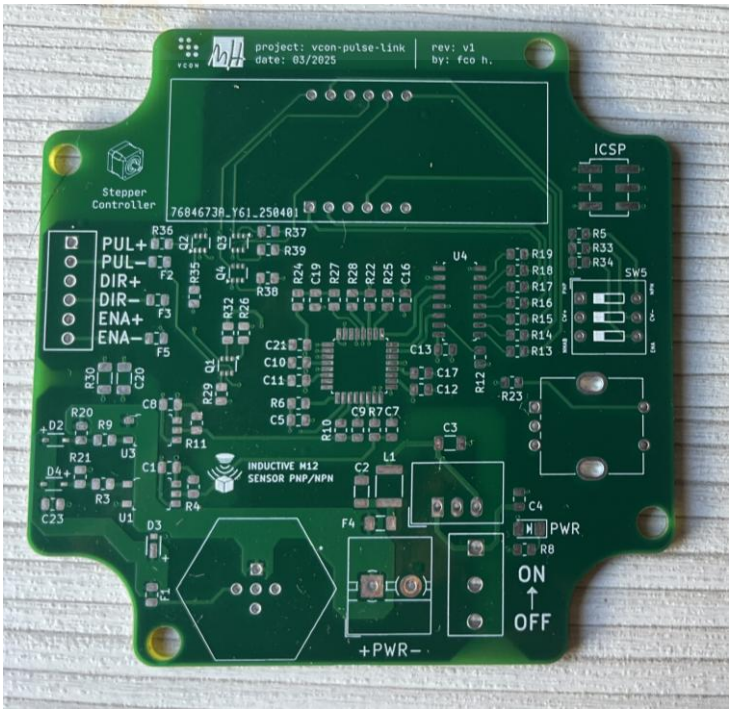


Ilustración 3.26 - Lote recibido de PCB multicapa sin componentes

Categoría 1. Concepto 2.– Componentes electrónicos (PCB)

Todos los componentes han sido adquiridos a través del distribuidor oficial Digikey. La siguiente Tabla 3.2 - Desglose de costes de los componentes electrónicos (Concepto 1.2) resume el coste total de los elementos pasivos, activos y los conectores que han sido empleados para el montaje del prototipo electrónico.

Para mejorar la trazabilidad de los componentes empleados, se ha añadido la columna de referencias que está vinculada directamente con el esquemático de KiCad.

Tabla 3.2 - Desglose de costes de los componentes electrónicos (Concepto 1.2)

REF (Esquemático)	Digikey-PN (Proveedor)	Qt	Ud	€/ud	€/PCB
R3, R9, R21, R32–R39	RMCF0805FT2K20CT-ND	10	ud	0,05 €	0,50 €
R4, R5, R11, R20, R22–R24, R28	RMCF0805FT10K0CT-ND	8	ud	0,05 €	0,40 €
R6–R8, R10, R12–R19, R25, R27	RMCF0805FT330RCT-ND	14	ud	0,05 €	0,70 €
R26, R29, R35	RMCF0805FT220RCT-ND	3	ud	0,05 €	0,15 €
R30	RMCF1206FT0R00CT-ND	1	ud	0,06 €	0,06 €
C1, C5, C7–C13, C16, C17, C19, C20, C23	311-1142-1-ND	14	ud	0,06 €	0,84 €
C2, C4	1276-2872-1-ND	2	ud	0,32 €	0,64 €
C3, C21	1276-3179-1-ND	2	ud	0,28 €	0,56 €
C14, C15	399-1909-1-ND	2	ud	0,07 €	0,14 €
Y1	SER3621TR-ND	1	ud	0,45 €	0,45 €
D1	160-1169-1-ND	1	ud	0,16 €	0,16 €
D3	264-CUHZ30VH3FCT-ND	1	ud	0,37 €	0,37 €
D4	RB751V40T1GOSCT-ND	2	ud	0,11 €	0,22 €
F1–F5	296-3527-5-ND	5	ud	0,20 €	1,00 €
L1	945-RLS-126-RTR-ND	1	ud	0,92 €	0,92 €
Q1–Q4	MMDT4401-FDITR-ND	4	ud	0,35 €	1,40 €
U1, U3	TLP2362(TPRECT-ND)	2	ud	0,91 €	1,82 €
U2	945-1648-5-ND	1	ud	3,04 €	3,04 €
U4	1727-2821-2-ND	1	ud	0,28 €	0,28 €
U5	COM-11405-ND	1	ud	1,48 €	1,48 €
U6	ATMEGA328P-AU-ND	1	ud	2,45 €	2,45 €
J1	732-691201720002S-ND	1	ud	1,25 €	1,25 €
J2	A124957-ND	1	ud	6,67 €	6,67 €
J3	612-TSM-103-01-SM-DV-ND	1	ud	1,23 €	1,23 €
J6	277-1277-ND	1	ud	3,32 €	3,32 €
SW3	2223-SLW-1276864-4A-D-ND	1	ud	0,77 €	0,77 €
SW4	PEC11R-4020F-N0024-ND	1	ud	1,74 €	1,74 €
SW5	2223-DS01C-254-S-03BE-ND	1	ud	0,53 €	0,53 €
Total, Concepto (2.2.-Componentes electrónicos)					**33,09 €**

Categoría 2 – Costes del ensamblaje y puesta en marcha del diseño

En este apartado se cuantifica el valor del proceso de construcción manual, sumando el tiempo de trabajo técnico dedicado y los materiales auxiliares empleados.

Categoría 2. Concepto 1.– Mano de obra cualificada

Esta categoría comprende las horas de trabajo técnico necesarias para la fabricación y puesta en marcha del prototipo. Para su estimación económica, se ha considerado la contratación de un estudiante de ingeniería en prácticas, con un coste de 7€/hora.

Tabla 3.3 - Coste de mano de obra cualificada (Concepto 2.1)

Elemento (Descripción)	Qt	Ud	€/ud	€/PCB
Soldadura de componentes	2	Horas	7,00 €	14,00 €
Montaje y revisión del prototipo	1	Horas	7,00 €	7,00 €
Verificación funcional del ensamblaje	1	Horas	7,00 €	7,00 €
Desarrollo y programación del firmware	8	Horas	7,00 €	56,00 €
Total, Concepto (2.1.- Mano de obra cualificada)				**84,00 €**

Categoría 2. Concepto 1.– Consumibles de taller

Incluye el material necesario para realizar las soldaduras y se aplican únicamente los productos consumibles, por lo que se tiene en cuenta que el material ya dispone de un equipamiento básico necesario. Esto implica que gastos como el cautín no computan.

Tabla 3.4 - Coste de consumibles de taller (Concepto 2.2)

Elemento (Descripción)	Qt	Ud	€/ud	€/PCB
Estaño para soldadura	1	ud	2,00 €	2,00 €
Alcohol isopropílico	1	ud	1,50 €	1,50 €
Flux en cinta	1	ud	1,00 €	1,00 €
Total, Concepto (2.2.- Materiales auxiliares de montaje)				**4,50 €**

Costes totales para la fabricación de un prototipo electrónico

Como síntesis, se agregan los costes parciales obtenidos en categorías anteriores para determinar el coste final de fabricación de un prototipo funcional. En la siguiente tabla se adjunta el desglose final, sumando la inversión del hardware (Categoría 1) y la valoración de todo el proceso del ensamblaje (Categoría 2).

Tabla 3.5 - Resumen de costes totales del prototipo electrónico

Elemento (Descripción)	€/PCB
Categoría 1. Concepto 1.1.- Circuito impreso desnudo (PCB)	4,83 €
Categoría 1. Concepto 1.1.- Componentes electrónicos (Bill Of Materials)	33,09 €
Categoría 2. Concepto 2.1.- Mano de obra cualificada	84,00 €
Categoría 2. Concepto 2.2.- Materiales auxiliares de montaje	4,50 €
Total, precio por producto electrónico	**126,42 €**

4. Diseño y desarrollo del Firmware

En este apartado se explica el funcionamiento del firmware desarrollado, desglosando cómo se estructuran sus distintos módulos, qué responsabilidades asumen cada uno de ellos y cómo se comunican para dar lugar al comportamiento deseado del sistema.

DESTACADO

El objetivo es conocer en detalle cómo opera el microcontrolador (MCU en adelante) con el hardware explicado en la sección anterior (3. Diseño y Desarrollo del Hardware).

4.1. Introducción y objetivos del firmware desarrollado

En función de los objetivos marcados por la empresa colaboradora del proyecto, se ha desarrollado un firmware acorde a la lógica de funcionamiento esperada.

Nota importante sobre compatibilidad del firmware

Este firmware está específicamente diseñado para microcontroladores ATMEGA de la familia ATmega328P. Su uso en otros modelos de la plataforma AVR podría requerir adaptaciones, especialmente en ciertos registros internos. Por tanto, no se garantiza su funcionamiento en MCUs diferentes sin modificaciones previas más profundas. Todo el código fuente desarrollado en este proyecto se encuentra en la sección: Anexo VI – Código fuente del firmware.

Función lógica principal del firmware

el sistema embebido implementado pretende ser el núcleo lógico del sistema, cuyo principal enfoque u objetivo por el que ha sido diseñado es: controlar la generación de pulsos, mediante la señal STEP que va hacia el driver conectado en la interfaz de salida.

Estas son las principales acciones que desarrolla el dispositivo:

- Interpretar el giro y pulsación del encoder para ajustar un parámetro interno, denominado delta, que determina la frecuencia objetivo de la señal STEP.
- Supervisar el sensor inductivo para habilitar o bloquear la generación de pulsos mediante una máquina de estados que distingue entre { SLEEP, WAIT y RUN }.
- Mostrar en tiempo real el modo de funcionamiento y el valor delta a través de un display de 7 segmentos multiplexado con un registro de desplazamiento.
- Comprobar si el usuario ha accionado el selector para guardar en la memoria EEPROM (no volátil) el modo de funcionamiento y el valor delta de arranque.

Por lo que en este apartado se expone desde un alto nivel las estrategias que se han seguido para llevar a cabo todas estas premisas impuestas previamente.

4.2. Arquitectura general del firmware desarrollado

Como buena práctica, se ha optado por realizar un diseño del software modular, que divide el sistema en bloques funcionales (clases) independientes. Esta organización permite algo muy importante en producción: mantenimiento del código y ejecución estable en tiempo real, dos de las condiciones más importantes en este proyecto.

DESTACADO

Esta arquitectura permite ahorrar tiempo cuando se depura; realizar las correcciones sobre los módulos que fallan resulta más simple y aporta una mayor trazabilidad que un código completamente monolítico, véase Ilustración 4.1- Arquitectura modular del firmware desarrollado para visualizar los módulos que componen el software.

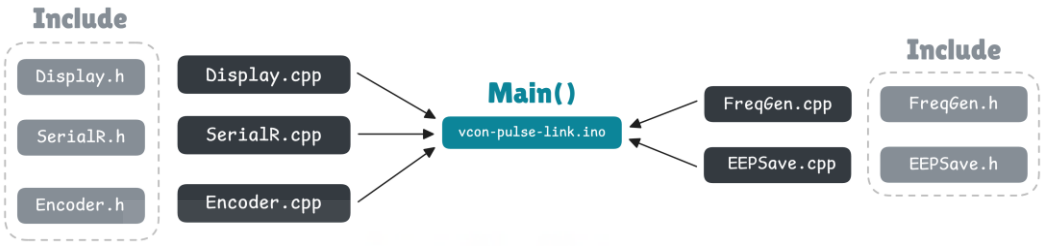


Ilustración 4.1- Arquitectura modular del firmware desarrollado

El código diseñado proporciona un nivel de escalabilidad elevado. Cada clase definida en el fichero .h actúa como una plantilla reutilizable que permite instanciar varios objetos físicos idénticos. Como ejemplo tendríamos la clase: `Encoder.h/Encoder.cpp` que permite gestionar múltiples encoders físicos sin necesidad de reescribir código ni modificar la lógica interna del sistema. Esta capacidad resulta esencial para ampliar funcionalidades/adaptar el diseño a nuevas necesidades de forma rápida y eficiente, véase Ilustración 4.2 - Instanciación de múltiples objetos a partir de la clase Encoder.

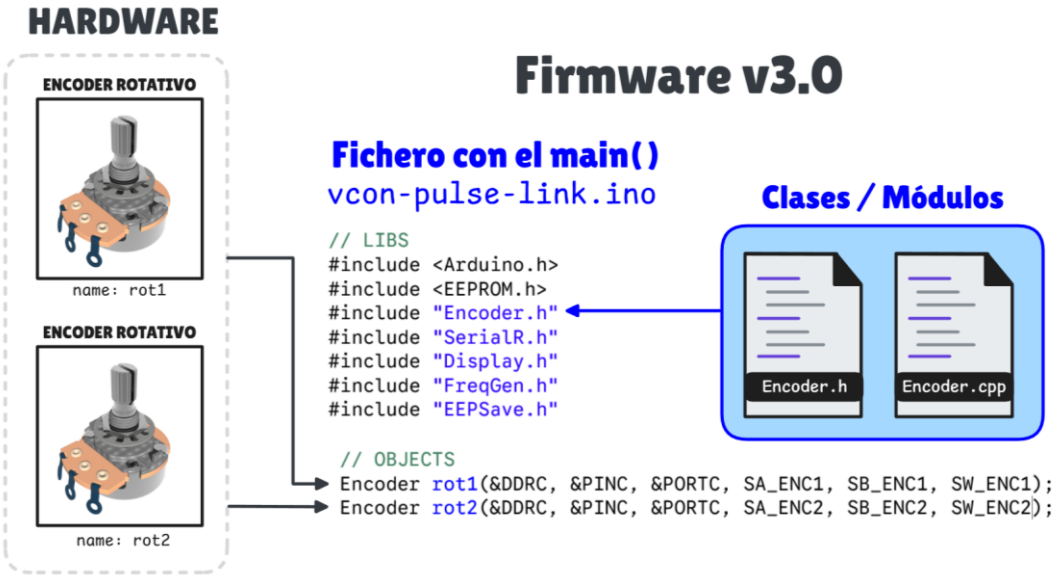


Ilustración 4.2 - Instanciación de múltiples objetos a partir de la clase Encoder

En un hipotético caso en el cual los requisitos del proyecto obliguen a utilizar un segundo encoder rotativo, al haber empleado programación orientada a objetos físicos, será suficiente con instanciar un nuevo objeto `Encoder rot2(...)`, asignando los pines de conexión tal y como se muestra en la ilustración anterior.

4.3. Acerca del lenguaje de programación del firmware

Dentro de los lenguajes de programación para sistemas embebidos, existen muchas tecnologías que permiten obtener una solución idéntica. En este apartado se justifica el uso del firmware implementado para desarrollar el dispositivo electrónico.

¿Qué ejecuta realmente un microcontrolador?

En primer lugar, es necesario comprender que un microcontrolador no ejecuta ningún lenguaje de programación de alto nivel como, por ejemplo: C++, Python o Arduino.

Es cierto que esos lenguajes son los que se emplean para programar el MCU, pero lo que realmente ejecuta la CPU interna es código máquina, es decir, una secuencia de instrucciones binarias almacenadas en su memoria flash.

DESTACADO

Las instrucciones binarias son específicas de la arquitectura del microcontrolador y son interpretadas directamente por su CPU. Ante esto, surge la necesidad de tener un compilador (diferente para cada modelo/serie) que permita generar un archivo binario, independientemente del lenguaje o las herramientas de desarrollo, véase Ilustración 4.3 - Flujo de trabajo de compilación en arquitectura STM32 (ARM Cortex-M)

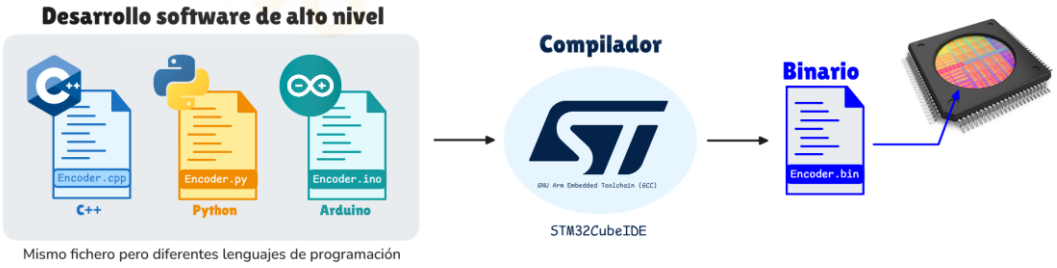


Ilustración 4.3 - Flujo de trabajo de compilación en arquitectura STM32 (ARM Cortex-M)

¿Qué es entonces un lenguaje de programación para desarrollo firmware?

Un lenguaje de programación es una herramienta que permite al desarrollador escribir el comportamiento deseado del sistema de forma comprensible para el ser humano. En sistemas embebidos, los lenguajes más utilizados son C/C++, ya que permiten un control preciso del hardware y se traducen eficazmente a código máquina.

El código fuente generado por estos lenguajes sigue un proceso de transformación que permite obtener un fichero binario que será alojado en la memoria del MCU.

Cadena de transformación del código fuente

El siguiente esquema pretende aclarar la cadena de transformación del código fuente. Véase Ilustración 4.4 - Cadena de transformación del código fuente para visualizar este ciclo de transformación del código fuente hasta la memoria FLASH del MCU.

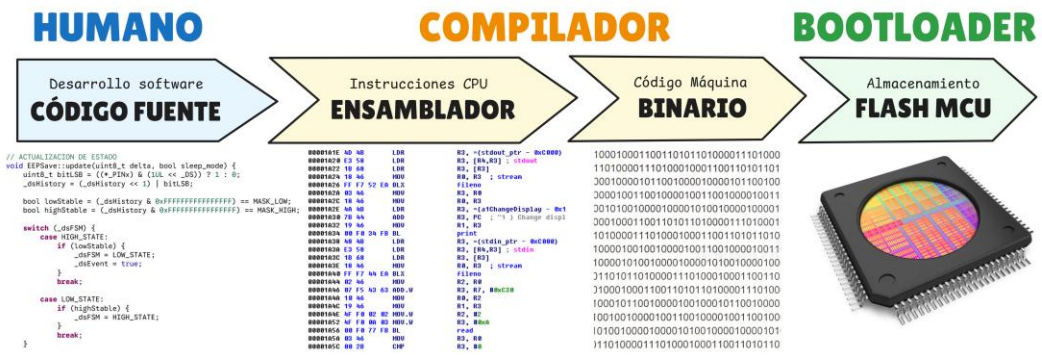


Ilustración 4.4 - Cadena de transformación del código fuente

El ensamblador es un lenguaje de programación de bajo nivel que contiene todas las instrucciones de la CPU. Aunque el programador no suele utilizarlo, este paso siempre existe, incluso con entornos de desarrollo modernos como el IDE Arduino.

Programación en BareMetal (Utilizada en el proyecto)

BareMetal es un método para desarrollar firmware que no utiliza capas de abstracción intermedias. En su lugar, se manipulan los registros de memoria directamente, por lo que este enfoque permite al programador acceder directamente a la memoria que desea configurar/controlar para afectar al comportamiento del sistema.

DESTACADO

BareMetal se utiliza junto con el lenguaje de programación C y las instrucciones que se generan son muy similares a desarrollar código ensamblador. Este método para la programación de microcontroladores es de bajo nivel en comparación con otros.

BareMetal aporta velocidad a costa de complejidad; por ejemplo, configurar un pin como entrada en un MCU es mucho más rápido con BareMetal, pero a su vez también es más complejo, al tener que acceder directamente a los registros de memoria del pin en concreto que se desea activar como entrada de datos al microcontrolador.

Información relevante acerca de BareMetal

Este proyecto utiliza BareMetal como técnica de programación, junto con otras librerías de alto nivel que se detallarán más adelante en este mismo capítulo. Los requisitos de velocidad del proyecto obligan a emplear ambas tecnologías.

A continuación, se expone una descripción de cada fichero que compone el código, con el objetivo de familiarizar al lector con las diferentes clases del sistema.

vcon-pulse-link.ino

Fichero principal del firmware: actúa como orquestador del código; inicializa todo el hardware, ejecuta el ciclo principal del programa, procesa todos los eventos, actualiza las máquinas de estado internas, sincroniza la memoria EEPROM y refresca el display.

Encoder.h / Encoder.cpp

El módulo gestiona la lectura del encoder rotativo incremental con pulsador integrado. Implementa funciones para la detección de giro (incremento/decremento) y para la detección de pulsación del botón mediante una FSM. Proporciona eventos al módulo principal para la modificación de parámetros y establecer modos de funcionamiento.

EEPSave.h / EEPSave.cpp

El módulo EEPSave gestiona el almacenamiento y recuperación de parámetros críticos del sistema en memoria no volátil. Implementa la lectura y escritura controlada de valores como el porcentaje de generación y el modo de reposo, evitando escrituras innecesarias para prolongar la vida útil de la EEPROM.

Display.h / Display.cpp

El Display se encarga de la gestión de un display 7 segmentos multiplexado de cuatro dígitos. Implementa la representación del modo de funcionamiento del motor y del valor porcentual de generación de frecuencia, incluyendo también una función que gestiona el refresco periódico de los dígitos para garantizar una visualización estable.

FreqGen.h / FreqGen.cpp

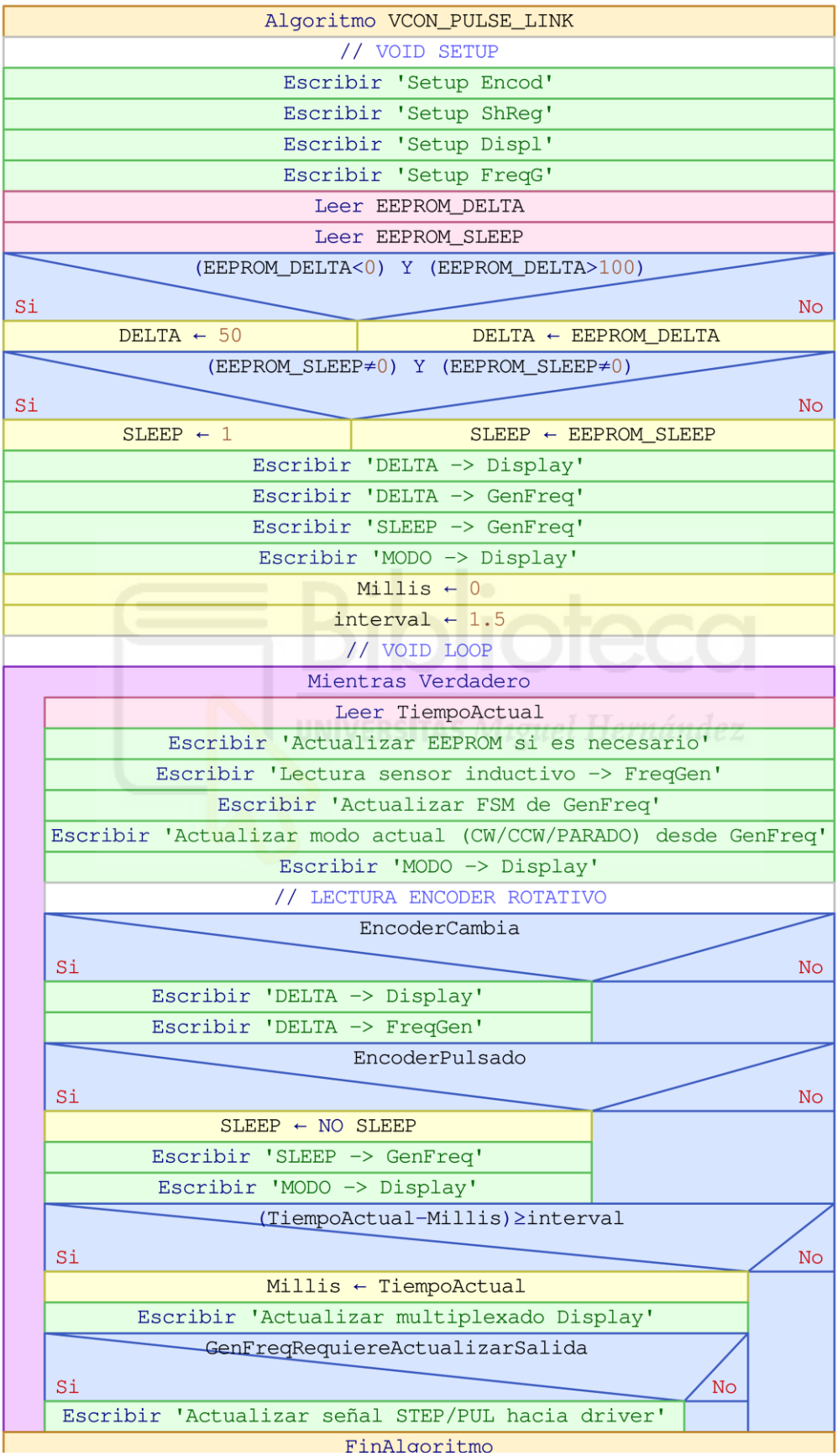
Este módulo implementa la generación de la señal de reloj (STEP/PUL) para el control del driver del motor. Incluye una máquina de estados interna que gestiona el modo de funcionamiento (giro horario, antihorario, detección de objeto, modo SLEEP); además, a través de esta FSM se gestiona el porcentaje de frecuencia generado.

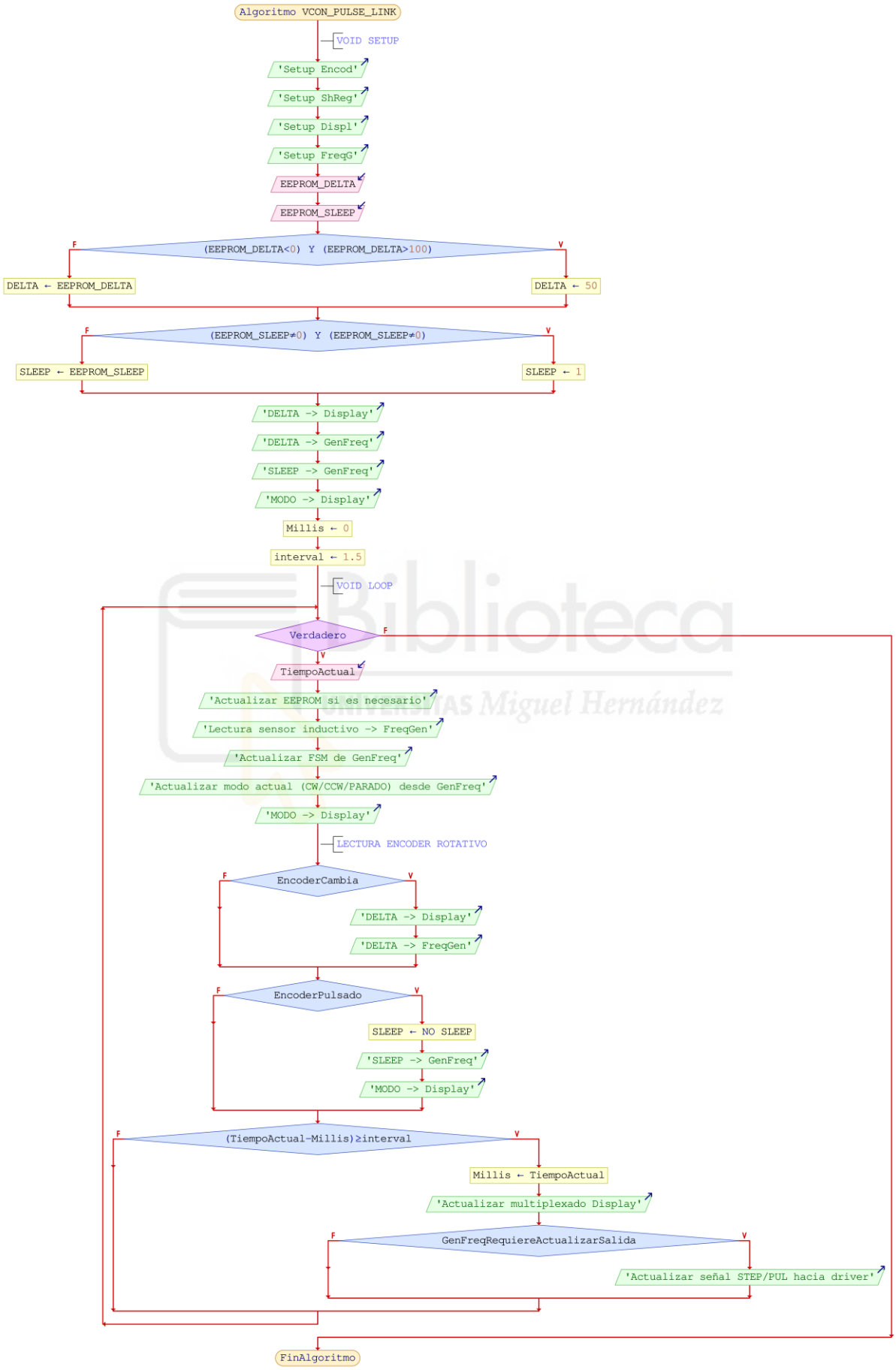
SerialR.h / SerialR.cpp

Este módulo gestiona el registro de desplazamiento encargado de la expansión del display (salidas digitales). Implementa la serialización de datos y el control de los pines necesarios para actualizar el estado visual del display de forma eficiente. Además, este módulo se utilizará como clase base para el módulo Display, que es el encargado de gestionar todos los eventos relacionados con la visualización.

A continuación, se muestra el diagrama de flujo principal del código desarrollado.

4.4. Diagrama de flujo del firmware desarrollado




Algoritmo VCON_PULSE_LINK
VOID SETUP
'Setup Encod'
'Setup ShReg'
'Setup Displ'
'Setup FreqG'
EEPROM_DELTA
EEPROM_SLEEP
(EEPROM_DELTA<0) Y (EEPROM_DELTA>100)
F
V
DELTA ← EEPROM_DELTA
DELTA ← 50
(EEPROM_SLEEP≠0) Y (EEPROM_SLEEP≠0)
SLEEP ← EEPROM_SLEEP
SLEEP ← 1
'DELTA -> Display'
'DELTA -> GenFreq'
'SLEEP -> GenFreq'
'MODO -> Display'
Millis ← 0
interval ← 1.5
VOID LOOP
Verdadero
TiempoActual
'Actualizar EEPROM si es necesario'
'Lectura sensor inductivo -> FreqGen'
'Actualizar FSM de GenFreq'
'Actualizar modo actual (CW/CCW/PARADO) desde GenFreq'
'MODO -> Display'
LECTURA ENCODER ROTATIVO
EncoderCambia
'DELTA -> Display'
'DELTA -> FreqGen'
EncoderPulsado
SLEEP ← NO SLEEP
'SLEEP -> GenFreq'
'MODO -> Display'
(TiempoActual-Millis)≥interval
Millis ← TiempoActual
'Actualizar multiplexado Display'
GenFreqRequiereActualizarSalida
'Actualizar señal STEP/PUL hacia driver'
FinAlgoritmo

5. Lógica de control y modos de operación.

El funcionamiento del dispositivo vcon-pulse-link lo gestiona el código diseñado; sin embargo, el usuario es capaz de interactuar directamente con este firmware mediante consignas, haciendo uso del encoder y de los selectores. Esto permite modificar las señales de control que se generan hacia el driver industrial.

Para entender mejor cómo el usuario puede actuar sobre el dispositivo, es necesario identificar la ubicación física de los elementos de control de la PCB, véase Ilustración 5.1 - Componentes del sistema electrónico vcon-pulse-link para ubicarlos.

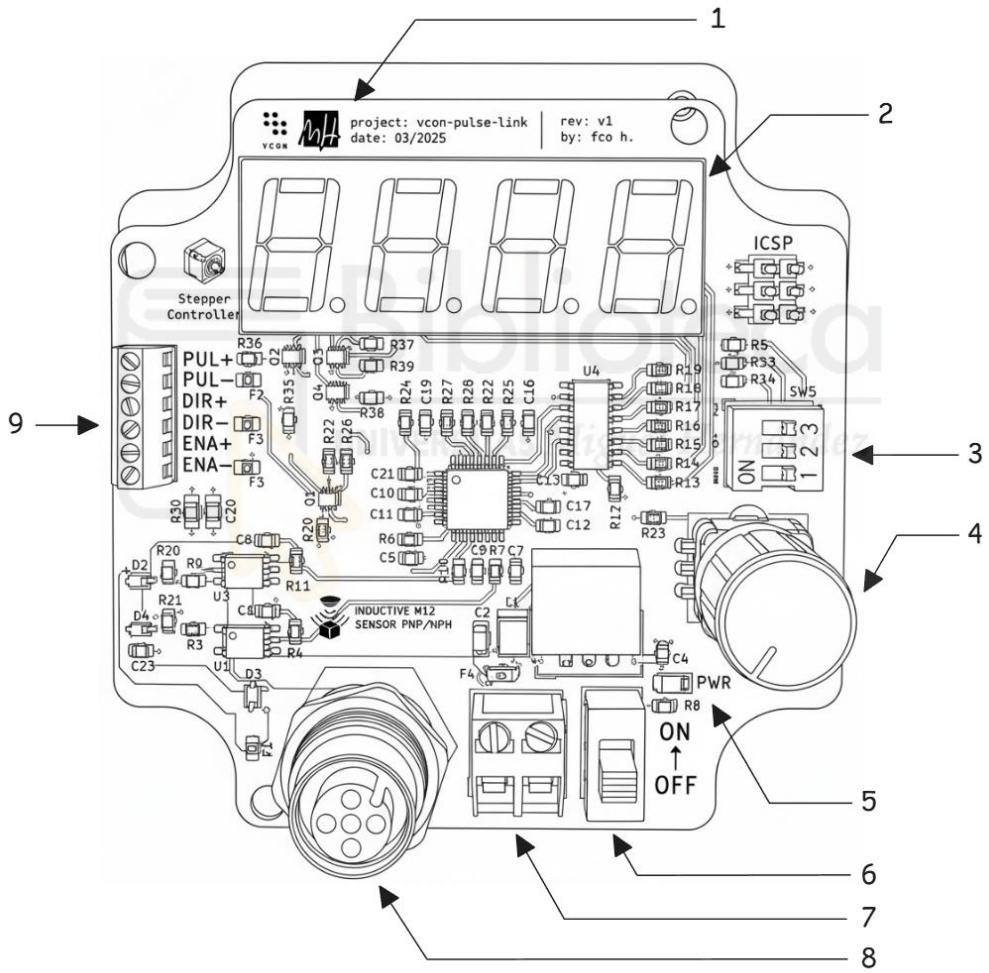


Ilustración 5.1 - Componentes del sistema electrónico vcon-pulse-link

El dispositivo vcon-pulse-link dispone de una serie de elementos electrónicos y físicos que permiten al usuario accionar y supervisar el funcionamiento tanto del driver como del motor al que se conecta. La numeración de los componentes se puede ver en la siguiente página, en la que se define cada uno de ellos.

Numeración de los componentes físicos y electrónicos de la PCB

1. Serigrafía con el nombre del producto, versión y autoría.
2. Módulo de visualización del display 7 segmentos de cuatro dígitos.
3. Selectores para ajustar modos de configuración del dispositivo.
4. Encoder rotativo con pulsador mecánico integrado.
5. Indicador de funcionamiento tipo LED (Power ON).
6. Interruptor de encendido general del dispositivo.
7. Conector de alimentación industrial (+24 V).
8. Conector para sensor inductivo (Codificación A).
9. Interfaz de comunicación con el driver industrial.

El firmware desarrollado incorpora una máquina de estados interna que determina el comportamiento del motor y la información mostrada al usuario. El sistema utiliza el primer dígito del display 7 segmentos para indicar el estado actual de la FSM, mediante una codificación visual específica que se ha establecido. Véase Ilustración 5.2 - Representación de estados del sistema en el display de 7 segmentos.

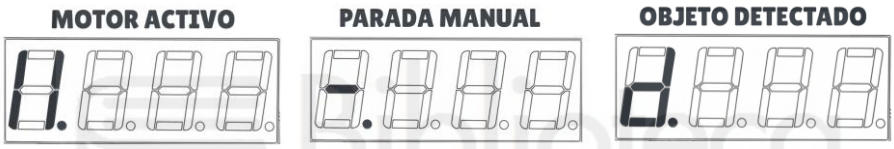


Ilustración 5.2 - Representación de estados del sistema en el display de 7 segmentos

Modo 1 – Motor activo. En el firmware encontrará este modo conocido como RUN, por lo que el sistema genera pulsos hacia el driver y el display debe visualizar el porcentaje de generación de frecuencia (Velocidad del motor).

Modo 2 – Parada Manual. En el firmware encontrará este modo conocido como SLEEP por lo que la generación de pulsos se detiene independientemente de si se detecta o no la presencia de un objeto en el sensor inductivo.

Modo 3 – Detección. En el firmware encontrará este modo como WAIT; en este estado, el sensor detecta la presencia de un objeto metálico y no envía pulsos al motor.

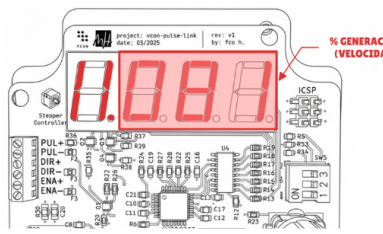
En cuanto a la velocidad del motor, se controlará mediante la variación de la frecuencia de la señal de pulsos, manteniendo un ciclo de trabajo del 50 %. El firmware muestra el valor porcentual (0-100 %) que se traduce a una frecuencia de 0 - 200 kHz

La velocidad angular resultante (n), expresada en RPM, se calcula siguiendo la relación matemática que vincula la frecuencia de pulsos con la configuración de microstep. Véase Ecuación 5.1 – Cálculo de la velocidad resultante expresada en RPM.

$$n = \frac{f_{pul}}{N_{steps} \cdot Microstep} \cdot 60$$

Ecuación 5.1 – Cálculo de la velocidad resultante expresada en RPM

Módulo de visualización - Display 7 segmentos | Ajuste de velocidad



La velocidad del motor NEMA controlado por el dispositivo PulseLINK, depende de los pulsos generados y del valor del microstep configurado del driver. El display 7 segmentos muestra el porcentaje de generación de una frecuencia variable entre 0Hz y 200kHz.

Esta información se puede ver en los tres últimos dígitos del display, siendo:

- 000 - Equivalente a un porcentaje de generación de 0Hz.
- 100 - Equivalente a un porcentaje de generación de 200 kHz.

Ante cualquier valor de generación, se mantiene constante el DUTY = 50%.

La interacción principal con el ajuste de velocidad se realiza por medio del encoder rotativo. La lógica programada interpreta las señales digitales para modificar todo el comportamiento en tiempo real a través de los siguientes comandos, véase Ilustración 5.3 – Configuración de los modos de funcionamiento mediante el encoder.

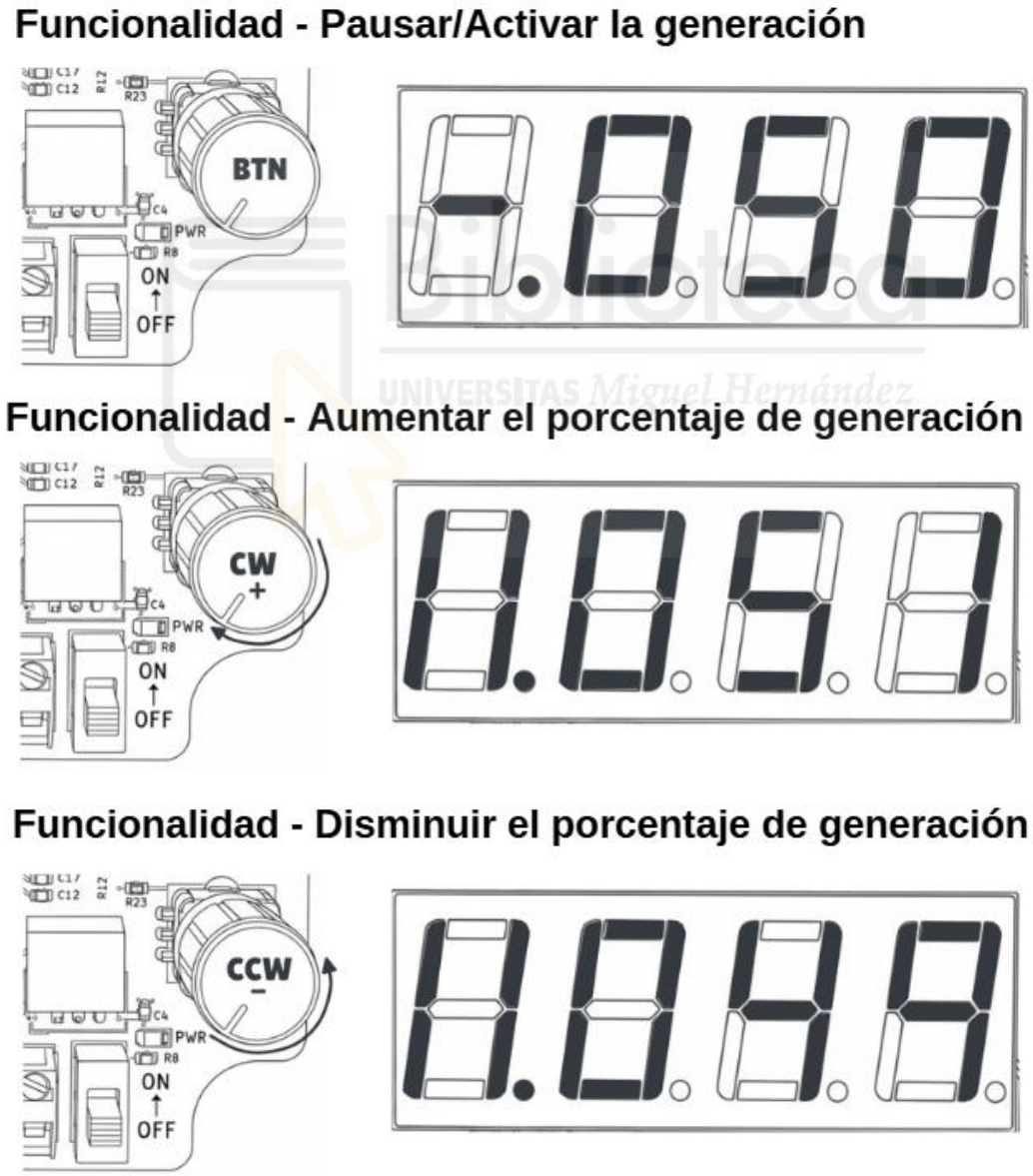


Ilustración 5.3 – Configuración de los modos de funcionamiento mediante el encoder

Finalmente, los selectores (DIP Switch) permiten definir parámetros críticos del lado de la seguridad y registrar valores específicos de memoria, véase Ilustración 5.4 - Selector DIP (SW5): ubicación en placa y detalle de configuración.

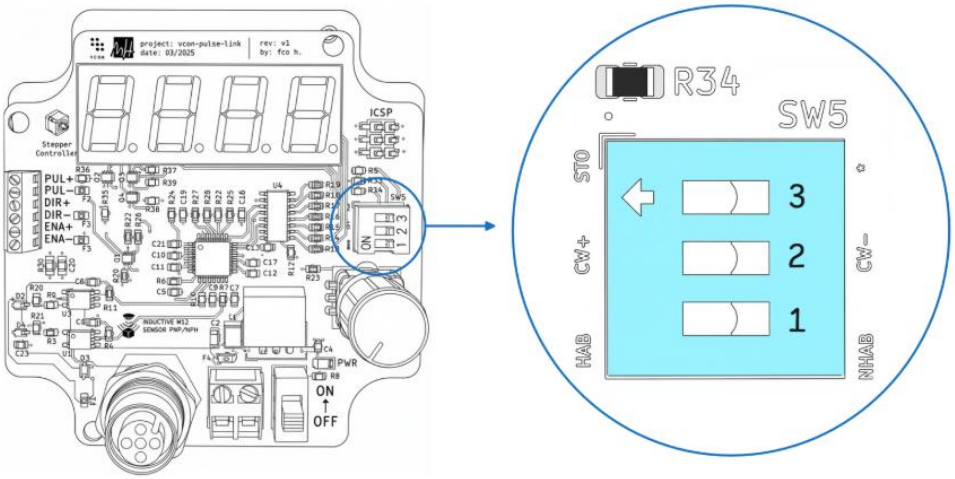


Ilustración 5.4 - Selector DIP (SW5): ubicación en placa y detalle de configuración

A continuación, se explica la utilidad de los selectores del esquema anterior:

Selector 1 (Habilitar / Inhabilitar). Capa de seguridad que sirve para inhibir el driver independientemente de lo que el firmware marque. Bloqueo de seguridad.

Selector 2 (Dirección). Define el sentido de giro del motor.

Selector 3 (Registro de memoria). Permite que el sistema recuerde la configuración que había tras un reinicio de energía, útil para no tener que reconfigurarlo todo.

Esto sería todo en cuanto a las acciones que puede utilizar el usuario para intervenir sobre el sistema. Para finalizar la descripción de la lógica de funcionamiento falta aún por explicar una funcionalidad adicional, conocida como **Arranque Suave.**

Arranque suave (Soft Start)

El arranque suave es una característica fundamental en el control de motores de tipo industrial, ya que estos actuadores no pueden responder a cambios instantáneos de frecuencia (saltos de velocidad) debido a la inercia mecánica del rotor y de la carga.

Por ejemplo, si el sistema intenta pasar de 0 Hz a una frecuencia alta de golpe, como por ejemplo 20 kHz, el motor sufrirá un bloqueo mecánico, perdiendo pasos. Lo más habitual es que esto suceda cuando el motor no es capaz de alcanzar el punto que se establece como consigna; es decir, el motor no es capaz de alcanzar la velocidad emitida por el dispositivo electrónico.

Los drivers industriales, en este punto, bloquean el motor y obligan a realizar un rearme, por lo que es necesario evitar llegar a esa situación.

Implementación del algoritmo. Para evitar ese fenómeno, el firmware no aplica el valor objetivo seleccionado por el usuario, que coincide con la consigna que aparece en el display. En su lugar, se dispone de un módulo generador que crea una rampa de aceleración lineal. El sistema compara permanentemente la velocidad actual con la velocidad deseada (objetivo) y ajusta la frecuencia de salida en intervalos constantes.

La pendiente de la rampa está determinada por la constante del firmware denominada _step_delay_ms, definida con un valor de 10 ms (Tdelay), lo que permite cuantificar con precisión el comportamiento dinámico del dispositivo, véase Ilustración 5.5 - Relación entre la aceleración y el tiempo de respuesta del motor.

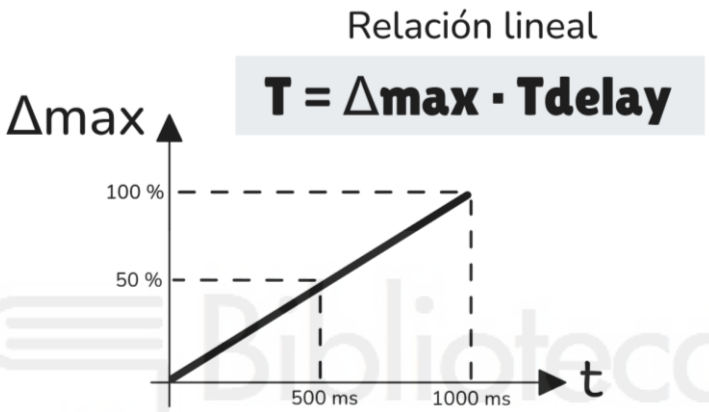


Ilustración 5.5 - Relación entre la aceleración y el tiempo de respuesta del motor

Esto implica que para una aceleración completa desde el 0 % hasta el 100 %, lo que equivale a 100 pasos porcentuales, el motor tardará 1000 ms = 1 segundo en llegar al valor máximo de velocidad establecido en la consigna.

Gracias a su lógica, el dispositivo garantiza que cualquier transición de velocidad, por brusca que sea la entrada introducida por el encoder rotativo, se va a traducir a un movimiento fluido y progresivo.

En la Ilustración 5.6 - Captura del comportamiento del SoftStart en el osciloscopio se puede visualizar el barrido de frecuencia desde el osciloscopio, asegurando un par de motor constante durante ese periodo de aceleración.

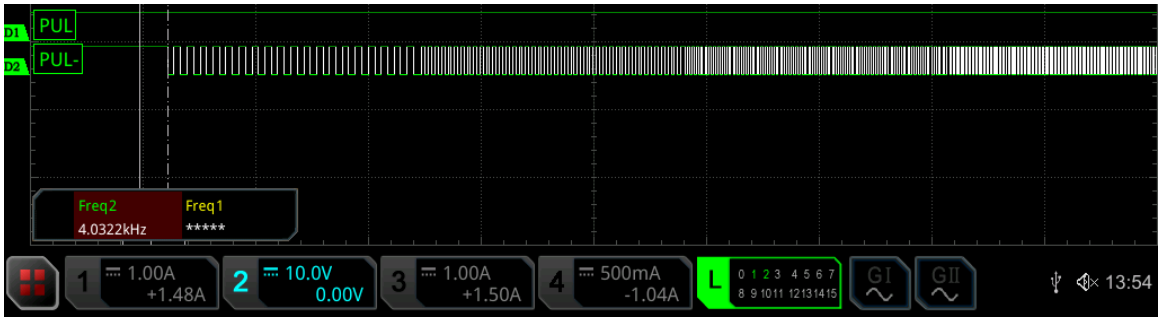


Ilustración 5.6 - Captura del comportamiento del SoftStart en el osciloscopio

6. Validación del prototipo.

Es necesario someter al sistema a una serie de ensayos con el objetivo de verificar su correcto funcionamiento. El proceso de validación del producto se estructura en dos fases diferenciadas según el entorno de pruebas y las condiciones de carga del motor:

6.1. Validación en vacío (Laboratorio) – Se realizan pruebas funcionales sobre el prototipo con el rotor del motor libre, sin carga mecánica. El objetivo de estas pruebas es verificar las señales digitales del sistema electrónico.

6.2. Validación en carga (Entorno industrial) – Se integra todo el sistema junto con la maquinaria real de la empresa Vector Conveyors. El objetivo de estas pruebas es validar la robustez del sistema y su capacidad real.

En cualquier caso, la arquitectura empleada para el control se mantiene invariable, de forma que el esquema empleado para la realización de todo el conjunto de pruebas es el que se muestra en la Ilustración 6.1 - Esquema de montaje del sistema durante las pruebas de validación.

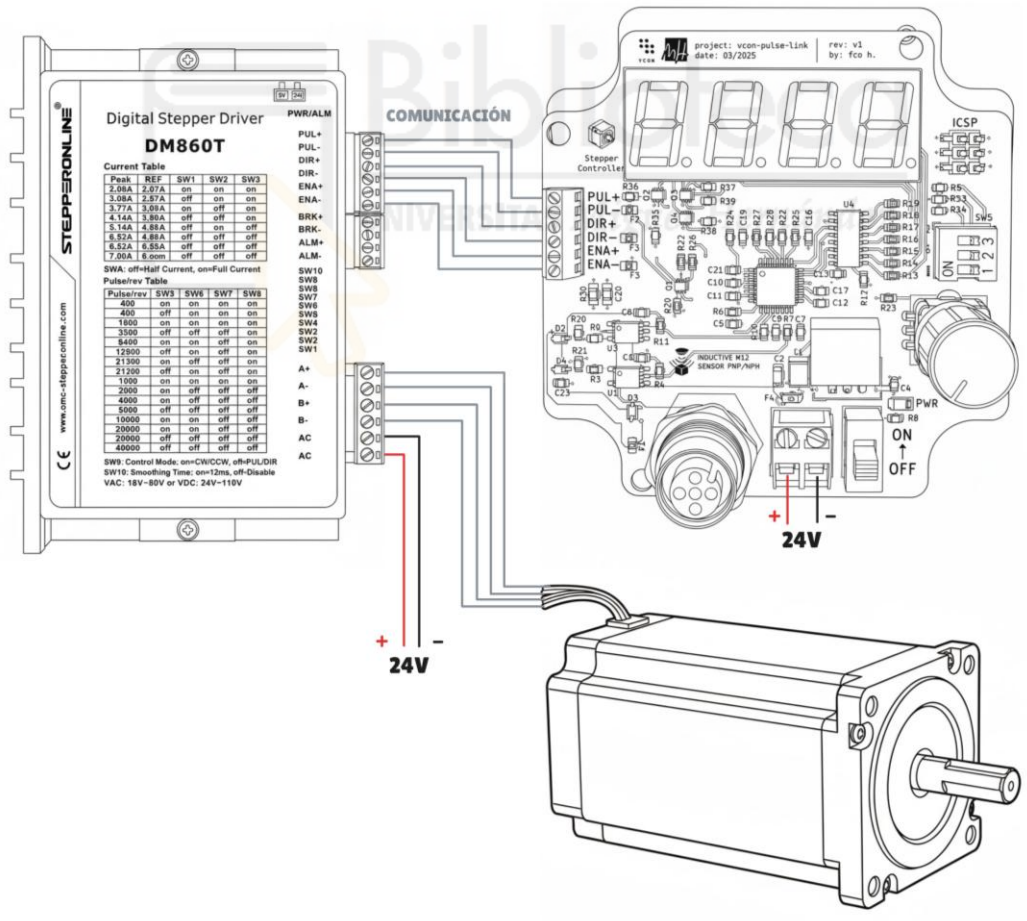


Ilustración 6.1 - Esquema de montaje del sistema durante las pruebas de validación.

6.1. Validación funcional del sistema con el motor en vacío.

Esta primera fase de validación se realiza en un entorno de laboratorio. El objetivo de este ensayo es verificar las señales eléctricas generadas por el diseño «vcon-pulse-link». El rotor se encuentra desacoplado de cualquier carga (rotor libre).

Ensayo 1: Verificación de frecuencia de generación de pulsos (Step Freq.)

El objetivo de este ensayo es comprobar que el sistema genera los pulsos de control con la frecuencia exacta correspondiente a la consigna visualizada en el display. Véase Ilustración 6.2 - Visualización de la frecuencia de generación de pulsos en el display.

Por otro lado, también se busca verificar que el valor máximo emitido por el dispositivo no exceda el ancho de banda de frecuencia permitido por el driver DM860T. Según la documentación oficial del DM860T, la frecuencia máxima admisible para los pulsos es de 200 kHz. Superar dicho límite podría saturar el optoacoplador, provocando la pérdida de pasos o el bloqueo del motor. [12]

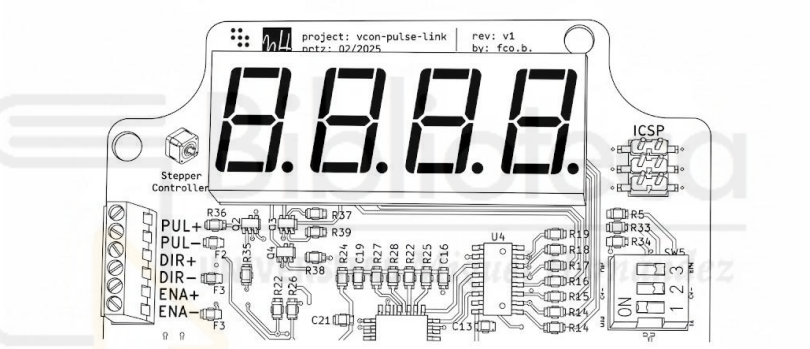


Ilustración 6.2 - Visualización de la frecuencia de generación de pulsos en el display

En cuanto al firmware desarrollado, la interfaz de usuario presenta un display con un valor porcentual que va de 0 a 100, denominado DELTA. La relación esperada entre ese valor y la frecuencia de salida del dispositivo es lineal, según la Ecuación 6.1 - Cálculo de la frecuencia de generación en función del parámetro DELTA:

$$f_{out} = \frac{\text{DELTA}}{100} \times 200\ \text{kHz}$$

Ecuación 6.1 - Cálculo de la frecuencia de generación en función del parámetro DELTA

Donde:

- **DELTA 0 –** Estado de reposo (0 kHz).
- **DELTA 50 –** 50% de la capacidad total del sistema (100 kHz).
- **DELTA 100 –** 100% de la capacidad (200 kHz).

Para el ensayo, se emplean los siguientes equipos de medida: un osciloscopio RIGOL MSO5104 y una sonda amperimétrica Testec TT-CC 990.

Con el objetivo de interpretar los resultados obtenidos en este apartado, es necesario conocer la relación matemática que vincula la frecuencia de la señal de control con la velocidad física del motor.

El cálculo de la velocidad del rotor del motor (n) depende de tres factores: la frecuencia de la señal STEP, las características físicas del motor y el microstepping.

Configuración del driver y resolución (Microstep) respecto del motor NEMA

Un motor paso a paso estándar tiene un paso de 1,8°. Esto implica que, en modo paso completo (full step), el motor necesita 200 pasos para completar una revolución. El tiempo necesario para completar un ciclo se obtiene a partir de la Ecuación 6.2 - Tiempo de una revolución en función de la frecuencia STEP.

$$T_{REV} = \frac{N_{STEPS}}{f_{STEP}} = \frac{200}{200.000} = 0.001\ s$$

Ecuación 6.2 - Tiempo de una revolución en función de la frecuencia STEP

Donde:

- **T_{REV} –** Tiempo en dar una revolución completa (s).
- **N_{STEPS} –** Número de pasos por revolución del motor.
- **F_{STEP} –** Frecuencia de la señal STEP (Hz)

Sin embargo, para garantizar suavidad y precisión, el driver se configura en modo de microstepping. Este parámetro divide cada paso físico en subpasos virtuales. Por lo tanto, el número total de pasos necesarios para que el motor dé una vuelta completa se modifica, quedando como **N_{TOTAL_STEPS}** = 200 x M, siendo M configurable en el driver.

Para este ensayo se va a utilizar un microstepping de 1/128 por seguridad.

Cálculo de la velocidad angular del rotor del motor

Finalmente, la velocidad angular del motor (n), expresada en revoluciones por minuto, se obtiene relacionando la velocidad de entrada de pulsos con la resolución aplicada, según la Ecuación 6.3 - Velocidad angular del motor.

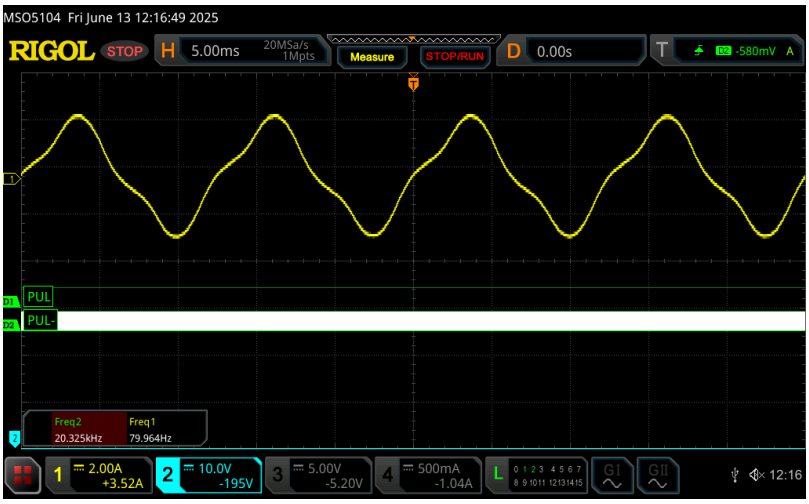
$$n = \frac{f_{STEP} \cdot 60}{200 \cdot M}$$

Ecuación 6.3 - Velocidad angular del motor

Donde:

- **f_{STEP} –** Frecuencia de la señal STEP (Hz).
- **60 –** Factor de conversión de segundos a minutos.
- **N_{STEPS} –** Pasos físicos por vuelta del motor (estándar 360º / 1,8º).
- **M –** Factor del microstepping (Pasos por vuelta).

Se muestran los resultados obtenidos para el ensayo suponiendo que DELTA varía de 10 en 10, comenzando en 0 y avanzando hasta 100.



pasos/vuelta = 200 (1.8°)

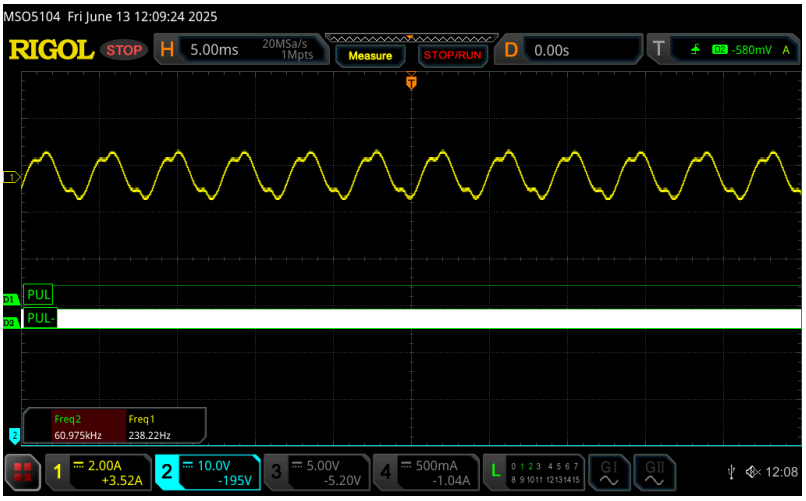
DELTA	10 %
FRECUENCIA	20 kHz
MICROSTEP	1/128
VELOCIDAD	46,875 rpm

Ilustración 6.3 – Ensayo 1. Verificación de la frecuencia de generación para 20 kHz

pasos/vuelta = 200 (1.8°)

DELTA	20 %
FRECUENCIA	40 kHz
MICROSTEP	1/128
VELOCIDAD	93,75 rpm

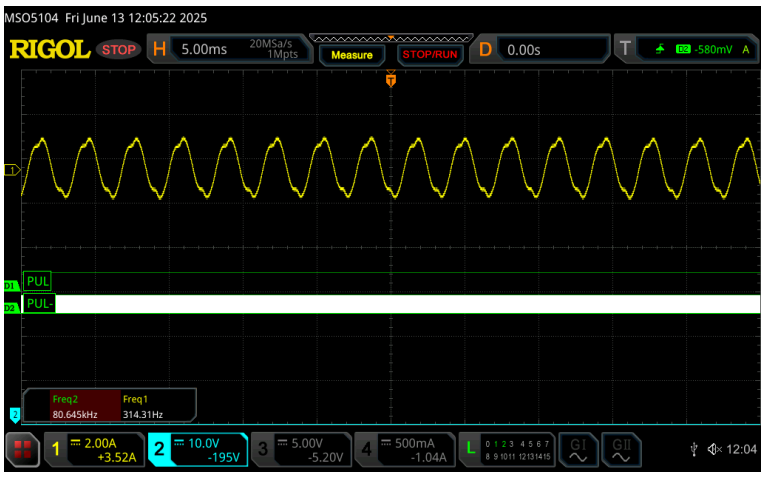
Ilustración 6.4 – Ensayo 1. Verificación de la frecuencia de generación, 40 kHz



pasos/vuelta = 200 (1.8°)

DELTA	30 %
FRECUENCIA	60 kHz
MICROSTEP	1/128
VELOCIDAD	140,625 rpm

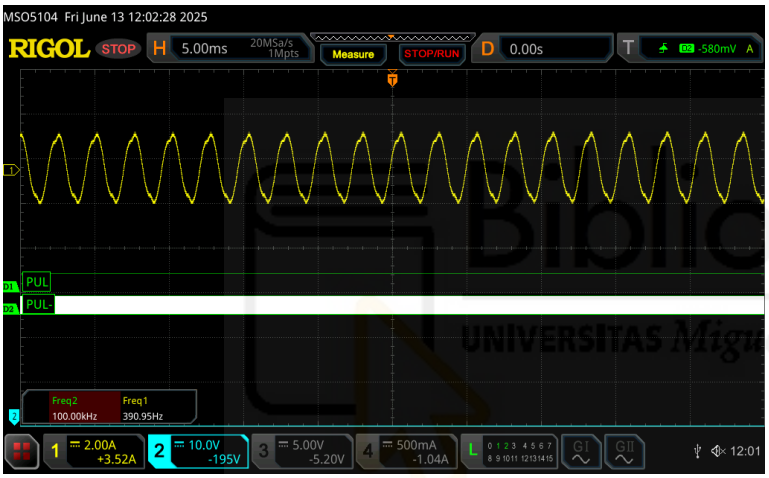
Ilustración 6.5 – Ensayo 1. Verificación de la frecuencia de generación, 60 kHz



pasos/vuelta = 200 (1.8°)

DELTA	40 %
FRECUENCIA	80 kHz
MICROSTEP	1/128
VELOCIDAD	187,5 rpm

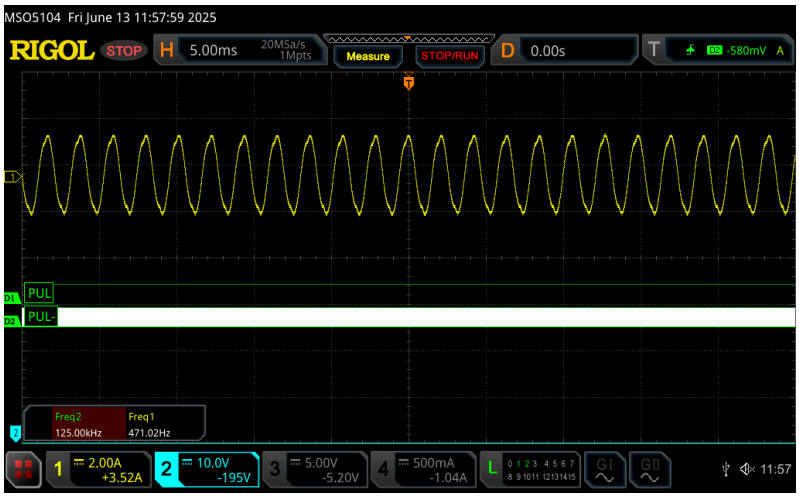
Ilustración 6.6 – Ensayo 1. Verificación de la frecuencia de generación para 80 kHz



pasos/vuelta = 200 (1.8°)

DELTA	50 %
FRECUENCIA	100 kHz
MICROSTEP	1/128
VELOCIDAD	234,375 rpm

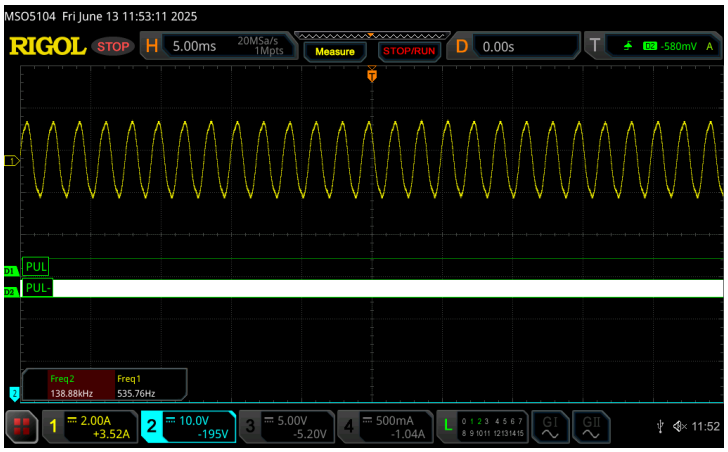
Ilustración 6.7 – Ensayo 1. Verificación de la frecuencia de generación para 100 kHz



pasos/vuelta = 200 (1.8°)

DELTA	60 %
FRECUENCIA	120 kHz
MICROSTEP	1/128
VELOCIDAD	281,25 rpm

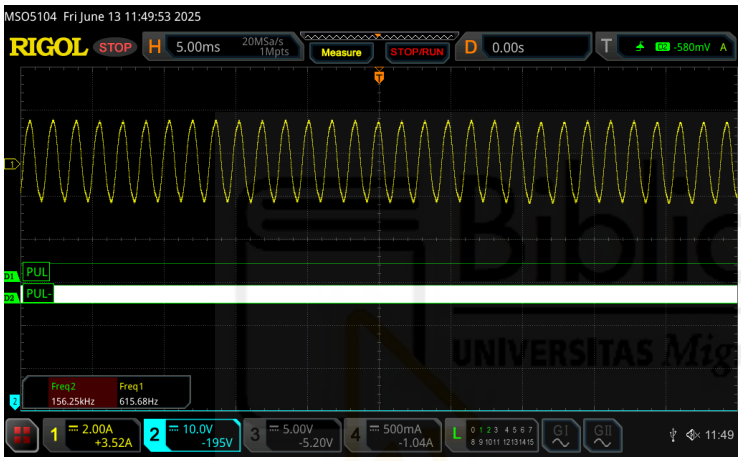
Ilustración 6.8 – Ensayo 1. Verificación de la frecuencia de generación para 120 kHz



pasos/vuelta = 200 (1.8°)

DELTA	70 %
FRECUENCIA	140 kHz
MICROSTEP	1/128
VELOCIDAD	328,125 rpm

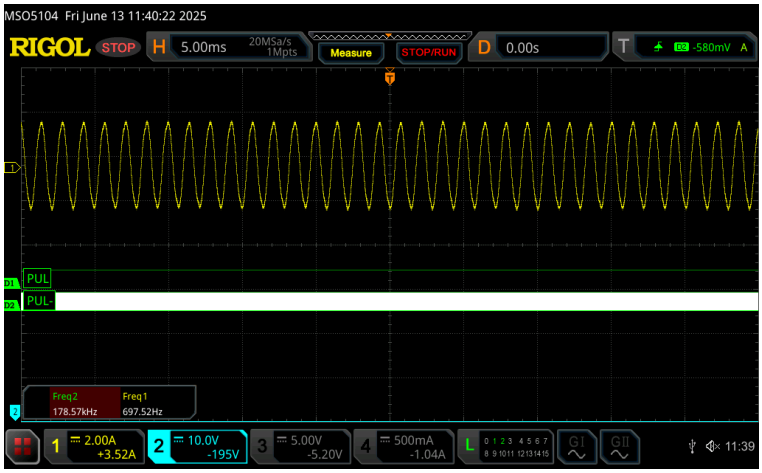
Ilustración 6.9 – Ensayo 1. Verificación de la frecuencia de generación para 140 kHz



pasos/vuelta = 200 (1.8°)

DELTA	80 %
FRECUENCIA	160 kHz
MICROSTEP	1/128
VELOCIDAD	375 rpm

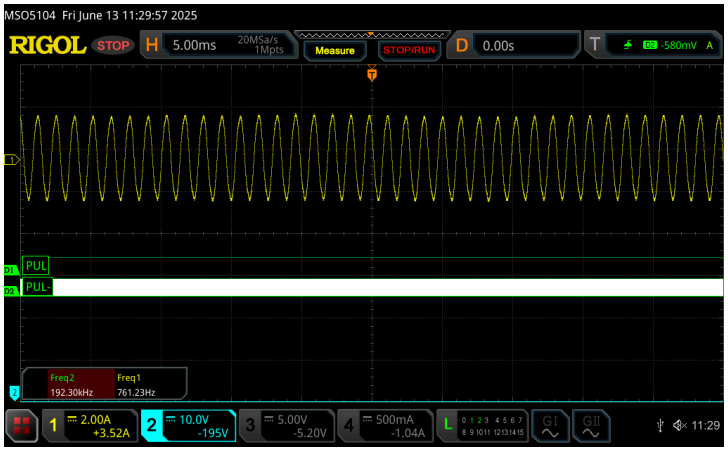
Ilustración 6.10 – Ensayo 1. Verificación de la frecuencia de generación para 160 kHz



pasos/vuelta = 200 (1.8°)

DELTA	90 %
FRECUENCIA	180 kHz
MICROSTEP	1/128
VELOCIDAD	421,875 rpm

Ilustración 6.11 – Ensayo 1. Verificación de la frecuencia de generación para 180 kHz



pasos/vuelta = 200 (1.8°)

DELTA	100 %
FRECUENCIA	200 kHz
MICROSTEP	1/128
VELOCIDAD	468,75 rpm

Ilustración 6.13 – Ensayo 1. Verificación de la frecuencia de generación para 200 kHz

pasos/vuelta = 200 (1.8°)

DELTA	0 %
FRECUENCIA	0 kHz
MICROSTEP	1/128
VELOCIDAD	0 rpm

Ilustración 6.12 – Ensayo 1. Verificación de la frecuencia de generación para 0 kHz

Ensayo 2: Tiempo de respuesta ante parada por presencia en sensor inductivo

El objetivo de este ensayo es medir la latencia del sistema ante un evento de parada. Se evaluará el tiempo transcurrido desde que el sensor inductivo detecta la presencia de un objeto hasta que el vcon-pulse-link detiene la ejecución de la señal STEP. Este es el parámetro más importante para garantizar la seguridad y la precisión del sistema.

Para llevar a cabo este ensayo, se utilizan los siguientes canales del osciloscopio:

- **Canal 1, M12 –** Señal de salida del sensor inductivo.
- **Canal 2, STEP –** Señal PUL diferencial enviada al driver DM860T.

Se provoca una detección manual acercando un objeto metálico al sensor inductivo. En este punto, el transistor interno del sensor inductivo M12 se cierra y se conecta al mismo pin de alimentación +VCC.

El osciloscopio se configura para dispararse (trigger) con el flanco de la señal del sensor, capturando a su vez el comportamiento de la señal de control STEP.

Como se puede observar en la gráfica resultante, tras el flanco de detección del sensor, el sistema tarda un tiempo en detener la generación de pulsos, que es lo que tarda en procesar el MCU el evento. Haciendo uso de los cursores de medida, se ha obtenido una latencia resultante de 47 µs.

Véase su demostración a través de la Ilustración 6.14 - Ensayo 2, tiempo de respuesta ante parada por presencia de objeto.

$$t_{reac} = 47\ [us] = 0,000047\ [s]$$

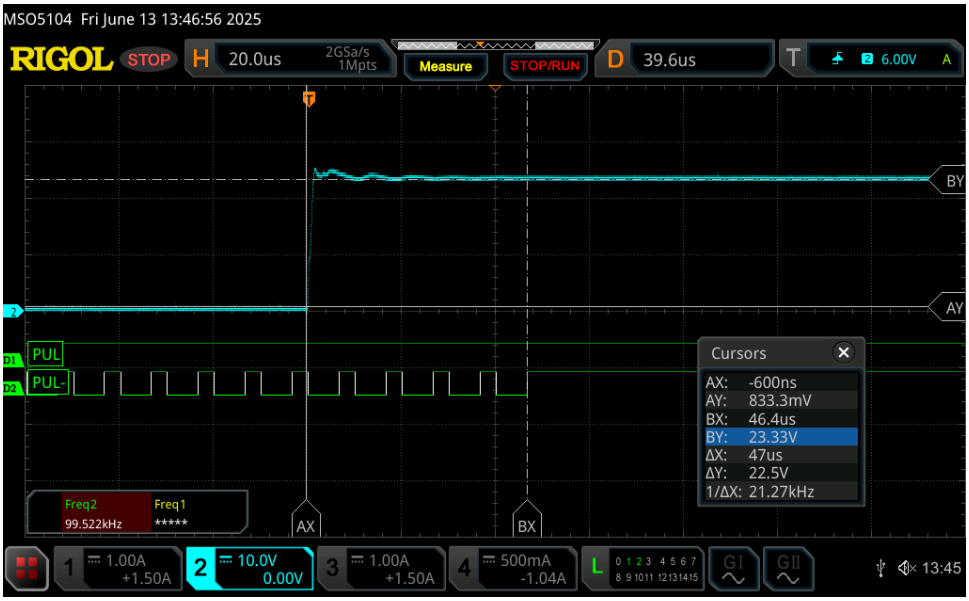


Ilustración 6.14 - Ensayo 2, tiempo de respuesta ante parada por presencia de objeto

Ensayo 3: Tiempo de respuesta del sistema ante liberación del sensor inductivo

El objetivo de este ensayo es cuantificar la latencia del sistema ante un evento de marcha. Se evaluará el tiempo total desde que el sensor inductivo deja de detectar la presencia de un objeto metálico hasta que el microcontrolador reanuda la generación de la señal STEP y, por lo tanto, el motor comienza a girar.

La configuración del banco de pruebas es idéntica a la descrita en el apartado anterior:

- **Canal 1, M12 –** Señal de salida del sensor inductivo.
- **Canal 2, STEP –** Señal PUL diferencial enviada al driver DM860T.

El sistema inicialmente se encuentra en un estado de parada por detección, situando un objeto metálico en el sensor inductivo. El motor se encuentra estático y no hay pulsos en la salida de la señal diferencial STEP.

Para realizar la medida, el método utilizado consiste en establecer en el osciloscopio un trigger en el flanco de bajada. De este modo, al retirar el objeto metálico del sensor inductivo, se captura el momento exacto en el que comienzan los pulsos.

Tras el flanco de bajada, el sistema tarda un tiempo en activar la generación de pulsos, correspondiente al tiempo necesario para que el MCU procese el evento. Haciendo uso de los cursores de medida, se ha obtenido una latencia resultante de 1,062 ms.

Véase su demostración a través de la Ilustración 6.15 - Ensayo 3, tiempo de respuesta ante liberación de objeto.

$$t_{reac} = 1,062\ [us] = 0,001062\ [s]$$

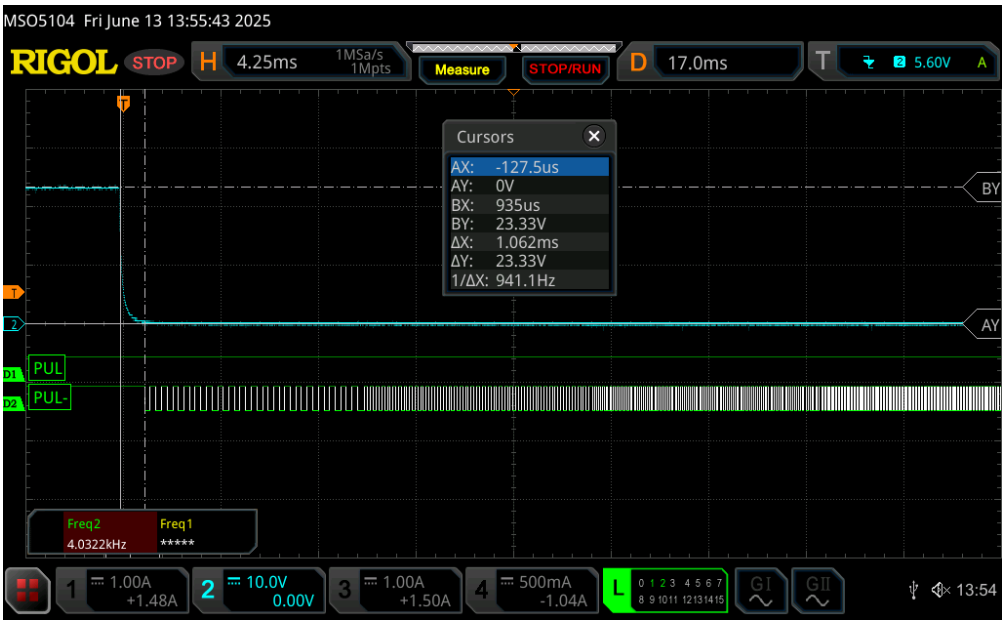


Ilustración 6.15 - Ensayo 3, tiempo de respuesta ante liberación de objeto

6.2. Validación funcional del sistema con el motor en carga.

Una vez que se ha certificado que las señales digitales de control son correctas (ensayo en vacío, pruebas en el laboratorio), la siguiente fase consiste en validar el sistema en un entorno industrial real.

En esta etapa, el producto se traslada a las instalaciones de *Vector Conveyors* con el objetivo de certificar que el sistema electrónico es capaz de controlar el motor bajo condiciones de carga nominales.

Véase Ilustración 6.16 - Estructura general del sistema de transporte utilizado por Vector Conveyors, donde se muestra la configuración del sistema de transporte actual y se destaca la ubicación del transfer de correas, sobre el que se ejecutarán todas las maniobras de puesta en marcha.

Ilustración 6.16 - Estructura general del sistema de transporte utilizado por Vector Conveyors

Para la ejecución del ensayo fue necesaria la intervención del cotutor del proyecto, Cristóbal Parra, quien llevó a cabo el montaje del producto electrónico en el sistema industrial, junto con la integración del cableado.

A continuación, se exponen imágenes que detallan los pasos seguidos para realizar el montaje, destacando las ubicaciones intervenidas. En la ilustración inicial se muestra el sistema sin ningún actuador instalado, mientras que en la segunda ilustración se muestra el transfer de correas con los elementos necesarios para el funcionamiento del sistema.

Véase Ilustración 6.17 - Sistema de transporte sin actuador conectado y Ilustración 6.18 - Sistema de transporte con producto electrónico integrado.

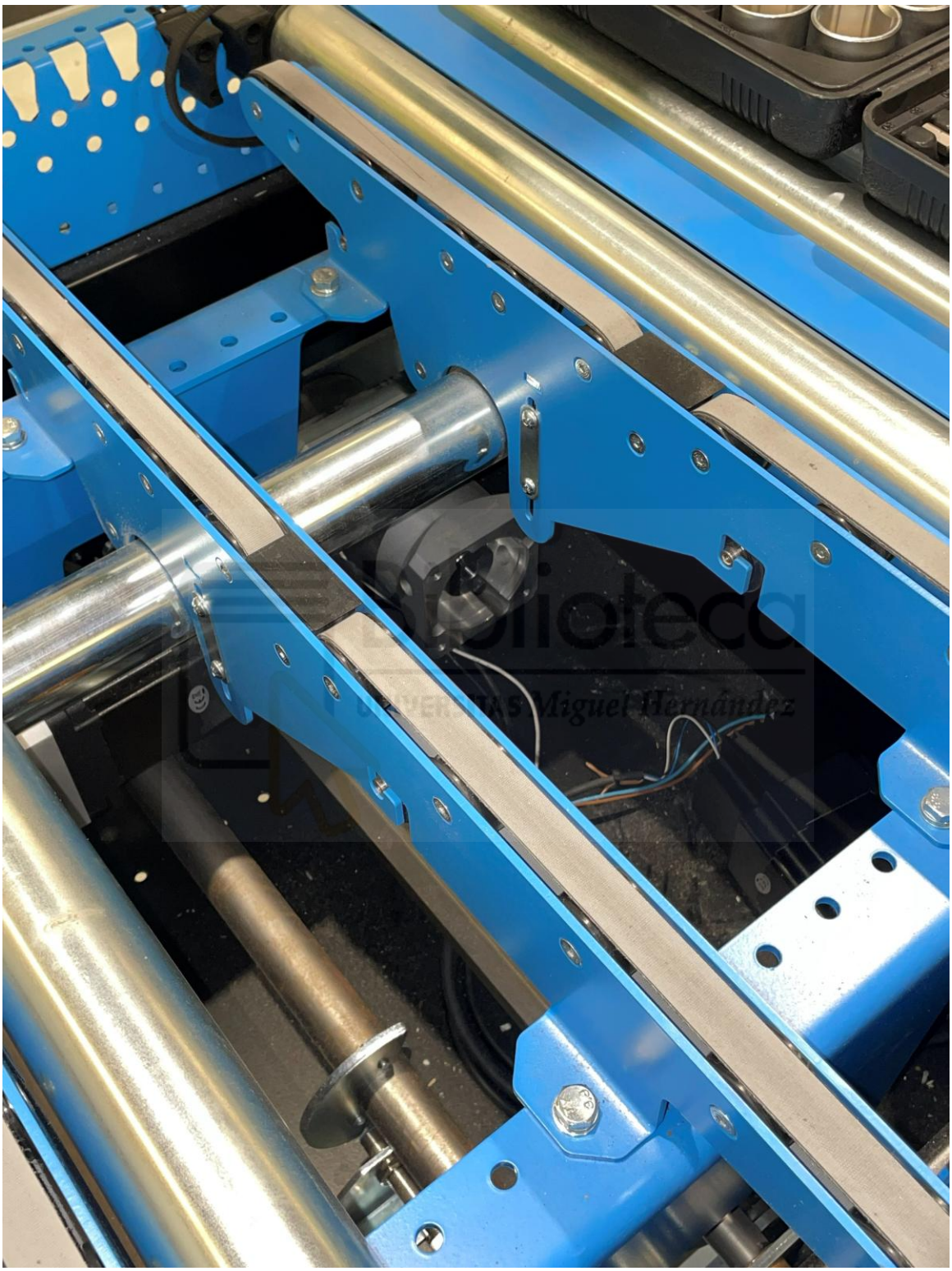


Ilustración 6.17 - Sistema de transporte sin actuador conectado

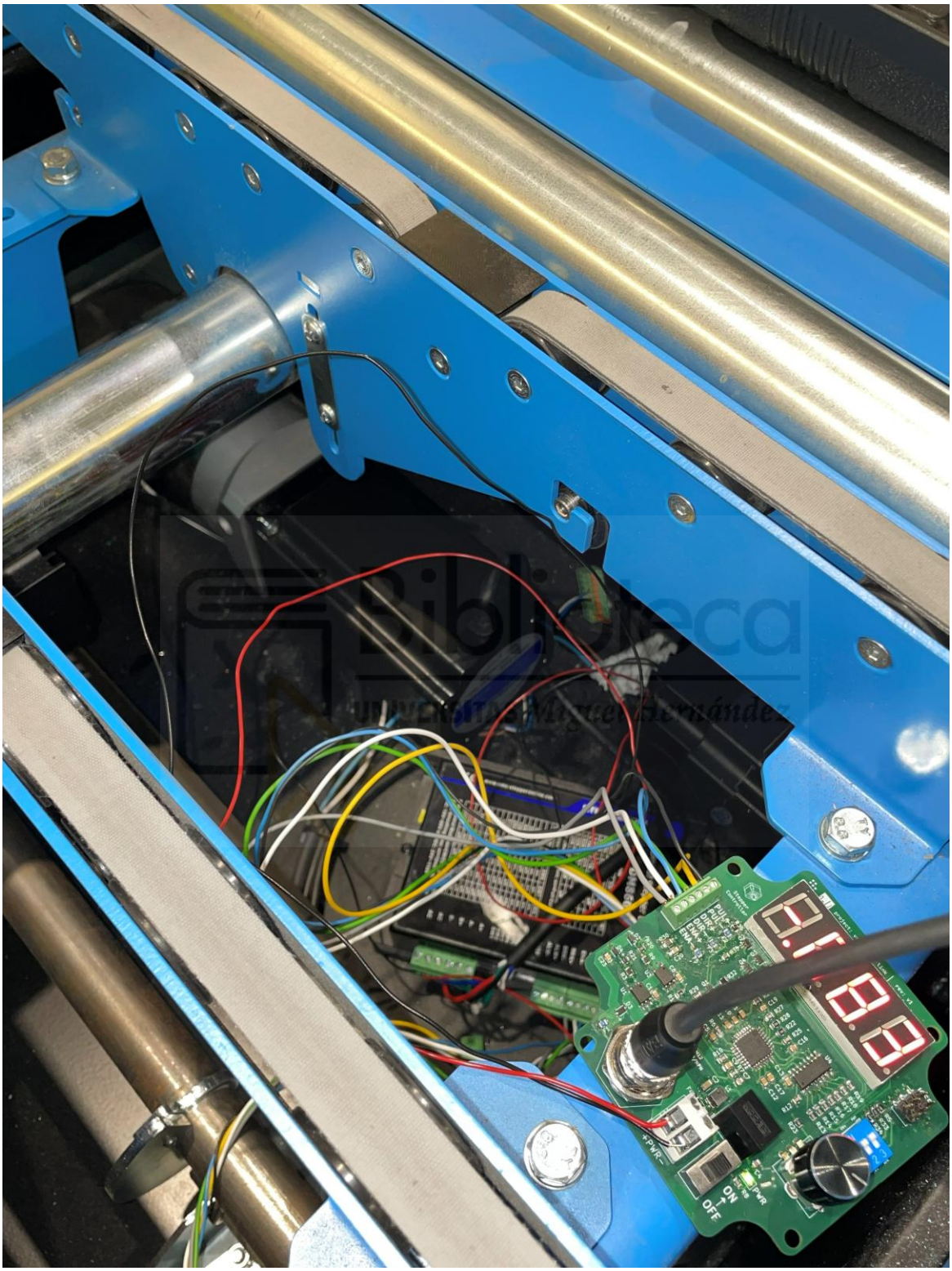


Ilustración 6.18 - Sistema de transporte con producto electrónico integrado

Durante las pruebas de funcionamiento se detectó un comportamiento erróneo relacionado con el par-velocidad del motor. Partiendo de una configuración con una resolución del driver equivalente a un microstep 1/128, se observó que el motor no era capaz de seguir la generación de pulsos (frecuencia) impuesta por el sistema.

Este fenómeno provocaba pérdida de sincronismo instantánea, debido a que el par resistente de la carga era superior al par de arranque del motor. Por ello, en la mayoría de las aplicaciones industriales se suele utilizar un variador de frecuencia que asegure, a revoluciones bajas, un par más alto y, cuando el sistema adquiere cierta inercia, se aumentan las revoluciones del motor variando de nuevo la frecuencia.

El sistema implementa este barrido mediante un arranque suave, pero en algunos casos no fue suficiente, especialmente al colocar una carga que ralentizaba la respuesta. Para mejorar la estabilidad, podría configurarse un barrido más lento, a costa de un mayor tiempo de respuesta. De este modo, el sistema sería capaz de seguir los pulsos enviados por el dispositivo de control.

En la siguiente Ilustración 6.19 - Curva característica par-velocidad de un motor eléctrico, se observa una curva de par típica para un motor eléctrico. Si no es capaz de alcanzar una velocidad de sincronismo acorde con el par de arranque, el motor no accederá a la zona estable.

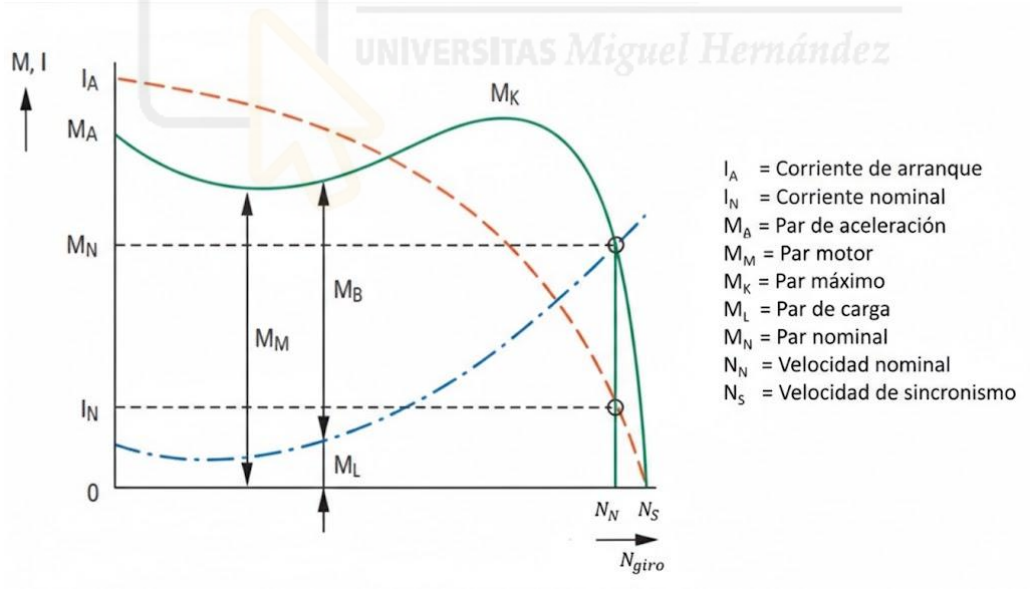


Ilustración 6.19 - Curva característica par-velocidad de un motor eléctrico

Aunque los motores paso a paso como los NEMA funcionan mediante un principio distinto al representado en la curva de par mostrada en la ilustración superior, a nivel conceptual el problema es similar: para iniciar el movimiento es necesario vencer el par resistente de la carga. En este contexto, el arranque debe realizarse con una frecuencia inicial baja para evitar la pérdida de sincronismo.

En cualquier caso, para subsanar este inconveniente y garantizar que el motor fuese estable ante cualquier situación, se analizó de forma conceptual la relación entre la velocidad de giro y la resolución mediante dos estrategias:

Modificación de la resolución – Reducir el divisor de pasos solucionaba el problema, ya que al disponer de una menor velocidad máxima se obtenía un mayor par incluso con porcentajes elevados de generación de frecuencia. Únicamente sería necesario verificar que el motor no presentase vibraciones excesivas a frecuencias más bajas.

Ajuste del porcentaje de generación de frecuencia – Al reducir el parámetro DELTA se disminuye la velocidad de consigna, lo que sitúa el motor en una zona de par más favorable. Esta fue la solución adoptada durante el ensayo.

DESTACADO

Tras realizar un barrido experimental de varias velocidades, se determinó el límite de operación del sistema. Para una resolución de 1/128, el valor máximo de DELTA que podía establecerse sin que el motor bloqueara fue DELTA = 76. Esto implica que la velocidad máxima en condiciones nominales es de 356,25 rpm.

Por lo tanto, este ensayo permitió identificar el límite de funcionamiento del driver DM860T bajo dichas condiciones de carga.

Procedimiento para efectuar el ensayo

Para comprender cómo el sistema gestiona este mecanismo, es necesario analizar el interior del eje del transfer. En su interior se encuentra acoplada una leva metálica con forma de media luna que gira solidariamente con el mecanismo de elevación. Esta pieza constituye la referencia física que permite determinar la posición del sistema:

- Cuando el transfer efectúa el movimiento de descenso, la parte metálica de la leva se alinea con el sensor inductivo del sistema, activándolo.
- Cuando el transfer asciende, la parte vacía de la leva pasa frente al sensor, desactivándolo al no existir superficie metálica frente a este.

Véase Ilustración 6.20 - Sistema de detección inductiva mediante leva metálica solidaria al eje.

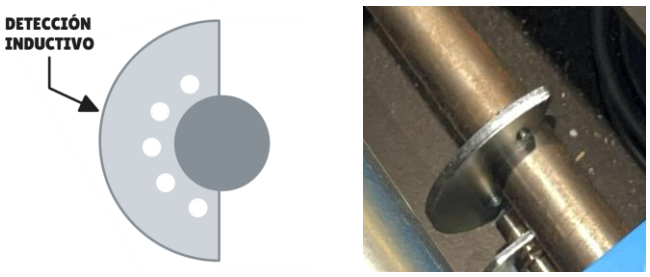


Ilustración 6.20 - Sistema de detección inductiva mediante leva metálica solidaria al eje

El objetivo de diseño es que el transfer se mantenga siempre en la posición superior, listo para clasificar la mercancía. Para lograrlo de forma robusta, se ha implementado una estrategia de control basada en la interacción de dos sensores.

La lógica es la siguiente: se ha instalado un **sensor de presencia** que detecta si hay una caja sobre las correas. Este sensor de presencia **alimenta al sensor inductivo con lógica negada.**

Esto genera la siguiente secuencia de funcionamiento:

1. **Llegada de la carga.** En el momento en que una caja entra al transfer, el sensor de presencia se activa y corta la alimentación eléctrica del sensor inductivo. Al desalimentar el sensor inductivo, el sistema mecánico accionado por el motor NEMA desciende. En condiciones normales, esto no ocurriría, ya que la media luna correspondiente a la posición de descenso es metálica; de ahí la necesidad de desactivar eléctricamente el sensor inductivo.
2. **Reinicio del ciclo.** Una vez que la caja desaparece, el sensor de presencia con lógica negada se desactiva y se restablece la alimentación del sensor inductivo. En ese instante, el sistema se encuentra en la zona correspondiente a la parte no metálica de la leva, por lo que el motor se activa hasta alcanzar nuevamente la zona metálica, coincidiendo el inicio de esta con la posición superior.

El sistema actual dispone de este método de lectura, por lo que se ha adaptado la lógica de funcionamiento del dispositivo a dicha arquitectura.

Véase Ilustración 6.21 - Secuencia de estados del transfer en presencia de carga.

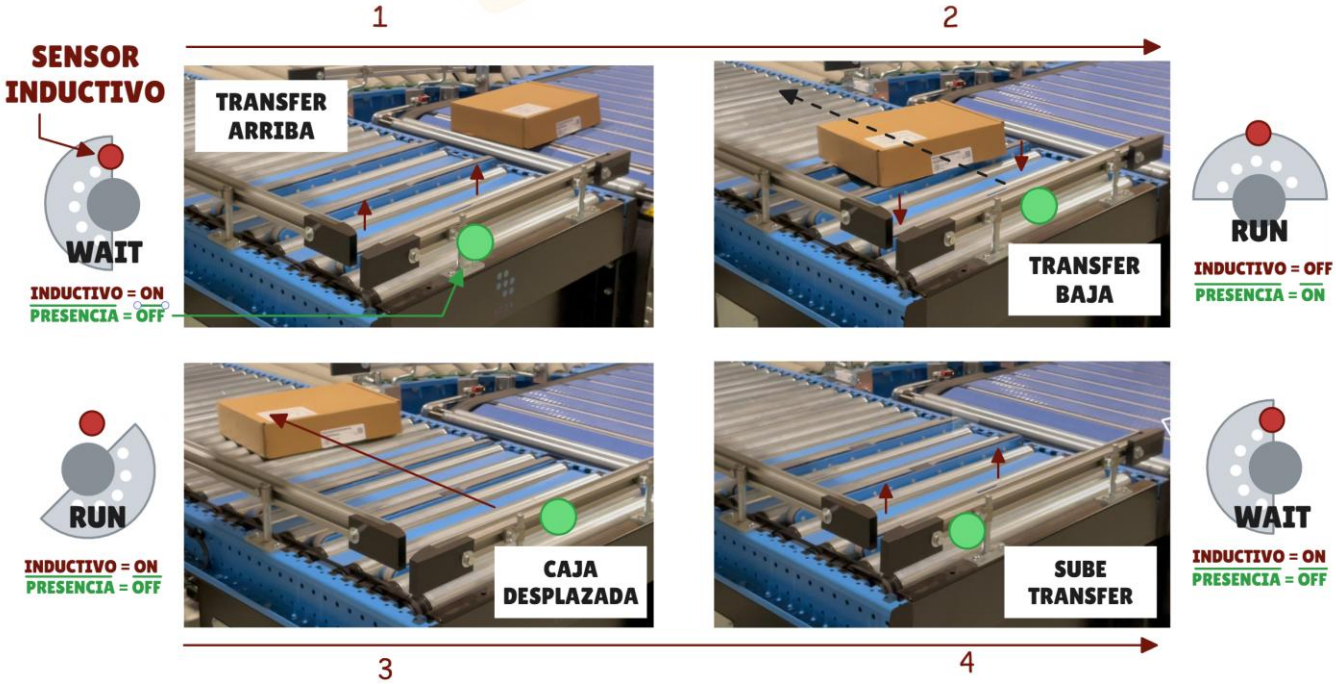


Ilustración 6.21 - Secuencia de estados del transfer en presencia de carga

7. Conclusiones.

El desarrollo del proyecto ha concluido de **forma satisfactoria**, permitiendo validar y consolidar el propósito principal de este Trabajo Final de Grado: **diseñar, fabricar y validar un prototipo electrónico funcional capaz de actuar como interfaz de control para motores paso a paso de alto par en entornos industriales**. Este sistema no solo resuelve un problema real para la empresa *Vector Conveyors*, sino que también sirve como una plataforma de desarrollo robusta para futuros ensayos experimentales.

A partir de las pruebas realizadas tanto en laboratorio como en planta, se confirma el cumplimiento de los objetivos generales e hitos específicos marcados al inicio:

- ✓ **Automatización e independencia operativa:** Se ha logrado automatizar el accionamiento de sistemas electromecánicos (transfer de correas). Al dotar al sistema de la capacidad de generar de forma autónoma sus propias señales digitales de control, se ha cumplido con el objetivo de eliminar la dependencia técnica de autómatas programables (PLC) para estas tareas.

- ✓ **Sintetización de señales y regulación flexible:** El hardware genera con total precisión las señales de temporización exigidas por el driver industrial DM860T. Asimismo, se ha cumplido el objetivo de regular la velocidad en tiempo real mediante un encoder rotativo. Esta característica dota al sistema de una gran flexibilidad, permitiendo ajustar dinámicamente la frecuencia de generación de pulsos en función de las exigencias de carga del motor en cada escenario.

- ✓ **Interfaz de usuario para gestión:** Se han implementado satisfactoriamente los controles físicos de la interfaz, logrando gestionar de forma eficaz tanto el sentido de giro del actuador (mediante selector) como el encendido y apagado seguro del sistema y del motor.

- ✓ **Arquitectura de software y persistencia:** El núcleo del microcontrolador ha sido programado con éxito basándose en una arquitectura de estados finitos (*Run, Sleep, Wait*). Además, se ha garantizado la persistencia de los datos del sistema, verificando que la memoria no volátil (EEPROM) almacena y recupera correctamente el último modo de operación tras un corte de suministro.

- ✓ **Interoperabilidad:** El dispositivo ha mostrado una excelente integración con la maquinaria de la planta industrial. El hardware de acondicionamiento procesa de forma segura las lecturas de los sensores industriales sin captar ruido en forma de interferencias, permitiendo al sistema reaccionar adecuadamente.

DESTACADO

El único objetivo general planteado que se ha proyectado como una **meta a largo plazo** es la reducción de la dependencia tecnológica frente a suministradores externos de drivers, lo cual abre el camino a las siguientes fases de desarrollo del producto.

Durante las fases finales del TFG se han realizado pruebas experimentales adicionales que excedían los objetivos, logrando verificar la viabilidad de comunicación directa con un circuito integrado controlador de motores en encapsulado SMD. En concreto se ha logrado conectar esta interfaz electrónica al componente TMC262, véase Ilustración 7.1 - Módulo controlador TMC262 encapsulado SMD.

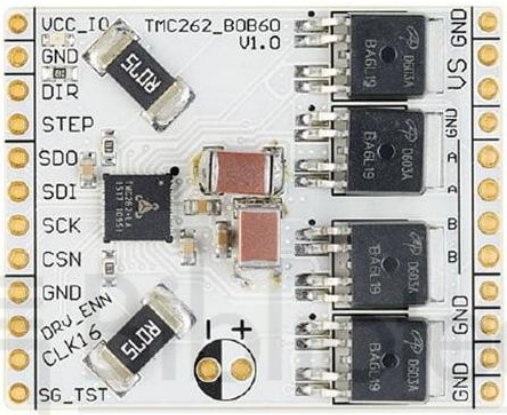


Ilustración 7.1 - Módulo controlador TMC262 encapsulado SMD

Por tanto, la principal línea de mejora y evolución del proyecto vcon-pulse-link consiste en realizar un nuevo diseño electrónico que integre dicho componente en la propia placa base. Esta evolución permitiría eliminar por completo la necesidad de un driver industrial externo DM860T, logrando un accionamiento completamente autónomo de motores tipo NEMA, ofreciendo a la empresa una solución tecnológica «todo en uno» (controlador lógico + etapa de potencia).

Debe considerarse que **un mayor grado de especialización electrónica** puede implicar una disminución de compatibilidad del sistema frente a distintas aplicaciones industriales específicas**.** En consecuencia, como línea prioritaria de trabajo futuro, será necesario analizar experimentalmente si la implementación de un controlador integrado como el TMC262, complementado con una etapa de potencia basada en transistores MOSFET externos, resulta adecuada para la totalidad de escenarios productivos de la empresa colaboradora del proyecto.

Todo ello con el objetivo de comprobar su encaje real en el mercado industrial y determinar si resulta económicamente rentable invertir en su producción.

8. Referencias bibliográficas.

[1] Autodesk, "Electronic design automation (EDA)," Autodesk fusion for electronic design automation, [Online]. Available: https://www.autodesk.com/solutions/eda-electronic-design-automation.

[2] KiCad, «KiCad Docs,» KiCad , 18 Febrero 2025. [En línea]. Available: https://docs.kicad.org/9.0/en/introduction/introduction.html.

[3] Altium, «Altium Docs,» Altium, 2025. [En línea]. Available: https://www.altium.com/es/altium-designer?srsltid=AfmBOordNQzjH-HKrJcc1T0zPTTWUh-jBwc5Q_LJQya0IWXnT3Jch6bY.

[4] Autodesk, «Autodesk Eagle,» Autodesk , 2025. [En línea]. Available: https://www.autodesk.com/br/products/eagle/overview.acessado#internal-link-what-is-eagle.

[5] V. S. Code, «code,» Microsoft, 10 Diciembre 2025. [En línea]. Available: https://code.visualstudio.com/docs/getstarted/getting-started.

[6] Arduino, «Software de Arduino,» Arduino, [En línea]. Available: https://arduino.cl/programacion/?srsltid=AfmBOorYqv6I-n1yDBpocrQ6t80WBD7QkhCGbLSpzNO6sDnAHkPv3-zV.

[7] Arduino, «Arduino Docs CLI,» Arduino, 2026. [En línea]. Available: https://docs.arduino.cc/arduino-cli/.

[8] R. P. GmbH, «R-78E5.0-0.5 DC/DC Converter Datasheet,» RECOM Power GmbH, Garching, Germany, 2017.

[9] M. T. Incorporated, «Data Sheet: ATmega48A/PA/88A/PA/168A/PA/328/P,» Microchip Technology Inc., Chandler, Arizona, EE. UU, 2020.

[10] Kingbright, «5614AS 14.2 mm (0.56 inch) Single Digit LED Display Datasheet,» Kingbright Electronic Co., Ltd., Taipei, Taiwan, 2015.

[11] Nexperia, «74HC595; 74HCT595 8-bit shift register with output latches; 3-state,» Nexperia B.V., Nijmegen, The Netherlands, 2018.

[12] O. Corporation, «DM860T Digital Stepper Driver User Manual,» OMC Corporation, Shenzhen, China, 2020.

UNIVERSITAS Miguel Hernández

VCON

ANEXO I

Bases de diseño y especificaciones técnicas

vcon-pulse-link

Interfaz Electrónica de Control Industrial

Anexos Justificativos | Versión 1.0

UMH - VCON

Tutoría TFG - Diseño de un dispositivo electrónico con GRBL para el control de una CNC personalizada

VCON

UNIVERSITAS Miguel Hernández

Materia	Realizado por	Fecha	Rev.	Fecha de revisión
Tutoría TFG	C. Parra	31/03/2023	0	10/10/2024

CONTENIDO

1. REVISIONES DEL DOCUMENTO

Revisión	Fecha	Editado por	Revisado por	Cambio
0	10/10/24	C. Parra	-	Creación del documento

2. INFORMACIÓN GENERAL

El presente documento pretende establecer las bases para la colaboración entre Fran Hidalgo, estudiante de la universidad Miguel Hernández de Elche (UMH) y Vector Conveyors (VCon), empresa dedicada a la fabricación de maquinaria ubicada en Muro de Alcoy. El tutor educacional será Dr. David Marroquí Sempere y el tutor de empresa será Cristóbal Parra Soriano.

3. PLANIFICACIÓN DE RECURSOS

Tras su estudio previo, VCon podrá facilitar ciertos componentes para pruebas reales como motores y drivers disponibles en el taller de VCon.

4. MOTORES

4.1 MODELO A: MOTOR DC SIN ESCOBILLAS CON DRIVER INTEGRADO

Este motor está equipado con un chip controlador de motor CC dedicado de alto rendimiento, que tiene una serie de ventajas como alta integración, tamaño pequeño, protección completa, cableado simple y claro y alta confiabilidad. Esta serie incluye una variedad de motores integrados que van desde 100 a 400W. Además, el controlador incorporado está equipado con la nueva tecnología PWM, que permite que el motor sin escobillas funcione a alta velocidad, con poca vibración, bajo nivel de ruido, buena estabilidad y alta confiabilidad.

Enlace:

https://www.omc-stepperonline.com/es/motor-cc-sin-escobillas-integrado-nema-34-86x86x168mm-200w-24v-1500rpm-1-27nm-8-33a-con-controlador-86ibl200-2415

4.1 MODELO B: NEMA 34 MOTOR PASO A PASO

Serie E Nema 34 Motor paso a paso Bipolar 1.8deg 12.0Nm(1699.34oz.in) 6.0A 86x86x151.5mm 4 cables.

Enlace:

https://www.omc-stepperonline.com/es/serie-e-nema-34-motor-paso-a-paso-bipolar-1-8deg-12-0-nm-1699-68oz-in-6-0a-86x86x151-5mm-4-cables-34he59-6004s

Driver para control del motor paso a paso.

Enlace:

https://www.omc-stepperonline.com/es/controlador-paso-a-paso-digital-2-4-7-2a-18-80vac-o-36-110vdc-para-motor-nema-34-dm860t?srsltid=AfmBOoruBeER6EVq4UMnLusFxl0pWljoSXWWLXdLFxUe7aj2fdXo2sXd

4.1 MODELO C: MOTOR BLDC

Motor BLDC de engranajes 24V 172W 350RPM 4.00Nm(566.45oz.in) 10:1.

24V 172W 350RPM Geared Brushless DC Motor 4.00Nm(566.45oz.in) 10:1 Caja de cambios de alta precisión - 57BLR70-24-02-HG10|STEPPERONLINE (omc-stepperonline.com)

5. OBJETIVOS

El objetivo principal de VCon es poder controlar motores MODELO B y MODELO C sin driver a través de una placa de control de diseño propia para no depender de controladores externos de otros fabricantes.

En este momento se utilizan motores MODELO A debido a su facilidad de incorporación e integración en las máquinas al no necesitar driver externo ni CPU de control para generación de pulsos.

Los motores MODELO A se controlan mediante señales digitales. A continuación, se describen las señales requeridas por el motor MODELO A según Figura 1.

Entrada digital 1: Marcha motor en sentido horario.

Entrada digital 2: Marcha motor en sentido antihorario.

Salida digital 1: Señal de alarma del motor.

Entrada analógica 0-5VDC: Control de velocidad.

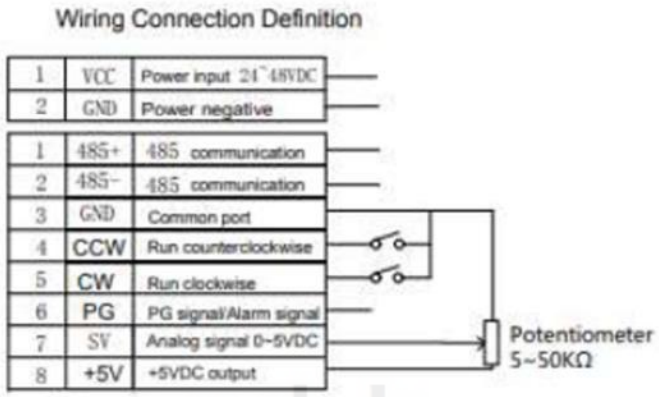


Figura 1. Conexionado motor MODELO A.

5.1 OBJETIVO 1

VCon espera poder disponer de una placa de control para el control de un motor MODELO B.

La placa debería ser capaz de recibir y gestionar las siguientes señales de control.

Entrada digital 1: Marcha motor en sentido horario.

Entrada digital 2: Marcha motor en sentido antihorario.

Entrada analógica 0-5VDC: Control de velocidad (este punto no es crítico y podría no implementarse en la primera versión).

La placa debería ser capaz de emitir las siguientes señales.

Salida digital 1: Señal de alarma del motor en caso de fallo o error(este punto no es crítico y podría no implementarse en la primera versión).

5.1 OBJETIVO 2

VCon espera poder disponer de una placa de control para el control de un motor MODELO C.

La placa debería ser capaz de recibir y gestionar las siguientes señales de control.

Entrada digital 1: Marcha motor en sentido horario.

Entrada digital 2: Marcha motor en sentido antihorario.

Entrada analógica 0-5VDC: Control de velocidad (este punto no es crítico y podría no implementarse en la primera versión).

La placa debería ser capaz de emitir las siguientes señales.

Salida digital 1: Señal de alarma del motor en caso de fallo o error (este punto no es crítico y podría no implementarse en la primera versión).

5.1 OBJETIVO 3

VCon espera poder disponer de una placa de control para el control de un motor MODELO B y MODELO C.

La placa debería ser capaz de recibir y gestionar las siguientes señales de control.

Entrada digital 1: Marcha motor en sentido horario.

Entrada digital 2: Marcha motor en sentido antihorario.

Entrada digital 3: Envío de 400 pulsos al motor en sentido horario.

Entrada digital 4: Envío de 400 pulsos al motor en sentido antihorario.

Entrada analógica 0-5VDC: Control de velocidad.

La placa debería ser capaz de emitir las siguientes señales.

Salida digital 1: Señal de alarma del motor en caso de fallo o error.

6. COMENTARIOS

La placa de control debe estar preparada para soportar intensidades de hasta 10A en 24VDC.

Los desarrollos futuros contemplan el funcionamiento de motores a 48VDC (solo potencia). La maniobra de las entradas y salidas digitales siempre será a 24VDC.

UNIVERSITAS Miguel Hernández

VCON

ANEXO II

Manual de instrucciones del sistema vcon-pulse-link

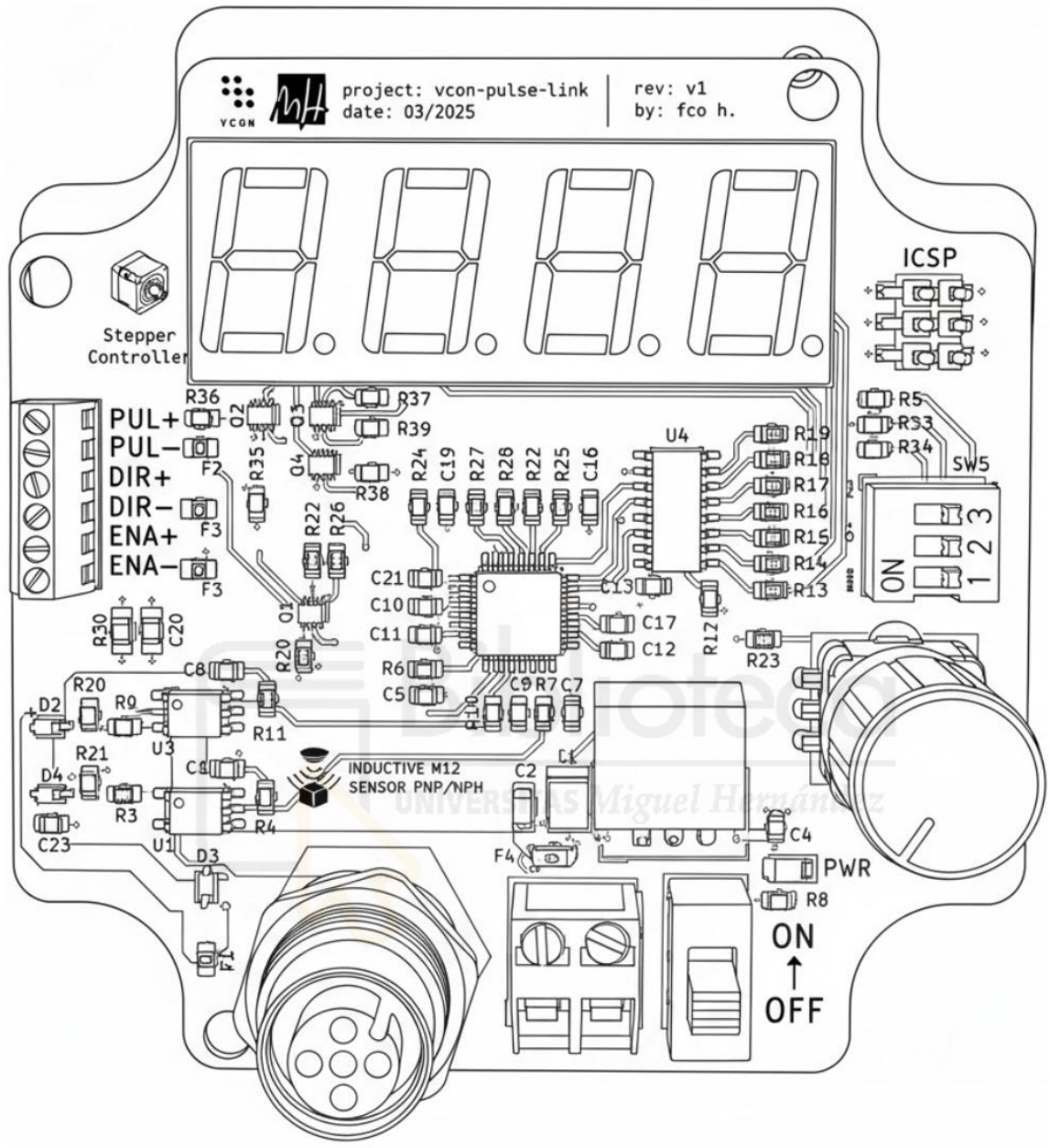
vcon-pulse-link

Interfaz Electrónica de Control Industrial

Anexos Justificativos | Versión 1.0

UNIVERSITAS Miguel Hernández

VCON



⚠ Lea este manual atentamente antes de utilizar este producto

vcon-pulse-link

Interfaz Electrónica de Control Industrial

Manual de instrucciones | Versión 1.0

1. Introducción del vcon-pulse-link

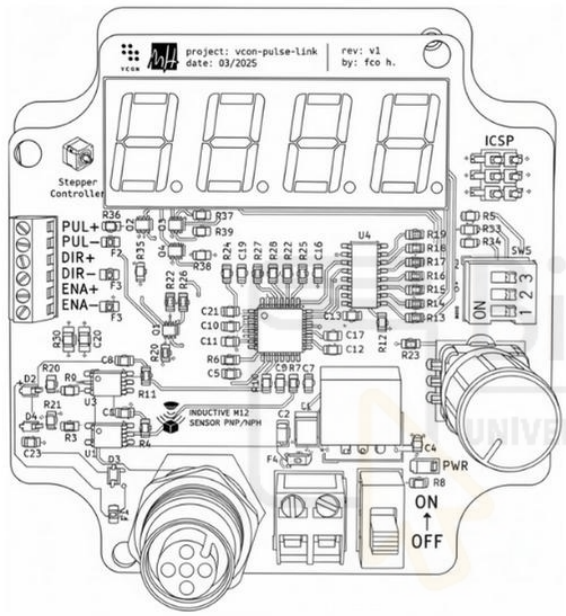
1.1. Presentación del dispositivo

El vcon-pulse-link es un dispositivo electrónico diseñado para actuar como unidad de control en sistemas que emplean drivers industriales para motores. Su objetivo es sustituir la necesidad de un controlador lógico programable (PLC), integrando en un único diseño todas las funciones necesarias para comunicar, gobernar y supervisar el comportamiento del driver y del motor conectado.

Gracias a su arquitectura compacta, robusta y orientada a entornos industriales, el dispositivo permite operar directamente sobre los parámetros que determinan el movimiento del motor.

A través de su interfaz de comunicación, el vcon-pulse-link es capaz de generar todas las señales de mando necesarias, garantizando una conexión estable y compatible con cualquier driver basado en un bus de control estándar.

Ilustración 1. Visualización del vcon-pulse-link

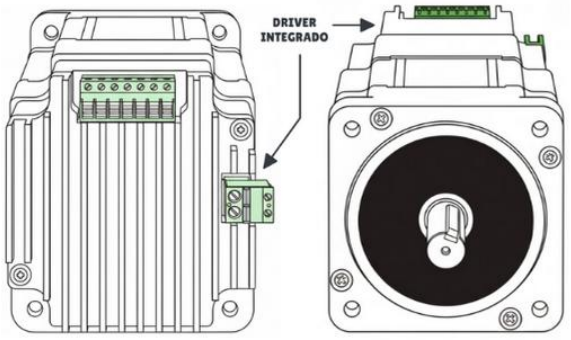


1.2. Objetivo funcional del dispositivo

El diseño del pulse-link responde a los requisitos planteados por la empresa colaboradora del proyecto, cuyo objetivo es disponer de una unidad electrónica capaz de actuar como enlace directo entre las condiciones de la instalación industrial y el driver encargado de accionar el motor eléctrico.

Este dispositivo constituye la primera fase de un proyecto más amplio, en el que se prevé integrar el propio driver dentro de la misma unidad. Su función actual es generar de forma autónoma y aislada las señales de control necesarias, sirviendo como base para la futura evolución del sistema hacia una solución totalmente integrada de control y potencia

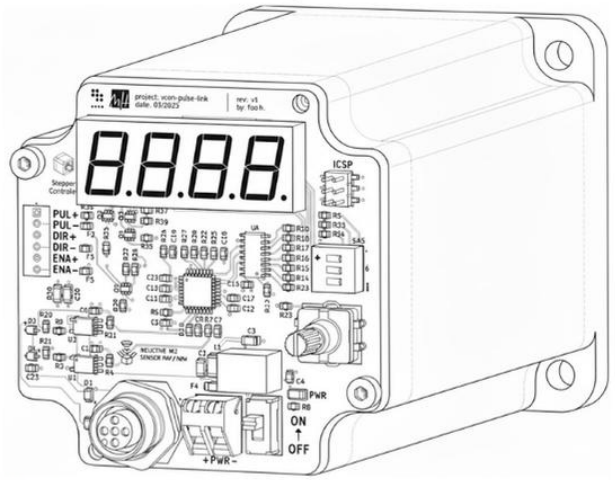
Ilustración 2. Objetivo funcional driver integrado



1.3. Montaje físico, integración en la instalación

vcon-pulse-link puede comunicarse con cualquier driver industrial que utilice un bus de control compatible. No obstante, por indicaciones de uso del dispositivo, su diseño mecánico ha sido optimizado para permitir su fijación directa en la parte trasera de motores paso a paso tipo **NEMA 34.** Por ello, su uso puede quedar restringido para drivers compatibles con ese motor.

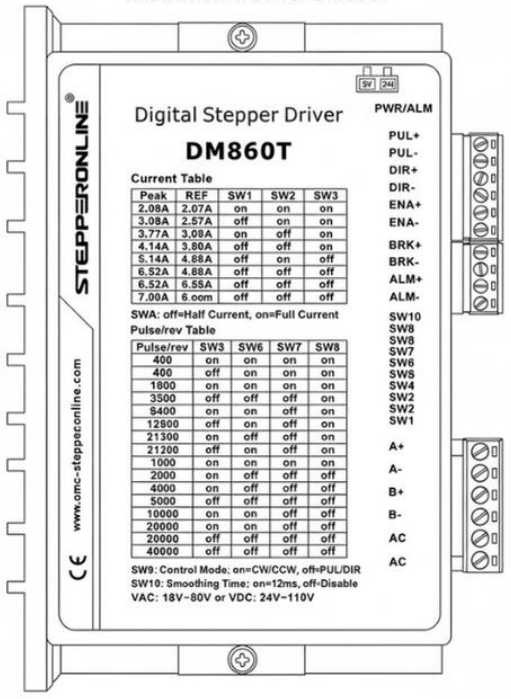
Ilustración 3. Montaje físico en el NEMA 34



1.4. Driver empleado en el desarrollo

Para el desarrollo del dispositivo se ha empleado como referencia el driver **DM860T**, un controlador industrial ampliamente utilizado para motores de alta precisión. Su elección permite establecer un entorno de pruebas representativo y compatible con la mayoría de sistemas basados en motores **NEMA** de gran tamaño.

Ilustración 4. Driver DM860T



Enlace de compra del driver DM860T
https://www.stepperonline.es/index.php?route=product/product/get_file&file=386/DM860T_V3.0.pdf

Enlace del datasheet del driver DM860T
https://www.stepperonline.es/index.php?route=product/product/get_file&file=386/DM860T_V3.0.pdf

2. Seguridad y recomendaciones de uso

2.1. Lectura previa del manual

Antes de realizar cualquier conexión eléctrica o puesta en marcha del sistema, el usuario debe leer y comprender este manual. Una instalación incorrecta o una interpretación errónea del sistema puede activar inesperadamente el motor, comprometiendo la seguridad del operario y de la instalación.

vcon-pulse-link es un equipo de control que actúa sobre el driver del motor, por lo que es altamente recomendable leer y entender el manual de uso del fabricante del driver conectado.

⚠ Advertencia

Un mal conexionado o una configuración no prevista puede provocar el arranque repentino del motor. Por favor, asegúrese de comprender todo el funcionamiento del sistema antes de aplicar tensión.

2.2. Comprobaciones previas a la activación

Antes de alimentar el dispositivo, se recomienda al usuario seguir el checklist que se detalla a continuación. Estas comprobaciones reducen el riesgo de dañar el dispositivo, por lo que es altamente recomendable asegurarse de su cumplimiento previamente.

- ☐ El dispositivo está conectado a una fuente de alimentación compatible, con polaridad y valor de tensión correcto.
- ☐ Driver industrial está bien configurado y dispone de un bus compatible con la interfaz del vcon-pulse-link.
- ☐ Verificar que el sensor inductivo es compatible con el tipo de entrada del dispositivo y no presenta daños visibles.
- ☐ Verificar que el motor está firmemente sujeto y no podrá desplazarse bajo ningún concepto al arrancar.
- ☐ Asegúrese que el vcon-pulse-link tiene los parámetros de configuración que el usuario desea.
- ☐ Verificar la seguridad del entorno, verificar que no hay ni cables sueltos ni piezas que puedan interferir con el motor.

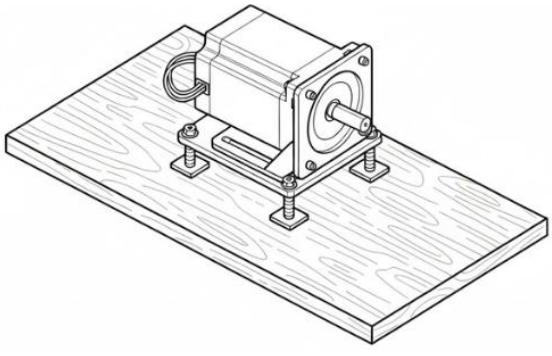
2.3. Montaje y anclaje seguro del dispositivo

La seguridad del sistema electrónico depende directamente de la estabilidad con la que esté fijado el motor al que va acoplado. Antes de anclar este dispositivo, es imprescindible asegurar que el motor está correctamente sujetado. Recuerde que cualquier vibración por no fijar correctamente el motor puede inducirse al vcon-pulse-link provocando daños irreversibles, por lo que este paso es de los más importantes de la instalación.

Situación 1 - Pruebas con el motor sin carga

Ante pruebas en vacío (realizadas en un banco de trabajo) se debe fijar el motor mediante un soporte válido para motores NEMA o una brida de fijación adecuada. Estos soportes están diseñados para mantener el motor inmóvil y evitar cualquier cualquier tipo de desplazamiento, vibración o giro inesperado.

Ilustración 5. Motor correctamente acoplado



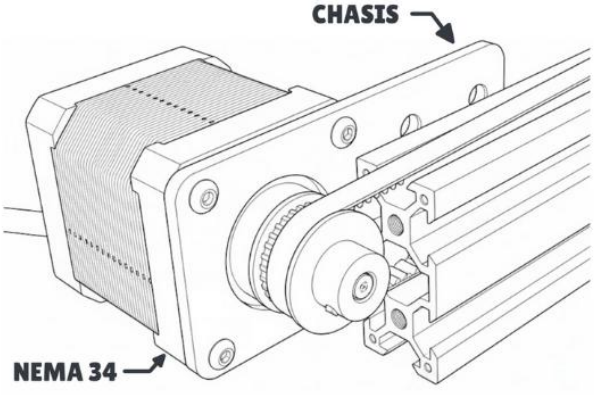
Situación 2 - Montaje en un sistema industrial

En una instalación real, el motor debe fijarse mediante soporte de tipo mecánico, garantizando que la estructura inferior sea capaz de soportar el par, vibraciones y cargas derivadas del uso.

⚠ Importante

Debe asegurarse de que el motor queda rígido y sin holguras, evitando los esfuerzos mecánicos sobre los propios cables o el dispositivo.

Ilustración 6. Motor correctamente acoplado

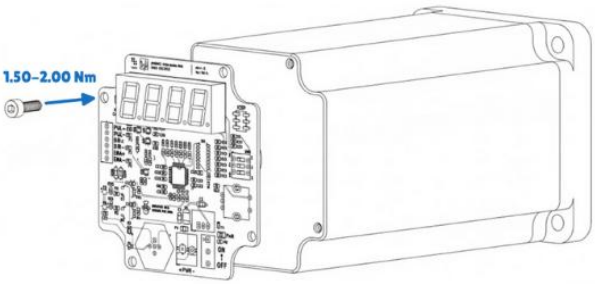


Una vez el motor esté correctamente anclado en cualquiera de los dos escenarios, el vcon-pulse-link puede fijarse posteriormente, utilizando los puntos de montaje incluidos en el diseño. Para realizar el montaje, el kit del dispositivo incluye los componentes:

Unidades	Elemento	Observaciones
2 ud.	Tornillo DIN 912 M4x12	Tornillo de cabeza cilíndrica hexagonal
2 ud.	Arandela DIN 125-A M4	Arandela estándar para distribuir carga
1 ud.	Herramienta Allen 3mm	Llave para tornillos de métrica DIN 912

Como referencia general, el par de apriete recomendado para los tornillos DIN 912 M4x12 utilizados en la fijación del vcon-pulse-link a la parte trasera del motor se sitúa en torno al **1.5-2.0 N·m**

Ilustración 7. Par de apriete necesario para los anclajes



2.4. Seguridad eléctrica - Normativa aplicable

La norma UNE EN 60204-1 establece los requisitos para que el equipamiento eléctrico de una máquina industrial sea seguro, fiable y adecuado para su uso por parte del operario.

Esta normativa exige un protocolo a seguir por el producto ante una desconexión eléctrica, el vcon-pulse-link no solo sigue esas indicaciones, sino que también permite configurar el dispositivo para que se produzca un rearme cuando vuelva la alimentación.

Su tensión de funcionamiento se clasifica dentro de los circuitos SELV/PELV (Safety / Protective Extra-Low Voltage) dentro del rango de seguridad.

Ilustración 8. Normativa aplicable al diseño

UNIVERSITAS Miguel Hernández

VCON

ANEXO III

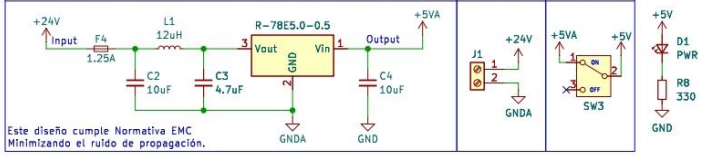
Esquemático del circuito diseñado con KiCad

vcon-pulse-link

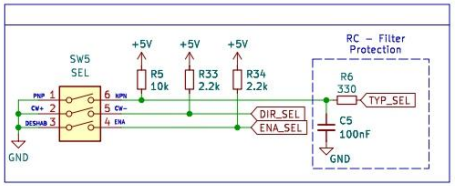
Interfaz Electrónica de Control Industrial

Anexos Justificativos | Versión 1.0

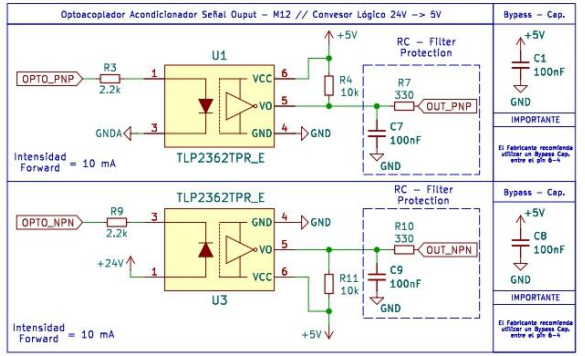
Power Supply Switching Regulator



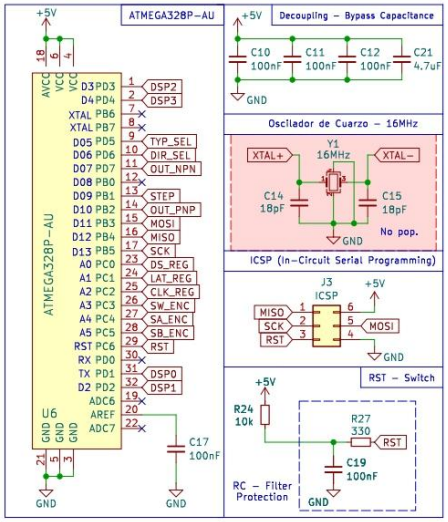
Operation Selectors Switch Mechanical Inputs



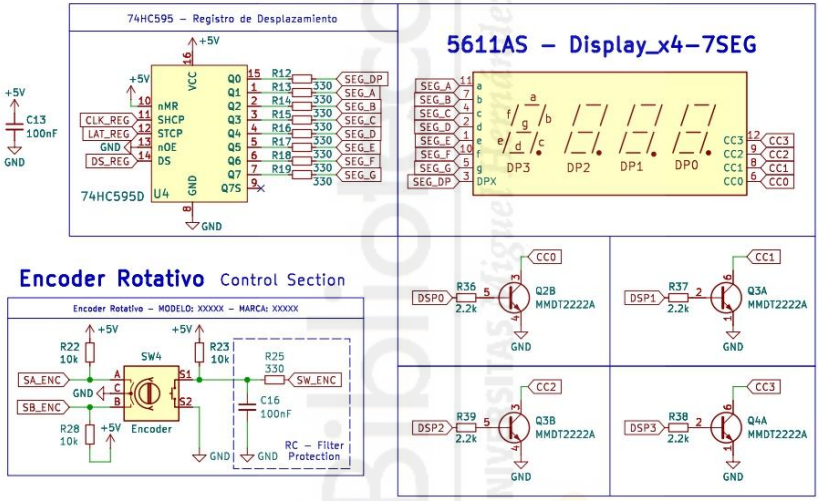
Optocoupler Conversor Logic Convert



MicroController Atmega328P-AU

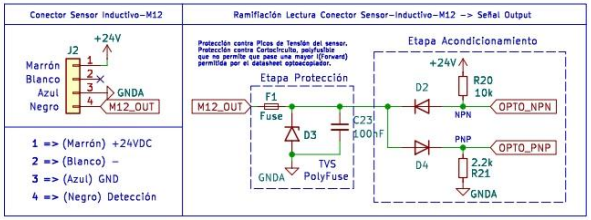


ShiftRegister (Registro) CONVERTSIÓN (SERIE - PARALELO)

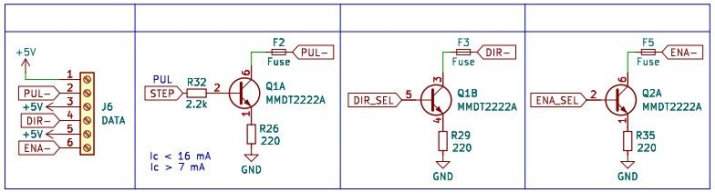


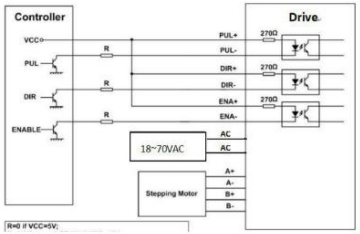
Encoder Rotativo Control Section

Inductive Sensor Interface PNP/NPN Selection



Buffer Output Colector Abierto - Interfaz de Salida





UNIVERSITAS Miguel Hernández

VCON SERVICE

Sheet: /
File: vcon-pulse-link.kicad_sch

Title:

Size: A3	Date:	Rev:
KiCad E.D.A. 9.0.6		Id: 1/1

UNIVERSITAS Miguel Hernández

VCON

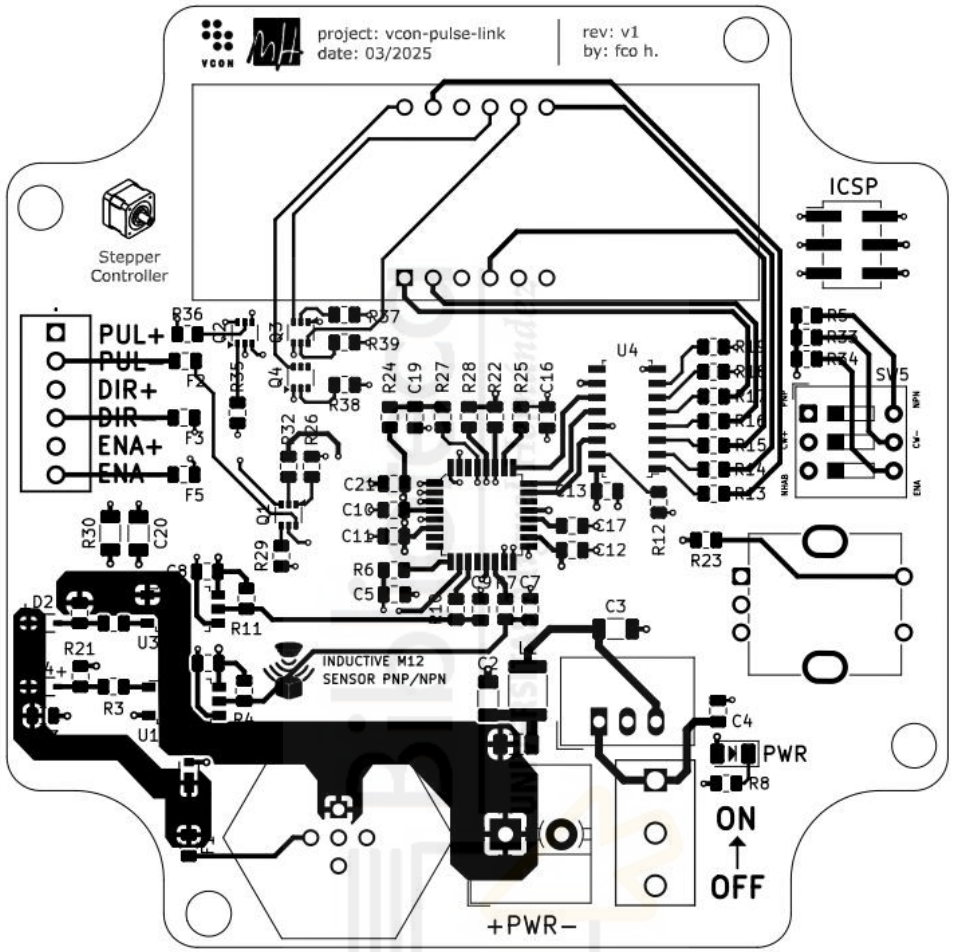
ANEXO IV

Planos del circuito impreso (PCB)

vcon-pulse-link

Interfaz Electrónica de Control Industrial

Anexos Justificativos | Versión 1.0

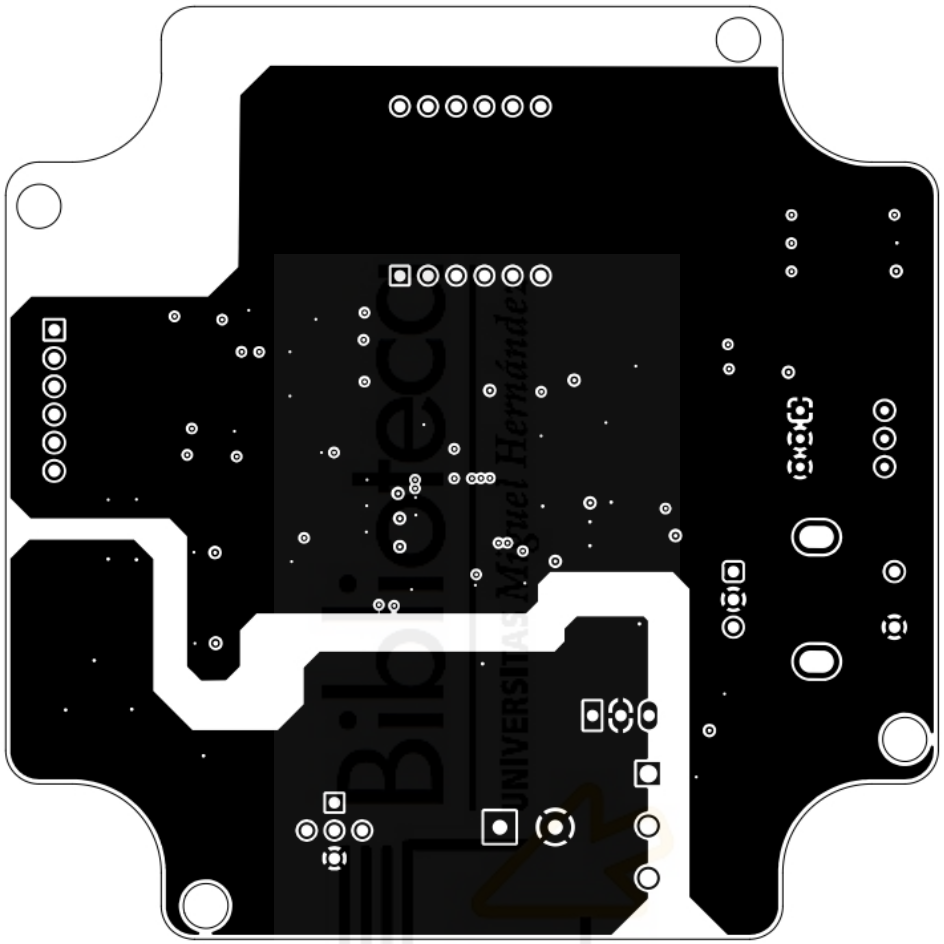


Grado Ingeniería Industrial.
Especialidad electrónica y automática industrial

UNIVERSITAS Miguel Hernández

VCON SERVICE

Sheet:		
File: vcon−pulse−link.kicad_pcb		
Title: vcon−pulse−link		
Size: A5	Date: 2025−05−20	**Rev: Fran Hidalgo**
KiCad E.D.A. 9.0.6		Id: 1/1

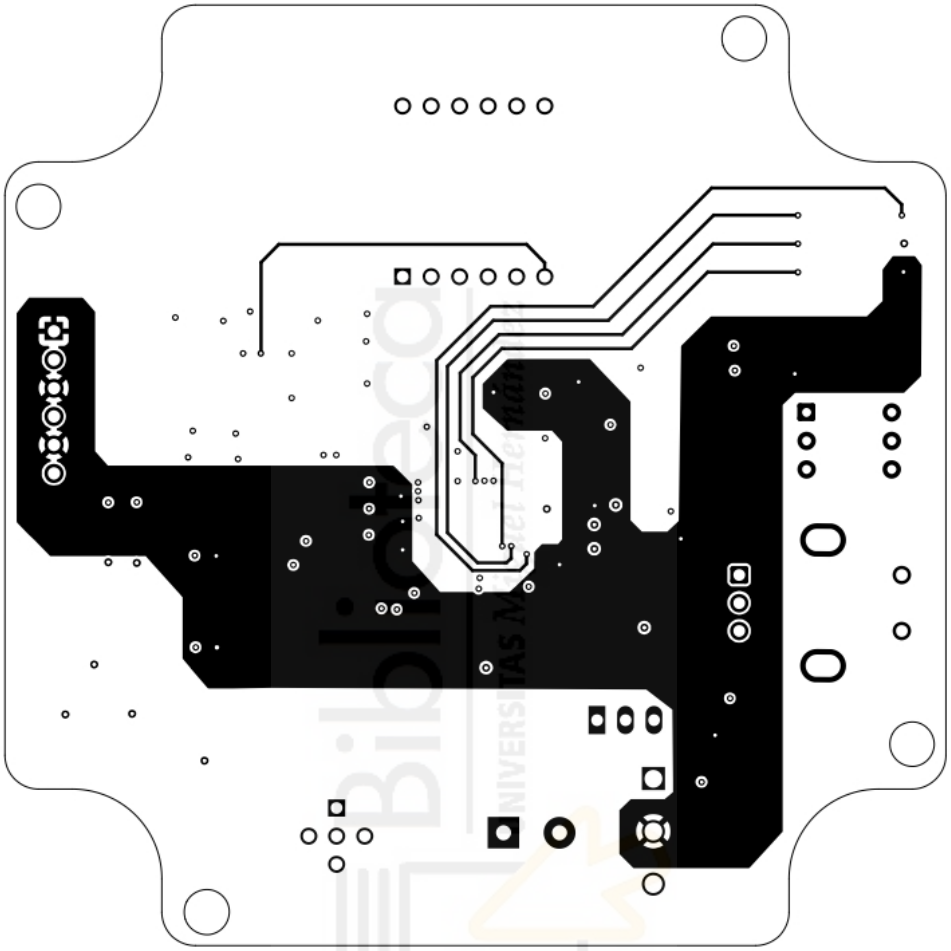


Grado Ingeniería Industrial.
Especialidad electrónica y automática industrial

UNIVERSITAS Miguel Hernández

VCON SERVICE

Sheet:		
File: vcon-pulse-link.kicad_pcb		
Title: vcon-pulse-link		
Size: A5	Date: 2025-05-20	**Rev: Fran Hidalgo**
KiCad E.D.A. 9.0.6		Id: 1/1

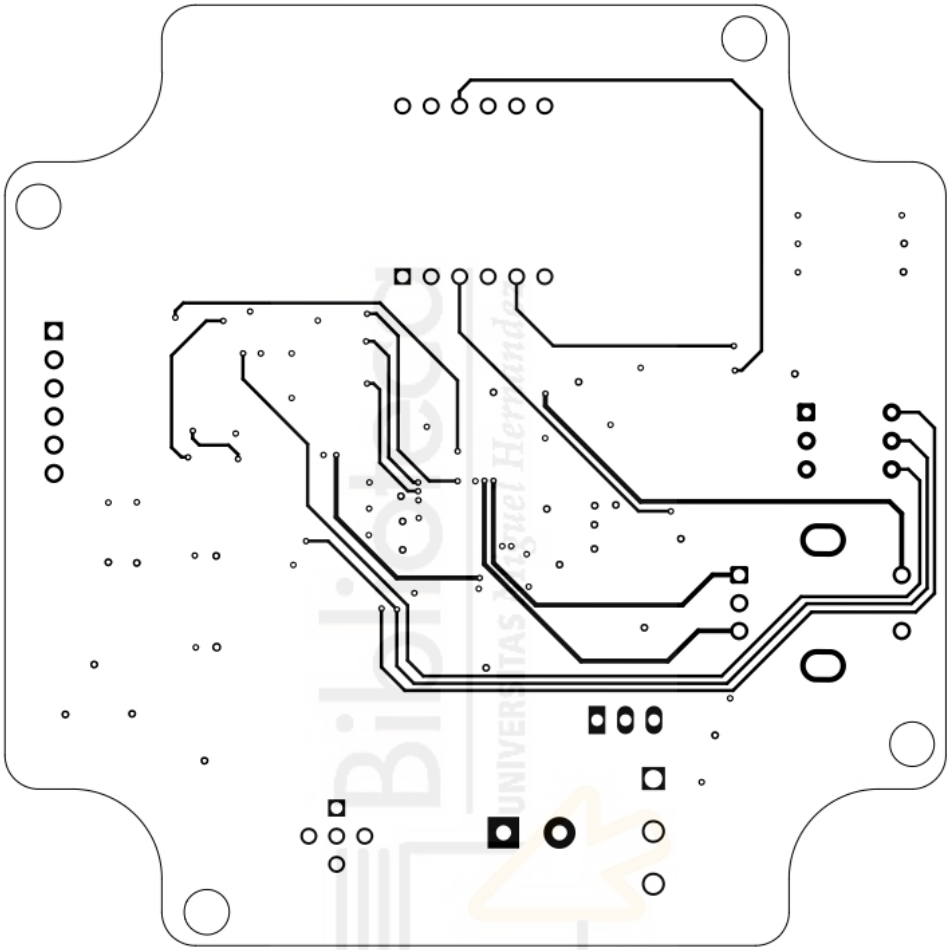


Grado Ingeniería Industrial.
Especialidad electrónica y automática industrial

UNIVERSITAS Miguel Hernández

VCON SERVICE

Sheet:		
File: vcon-pulse-link.kicad_pcb		
Title: vcon-pulse-link		
Size: A5	Date: 2025-05-20	**Rev: Fran Hidalgo**
KiCad E.D.A. 9.0.6		Id: 1/1



Grado Ingeniería Industrial.
Especialidad electrónica y automática industrial

UNIVERSITAS Miguel Hernández

VCON SERVICE

Sheet: File: vcon-pulse-link.kicad_pcb		
Title: vcon-pulse-link		
Size: A5	Date: 2025-05-20	**Rev: Fran Hidalgo**
KiCad E.D.A. 9.0.6		Id: 1/1

UNIVERSITAS Miguel Hernández

VCON

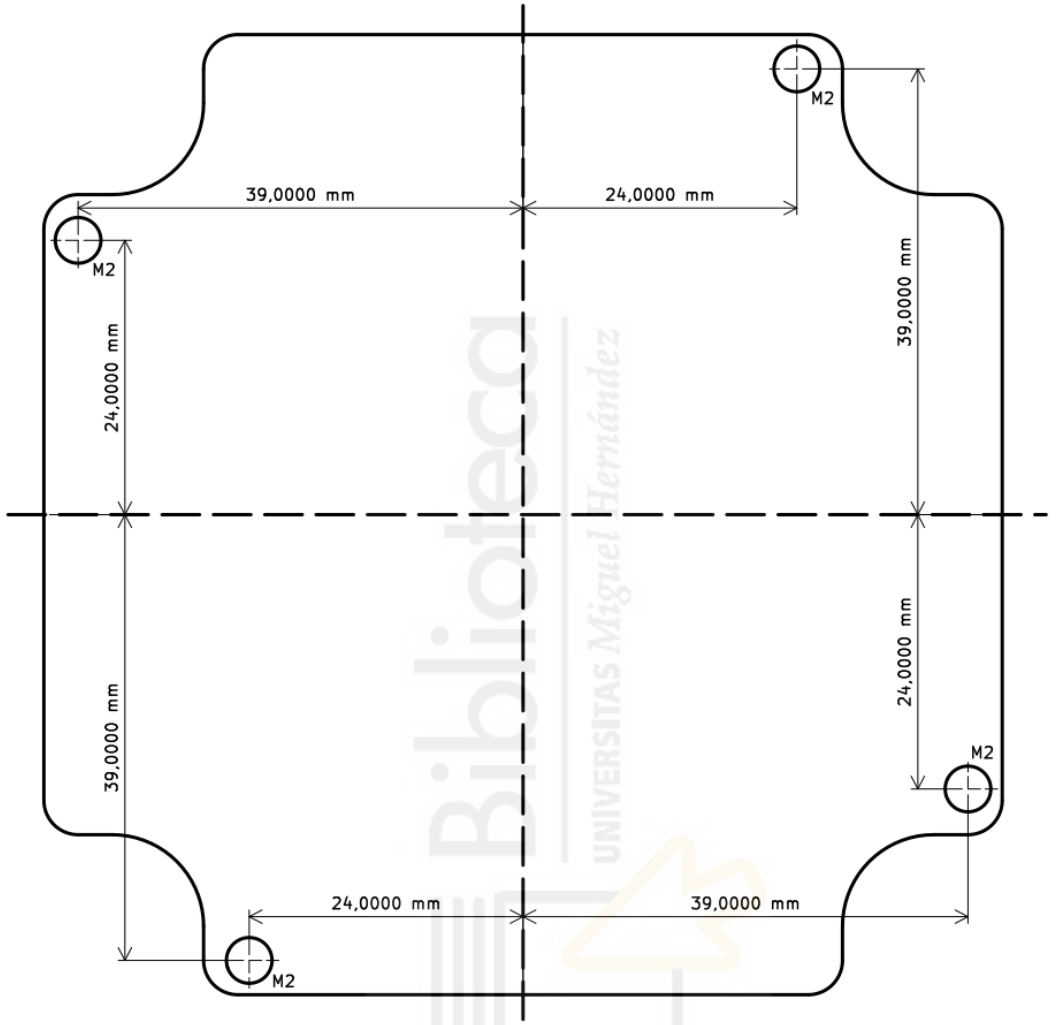
ANEXO V

Planos de acotación del circuito impreso (PCB)

vcon-pulse-link

Interfaz Electrónica de Control Industrial

Anexos Justificativos | Versión 1.0

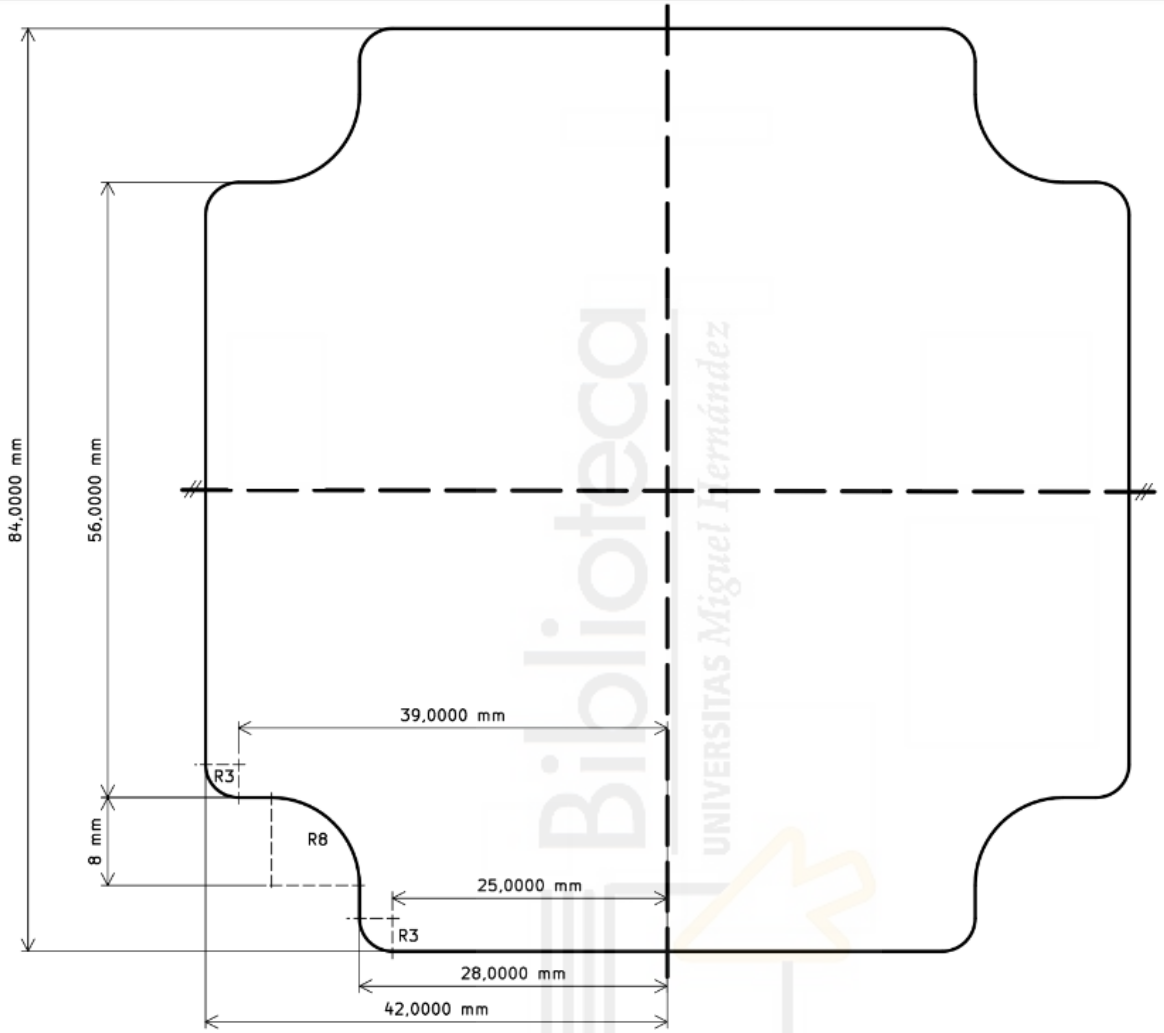


Grado Ingeniería Industrial.
Especialidad electrónica y automática industrial

UNIVERSITAS Miguel Hernández

VCON SERVICE

Sheet: File: vcon-pulse-link.kicad_pcb		
Title:		
Size: A5	Date:	**Rev:**
KiCad E.D.A. 9.0.6		Id: 1/1



Grado Ingeniería Industrial.
Especialidad electrónica y automática industrial

UNIVERSITAS Miguel Hernández

VCON SERVICE

Sheet: File: vcon-pulse-link.kicad_pcb		
Title:		
Size: A5	Date:	**Rev:**
KiCad E.D.A. 9.0.6		Id: 1/1

UNIVERSITAS Miguel Hernández

VCON

ANEXO VI

Código fuente del firmware

vcon-pulse-link

Interfaz Electrónica de Control Industrial

Anexos Justificativos | Versión 1.0

vcon-pulse-link.ino

```
/*
 *c Proyecto: PulseLINK
 *
 * Descripción: Firmware (proyecto educativo en fase experimental) para generar
señales
 * de control STEP/DIR de frecuencia variable mediante encoder rotativo. Se
visualizará
 * la frecuencia de generación (en %) en un display de 7 segmentos.
 *
 * Autor: Fran Hidalgo
 * Fecha de desarrollo: 12/02/2025
 * Última modificación: -
 *
 * Todos los derechos reservados al autor del proyecto © 2025. Este proyecto y
su código
 * se utilizarán exclusivamente con fines académicos.
 *
 *   >> Firmware diseñado exclusivamente para la placa PulseLINK del proyecto <<
 *   Vea el archivo README.md para más información sobre como cargar el
firmware...
 */

// LIBS
#include <Arduino.h>
#include <EEPROM.h>
#include "Encoder.h"
#include "SerialR.h"
#include "Display.h"
#include "FreqGen.h"
#include "EEPSave.h"

// CONSTANTS - ENCODER
#define SW_ENC PC3    // A3
#define SA_ENC PC4    // A4
#define SB_ENC PC5    // A5

// CONSTANTS - REGISTRO
#define  DS_REG PC0  // A0
#define LAT_REG PC1  // A1
#define CLK_REG PC2  // A2

// CONSTANTS - DISPLAY
#define DISP3 PD4   // D4
#define DISP2 PD3   // D3
#define DISP1 PD2   // D2
#define DISP0 PD1   // D1

// CONSTANTS - FREQGEN
#define OUT_PNP PB2   // D10

// CONSTANT - EEPSAVE
#define DS_EEP PD5 // D5

// OBJECTS
Encoder rot(&DDRC, &PINC, &PORTC, SA_ENC, SB_ENC, SW_ENC);
SerialR reg(&DDRC, &PINC, &PORTC, CLK_REG, LAT_REG, DS_REG);
Display dis(&DDRD, &PIND, &PORTD, DISP3, DISP2, DISP1, DISP0, reg);
FreqGen gen(&DDRB, &PINB, &PORTB, OUT_PNP);
EEPSave eep(&DDRD, &PIND, &PORTD, DS_EEP);

// GLOBAL CONTROL VARIABLES
bool sleep_mode;
```

```
unsigned long prevMillis = 0;
unsigned long  interval = 1.5;

// SETUP
void setup() {
    // INIT OBJ
    rot.begin();
    reg.begin();
    dis.begin();
    gen.begin();

    // LOAD MEMORY DATA
    uint8_t delta = EEPROM.read(0);
    sleep_mode = EEPROM.read(1);

    // COMPROBAR DATOS A INTRODUCIR EN EEPROM
    if (delta < 0 && delta > 100) delta = 50;
    if (sleep_mode != 0 && sleep_mode != 1) sleep_mode = 1;

    // STORE DATA DELTA
    dis.storeDelta(delta);
    gen.setTargetDelta(delta);

    // LOAD DATA SLEEP MODE
    gen.setSleepMode(sleep_mode);
    dis.storeMode(gen.plotState());
}

void loop() {

    unsigned long nowMillis = millis();

    // GUARDAR DATOS EN EEPROM
    eep.update(rot.plotDelta(), sleep_mode);

    // LECTURA DEL SENSOR INDUCTIVO CON FSM
    if (gen.tick()) dis.storeMode(gen.plotState());

    // LECTURA DEL ENCODER ROTATIVO
    if (rot.update()) {
        dis.storeDelta(rot.plotDelta());
        gen.setTargetDelta(rot.plotDelta());
    }

    // PULSADOR PARA EL SLEEP MODE
    if (rot.plotSW()) {
        sleep_mode = !sleep_mode;
        gen.setSleepMode(sleep_mode);
    }

    // EJECUTAR TIMER DEL DISPLAY
    if (nowMillis - prevMillis >= interval) {

        // AJUSTE DEL TIMER
        prevMillis = nowMillis;

        // CODE INTERUPT...
        dis.update();
        if(gen.needUpload()){
            gen.uploadFreq();
        }
    }
}
```

Display.h

```
/*
 * Proyecto: PulseLink
 * Class File: Display.h
 *
 * Bibliografía:
 * Fecha de Desarrollo: 20/02/2025
 * Descripción: Esta clase permite controlar un Display 7 segmentos
multiplexado, para su
 * correcto funcionamiento es necesario utilizar un registro de desplazamiento
74HC595.
 *   >> CONEXIONES ENTRE 74HC595 y el Display (5641AS):
 *        Q0 => DP (Punto_Decimal)
 *        Q1 => A  (Segmento_A)
 *        Q2 => B  (Segmento_B)
 *        Q3 => C  (Segmento_C)
 *        Q4 => D  (Segmento_D)
 *        Q5 => E  (Segmento_E)
 *        Q6 => F  (Segmento_F)
 *        Q7 => G  (Segmento_G)
 *
 * Anotaciones de Diseño:
 *    Cuando creamos un objeto "Display", debemos pasarle como parámetro para el
constructor
 *    otro objeto asociado al registro de desplazamiento. La clase "Display"
internamente se
 *    encargará  de hacer llamadas a funciones del objeto "SerialR",
simplificando el código.
 * Ejemplo de Funcionamiento:
 *    Esta clase envía una lista de bits asignados como constantes en esta misma
clase, pero
 *    para que funcione es necesario hacer las conexiones físicas como aparece
en la parte
 *    superior del esquema. Por ejemplo, si queremos enviar el número 9:
 *
 *      - CTE "Display.h" => {}
 *      - SALIDA DEL REG. => {Q7, Q6, Q5, Q4, Q3, Q2, Q1, Q0}
 *      - CONEXIONES DISPLAY {DP, A,  B,  C,  D,  E,  F,  G}
 *
 *     Recordar que el registro de desplazamiento realizará el envío de bits de
forma secuencial
 *     primero el más significativo (Q7) y luego el menos significativo (Q0).
 *
 * Display Información:
 *    El display (5641AS) tiene 4 displays de 7 segmentos, por tanto el primer
display está
 *    asociado al modo de funcionamiento, colocando una letra equivalente a:
 *      - d:  Default (No se genera STEP/DIR)
 *      - C:  ClockWise (Sentido de giro horario)
 *      - _C: CounterClockWise (Sentido de giro antihorario)
 *
 *    Los otros tres números se encargan de mostrar el porcentaje de generación.
Es decir,
 *    un número de 0-100.
 * Importante:
 *    El multiplexado del display se realiza por medio de una función de la
clase. Por tanto,
 *    desde el main debemos ejecutar esa función para que se produzca ese
efecto. La frec. (F)
 *    se establece desde el main... Recomendando crear un Timer cada 100Hz,
asegurando el buen
 *    funcionamiento del multiplexado.
 */
```

```
#ifndef DISPLAY_H
#define DISPLAY_H

#include <Arduino.h>
#include "SerialR.h"

// CONSTANTES DEL DISPLAY 7 SEG
const bool _SEG[13][8] = {
//  { G, F, E, D, C, B, A, DP}   // DISPLAY_DATA
    { 0, 1, 1, 1, 1, 1, 1, 0 },  // 0
    { 0, 0, 0, 0, 1, 1, 0, 0 },  // 1
    { 1, 0, 1, 1, 0, 1, 1, 0 },  // 2
    { 1, 0, 0, 1, 1, 1, 1, 0 },  // 3
    { 1, 1, 0, 0, 1, 1, 0, 0 },  // 4
    { 1, 1, 0, 1, 1, 0, 1, 0 },  // 5
    { 1, 1, 1, 1, 1, 0, 1, 0 },  // 6
    { 0, 0, 0, 0, 1, 1, 1, 0 },  // 7
    { 1, 1, 1, 1, 1, 1, 1, 0 },  // 8
    { 1, 1, 0, 1, 1, 1, 1, 0 },  // 9
    { 1, 0, 0, 0, 0, 0, 0, 1 },  // STATE 0 - MODE 0 - SLEEP
    { 1, 0, 1, 1, 1, 1, 0, 1 },  // STATE 1 - MODE 1 - WAIT
    { 0, 1, 1, 0, 1, 1, 0, 1 }   // STATE 2 - MODE 2 - RUN
  };

class Display {
    private:
        // ADDRESS PORT
        volatile uint8_t *_DDRx;
        volatile uint8_t *_PINx;
        volatile uint8_t *_PORTx;

        // ADDRESS PINS
        uint8_t _DISP3;   // (MSB)
        uint8_t _DISP2;
        uint8_t _DISP1;
        uint8_t _DISP0;   // (LSB)

        // VECTOR ADDRESS PINS
        uint8_t _DISP[4];

        // OBJECT SERIALR
        SerialR& _SR;

        // PARAMETROS DEL DISPLAY
        bool _dispAct[4] = {1, 1, 1, 1};  // DISPLAY ACTIVO (MULTIPLEXADO)
        uint8_t _index = 3;               // INDEX DEL DISPLAY ACTIVO

        // INIT MEMORY DISPLAY
        bool _dispMem[4][8] = {
          { 1, 1, 1, 1, 1, 1, 1, 1 },  // [0] = UD.  (LSB)
          { 1, 1, 1, 1, 1, 1, 1, 1 },  // [1] - DEC.
          { 1, 1, 1, 1, 1, 1, 1, 1 },  // [2] - CEN.
          { 1, 1, 1, 1, 1, 1, 1, 1 },  // [3] - LET. (MSB)
        };

    public:
        Display(
            // ADDRESS PORT
            volatile uint8_t *DDRx,
            volatile uint8_t *PINx,
            volatile uint8_t *PORTx,
```

```
            // ADDRESS PINS
            uint8_t DISP3,
            uint8_t DISP2,
            uint8_t DISP1,
            uint8_t DISP0,

            // OBJECT SERIALR
            SerialR& SR
        );

        // SETUP FUNCTIONS
        void begin();

        // MAIN FUNCTIONS
        void update();

        // STORE FUNCTIONS
        void storeDelta(uint8_t delta);
        void storeMode(uint8_t mode);
};

#endif // DISPLAY_H
```

Display.cpp

```
#include "Display.h"

// CONSTRUCTOR
Display::Display(
    // ADDRESS PORT
    volatile uint8_t *DDRx,
    volatile uint8_t *PINx,
    volatile uint8_t *PORTx,

    // ADDRESS PINS
    uint8_t DISP3,
    uint8_t DISP2,
    uint8_t DISP1,
    uint8_t DISP0,

    // OBJECT SERIALR
    SerialR& SR
) : _SR(SR) {
    // ASIGNATION PORT
    _DDRx = DDRx;
    _PINx = PINx;
    _PORTx = PORTx;

    // ASIGNATION PINS
    _DISP3 = DISP3;
    _DISP2 = DISP2;
    _DISP1 = DISP1;
    _DISP0 = DISP0;

    // VECTOR ADDRESS PINS
    _DISP[0] = _DISP0;  // (LSB)
    _DISP[1] = _DISP1;
    _DISP[2] = _DISP2;
    _DISP[3] = _DISP3;  // (MSB)
}

// SETUP DEL DISPLAY (PORTS)
void Display::begin() {
    // SETUP PINOUT & APAGAR DISPLAY
    for(uint8_t d = 0; d < 4; d++) {
        *_DDRx  |= (1 << _DISP[d]);  // OUTPUT...
        *_PORTx |= (1 << _DISP[d]); // OFF
    }
    // SECUENCIA DE INICIALIZACIÓN...
    _SR.begin();
}

// MAIN FUNCTION - UPDATE DISPLAY
void Display::update() {
    // Evitar efectos de sombras - 'Ghosting'
    *_PORTx &= ~(1 << _DISP[_index]);   // OFF = '0' (Apaga el display actual)
    _index = (_index + 1) % 4;          // NEXT DISPLAY

    // Multiplexado del display
    _SR.sendVect(_dispMem[_index]);
    *_PORTx |= (1 << _DISP[_index]);    // ON = '1' (Activa el nuevo display)

    // Ajuste de Conmutación
    delayMicroseconds(50);
}
```

```
void Display::storeDelta(uint8_t delta) {
    uint8_t centena = delta / 100;
    uint8_t decena  = (delta / 10) % 10;
    uint8_t unidad  = delta % 10;

    for (uint8_t i = 0; i < 8; i++) {
        _dispMem[2][i] = _SEG[centena][i];
        _dispMem[1][i] = _SEG[decena][i];
        _dispMem[0][i] = _SEG[unidad][i];
    }
}

void Display::storeMode(uint8_t mode) {

    // SETUP DEL MODO DE FUNCIONAMIENTO
    for (uint8_t i = 0; i < 8; i++) {
        _dispMem[3][i] = _SEG[mode+10][i];
    }
}
```

EEPSave.h

```
#ifndef EEPSAVE_H
#define EEPSAVE_H

#include <Arduino.h>
#include <EEPROM.h>

class EEPSave {
    private:
        // ADDRESS PORT
        volatile uint8_t *_DDRx;
        volatile uint8_t *_PINx;
        volatile uint8_t *_PORTx;

        // ADDRESS PINS
        uint8_t _DS;

        // FLANCO EVENTO
        bool _dsEvent = false;

        // FILTRO DIGITAL DE FLANCO
        uint64_t _dsHistory = 0;
        const uint64_t MASK_LOW = 0x0000000000000000;
        const uint64_t MASK_HIGH = 0xFFFFFFFFFFFFFFFF;

        // FSM (Finite State Machine)
        enum FSM { HIGH_STATE, LOW_STATE };
        FSM _dsFSM = LOW_STATE;

    public:
        EEPSave(
            // ADDRESS PORT
            volatile uint8_t *DDRx,
            volatile uint8_t *PINx,
            volatile uint8_t *PORTx,

            // ADDRESS PINS
            uint8_t pinDS
        );

        // SETUP FUNCTIONS
        void begin();

        // MAIN FUNCTIONS
        void update(uint8_t delta, bool sleep_mode);
};

#endif // ENCODER_H
```

EEPSave.cpp

```cpp
#include "EEPSave.h"

// CONSTRUCTOR
EEPSave::EEPSave(
    volatile uint8_t *DDRx,
    volatile uint8_t *PINx,
    volatile uint8_t *PORTx,
    uint8_t pinDS
) {
    _DDRx = DDRx;
    _PINx = PINx;
    _PORTx = PORTx;

    _DS = pinDS;
}

// SETUP DE PINES
void EEPSave::begin() {
    *_DDRx &= ~(1 << _DS);  // Entrada
    *_PORTx |= (1 << _DS);  // Pull-up activado
}

// ACTUALIZACION DE ESTADO
void EEPSave::update(uint8_t delta, bool sleep_mode) {
    uint8_t bitLSB = ((*_PINx) & (1UL << _DS)) ? 1 : 0;
    _dsHistory = (_dsHistory << 1) | bitLSB;

    bool lowStable = (_dsHistory & 0xFFFFFFFFFFFFFFFF) == MASK_LOW;
    bool highStable = (_dsHistory & 0xFFFFFFFFFFFFFFFF) == MASK_HIGH;

    switch (_dsFSM) {
        case HIGH_STATE:
            if (lowStable) {
                _dsFSM = LOW_STATE;
                _dsEvent = true;
            }
            break;

        case LOW_STATE:
            if (highStable) {
                _dsFSM = HIGH_STATE;
            }
            break;
    }

    if (_dsEvent) {
        _dsEvent = false;

        // ALMACENAR DATOS DE DELTA
        if (delta >= 0 && delta <= 100) {
            EEPROM.update(0, delta);
        }

        // ALMACENAR DATOS DE SLEEP MODE
        EEPROM.update(1, sleep_mode ? 1 : 0);
    }
}
```

Encoder.h

```
/*
 * Proyecto: PulseLink
 * Class File: Encoder.h
 *
 *  Bibliografía: https://forum.arduino.cc/t/reading-rotary-encoders-as-a-state-
machine/937388
 *  Fecha de Desarrollo: 20/01/2025
 *
 *  Descripción: Especificando pines de conexión del Encoder rotativo, podemos
crear objetos con
 *  funcionalidades específicas de lectura del periférico. Se utiliza una
FSM/MEALY que procesa
 *  los pulsos de los canales A y B del encoder para determinar la dirección de
giro.
 *
 *    >>> Una vez determinado el giro, se incrementa/decrementa una parámetro
que varía [0, 100],
 *        que simula un contador ajustable por el usuario mediante el movimiento
del Encoder. <<<
 *
 *  Nomenclatura: Es posible que en algunos Encoder cambie, frecuentemente se
utiliza_
 *      - CLK : Canal A (SA) del Encoder
 *      - DT  : Canal B (SB) del Encoder
 *      - SW  : Pulsador (opcional)
 *
 *  Metodología:
 *      -> void reset()     // Reinicia la FSM => state = 0;
 *      -> bool update()    // Verifica si hay cambios (delta / SW) => Devuelve
True si hay cambios
 *
 *      -> bool plotSW()      // Devuelve true si en el último update se pulsó
el SW.
 *      -> uint8_t plotDelta()  // Devuelve el valor de delta => tipo uint8_t
 *
 *      >>> plotSW() y plotDelta() deben usarse en el loop después de update <<<
 *
 *  Anotaciones de Diseño:
 *      El encoder se diseñó con resistencias pull-up (+5V), lo que supone que
ante un giro
 *      habrá un periodo de tiempo en el que los pines SA y SB estarán a nivel
bajo, con un
 *      desfase de 90° entre ellos, para determinar el giro.
 *
 *      => Pronunciese "<<" como "sucede antes de", es decir, "SA << (sucede
antes de) SB"...
 *      Posibles combinaciones:
 *          - Giro Sentido Horario (CW): SA << SB.
 *          - Giro Sentido Antihorario (CCW): SB << SA.
 *
 */

#ifndef ENCODER_H
#define ENCODER_H

#include <Arduino.h>
#include <EEPROM.h>
```

```
class Encoder {
    private:
        // ADDRESS PORT
        volatile uint8_t *_DDRx;
        volatile uint8_t *_PINx;
        volatile uint8_t *_PORTx;

        // ADDRESS PINS
        uint8_t _SA;
        uint8_t _SB;
        uint8_t _SW;

        // PARAMETROS DEL ENCODER
        uint8_t _delta;             // SE MUEVE DESDE [0, 100]
        bool _lastSWState = false;  // VALOR SW TRAS EL ÚLTIMO UPDATE

        // PARAMETROS DE LA FSM
        uint8_t _state = 0;

        // FLANCO LECTOR SW
        uint64_t _swHistory = 0;
        bool _swEvent = false;
        enum ButtonFSM { IDLE, WAIT_STABLE, PRESSED };
        ButtonFSM _swStateFSM = IDLE;

    public:
        Encoder(
            // ADDRESS PORT
            volatile uint8_t *DDRx,
            volatile uint8_t *PINx,
            volatile uint8_t *PORTx,

            // ADDRESS PINS
            uint8_t pinSA,
            uint8_t pinSB,
            uint8_t pinSW
        );

        // SETUP FUNCTIONS
        void begin();

        // MAIN FUNCTIONS
        void reset();
        bool update();

        // LECTURE FUNCTIONS
        bool plotSW();
        uint8_t plotDelta();
};

#endif // ENCODER_H
```

Encoder.cpp

```cpp
#include "Encoder.h"

// CONSTRUCTOR
Encoder::Encoder(
    volatile uint8_t *DDRx,
    volatile uint8_t *PINx,
    volatile uint8_t *PORTx,
    uint8_t pinSA,
    uint8_t pinSB,
    uint8_t pinSW
) {
    _DDRx = DDRx;
    _PINx = PINx;
    _PORTx = PORTx;

    _SA = pinSA;
    _SB = pinSB;
    _SW = pinSW;
}

// SETUP DE PINES
void Encoder::begin() {
    // CONFIGURACIÓN DE PUERTOS
    *_DDRx &= ~((1 << _SA) | (1 << _SB) | (1 << _SW));  // Entradas
    *_PORTx |= (1 << _SA) | (1 << _SB) | (1 << _SW);    // Pull-ups activados

    // CARGAR DE LA MEMORIA EL PRIMER VALOR DE DELTA
    const uint8_t _DS = EEPROM.read(0);
    if(_DS >= 0 && _DS <= 100) {
        _delta = _DS;  // CARGAR DELTA DESDE EEPROM
    } else {
        _delta = 0;    // INICIALIZAR A 0
    }
}

void Encoder::reset() {
    _delta = 0;
}

// ACTUALIZACIÓN DEL ESTADO DEL ENCODER Y SWITCH
bool Encoder::update() {
    // --- LECTURA ENCODER (A y B) ---
    bool readSA = (*_PINx & (1 << _SA)) == 0;
    bool readSB = (*_PINx & (1 << _SB)) == 0;

    switch (_state) {
        case 0:
            if (!readSA) _state = 1;
            else if (!readSB) _state = 4;
            break;
        case 1:
            if (!readSB) _state = 2;
            else if (readSA) _state = 0;
            break;
        case 2:
            if (readSA) _state = 3;
            else if (readSB) _state = 0;
            break;
        case 3:
            if (readSA && readSB) {
                _state = 0;
                if (_delta > 0) {
```

```
                    _delta--;
                    return true;
                }
            }
            break;
        case 4:
            if (!readSA) _state = 5;
            else if (readSB) _state = 0;
            break;
        case 5:
            if (readSB) _state = 6;
            else if (readSA) _state = 0;
            break;
        case 6:
            if (readSA && readSB) {
                _state = 0;
                if (_delta < 100) {
                    _delta++;
                    return true;
                }
            }
            break;
    }

    // ----- ACTUALIZACIÓN HISTORIAL SW -----
    bool readSW = (*_PINx & (1 << _SW)) == 0;
    _swHistory = (_swHistory << 1) | readSW;

    // DEFINIMOS LOS UMBRALES DE FILTRO
    const uint64_t PULSADO_MASK = 0x3FFFFFFFFFFF;      // últimos 50 bits = 1
    const uint64_t SUELTO_MASK  = 0x0000000000000000;  // últimos 40 bits = 0

    bool swEstablePulsado = ((_swHistory & PULSADO_MASK) == PULSADO_MASK);
    bool swEstableSuelto  = ((_swHistory & PULSADO_MASK) == SUELTO_MASK);

    // FSM del botón
    switch (_swStateFSM) {
        case IDLE:
            if (swEstablePulsado) {
                _swStateFSM = WAIT_STABLE;  // Transición inicial
            }
            break;

        case WAIT_STABLE:
            if (swEstablePulsado) {
                _swStateFSM = PRESSED;
                _swEvent = true;            // FLANCO DETECTADO
                return true;
            } else if (swEstableSuelto) {
                _swStateFSM = IDLE;         // Volvemos atrás
            }
            break;

        case PRESSED:
            if (swEstableSuelto) {
                _swStateFSM = IDLE;         // Preparado para próxima pulsación
            }
            break;
    }
        return false;
}
```

```
// FLANCO DISPONIBLE PARA CONSUMIR
bool Encoder::plotSW() {
    if (_swEvent) {
        _swEvent = false;
        return true;
    }
    return false;
}

uint8_t Encoder::plotDelta() {
    return _delta;
}
```

FreqGen.h

```
/*
 * Proyecto: PulseLiNK
 * Archivo: FreqGen.h
 *
 * Descripción: Clase encargada de generar señales STEP de frecuencia variable
 * controladas mediante encoder rotativo. Implementa una máquina de estados:
 *
 *   - SLEEP: Usuario ha puesto el dispositivo en modo reposo:
 *            (sleep_mode = true, sleep_mode = false).
 *   - WAIT: Esperando que el sensor no detecte objeto:
 *           (OUT_PNP = 0 = DETECTADO, OUT_PNP = 1 = NO DETECTADO).
 *   - RUN: Generación activa, ajustando progresivamente la frecuencia
 *          (OCR1A = 39 (50%), OCR1A = 0 (0%)).
 *
 * El valor de frecuencia se determina con una tabla LUT basada en `delta`.
 *
 * Autor: Fran Hidalgo
 * Fecha: 25/05/2025
 */

#ifndef FREQGEN_H
#define FREQGEN_H

#include <Arduino.h>

class FreqGen {

    public:
        // FSM STATES
        enum class State : uint8_t {
            SLEEP = 0, WAIT = 1, RUN = 2
        };

        // CONSTRUCTOR
        FreqGen(
            // ADDRESS PORT
            volatile uint8_t *DDRx,
            volatile uint8_t *PINx,
            volatile uint8_t *PORTx,

            // ADDRESS PINS
            uint8_t pinOUT_PNP
        );

        // SETUP
        void begin();

        // FSM PNP
        bool tick();

        // INTERFAZ DE CONTROL
        void setTargetDelta(uint8_t delta);
        void setSleepMode(bool sleep_mode);

        // INTERFAZ DE CONSULTA
        uint8_t plotState();
        bool needUpload();
        void uploadFreq();

    private:
        // ADDRESS PORT
        volatile uint8_t *_DDRx;
```

```
        volatile uint8_t *_PINx;
        volatile uint8_t *_PORTx;

        // ADDRESS PINS
        uint8_t _OUT_PNP;

        //ESTADO ACTUAL
        State _state = State::SLEEP;

        // CONTROL DE FRECUENCIA
        uint8_t _delta_target = 0;
        uint8_t _delta_current = 0;
        bool _need_upload = false;

        // TIMER --------------------------
        unsigned long _last_step_time = 0;
        const uint16_t _step_delay_ms = 10;
        // --------------------------------

        // MODO SLEEP
        bool _sleep_mode = true;

        // PRIVATE TIMER
        void setupTimer();
        void startTimer();
        void stopTimer();

        // LUT-TABLE DE FRECUENCIAS PARA OCR1A
        const uint16_t FreqGen::LUT[101] = {
          2000,  2000, 1000,  667,  500,  400,  333,  286,  250,  222,
           200,   182,  167,  154,  143,  133,  125,  118,  111,  105,
           100,    95,   91,   87,   83,   80,   77,   74,   71,   69,
            67,    65,   63,   61,   59,   57,   56,   54,   53,   51,
            50,    49,   48,   47,   45,   44,   43,   43,   42,   41,
            40,    39,   38,   38,   37,   36,   36,   35,   34,   34,
            33,    33,   32,   32,   31,   31,   30,   30,   29,   29,
            29,    28,   28,   27,   27,   27,   26,   26,   26,   25,
            25,    25,   24,   24,   24,   24,   23,   23,   23,   22,
            22,    22,   22,   22,   21,   21,   21,   21,   20,   20,
            20
        };

};

#endif // FREQGEN_H
```

FreqGen.cpp

```cpp
#include "Arduino.h"
#include "FreqGen.h"

// CONSTRUCTOR
FreqGen::FreqGen(
    // ADDRESS PORT
    volatile uint8_t *DDRx,
    volatile uint8_t *PINx,
    volatile uint8_t *PORTx,

    // ADDRESS PINS
    uint8_t pinOUT_PNP
) {
    // ASIGNATION PORT
    _DDRx = DDRx;
    _PINx = PINx;
    _PORTx = PORTx;

    // ASIGNATION PINS
    _OUT_PNP = pinOUT_PNP;
}

// SETUP DEL FREQGEN (PORTS)
void FreqGen::begin() {

    // OUT_PNP como entrada con pull-up
    *_DDRx &= ~(1 << _OUT_PNP);
    *_PORTx |= (1 << _OUT_PNP);

    // OC1A (PB1) como salida
    DDRB |= (1 << PB1);

    // Setup Timer1 y dejarlo apagado
    setupTimer();
    stopTimer();
}

// PRIVATE TIMER
void FreqGen::setupTimer() {
    TCCR1A = 0;
    TCCR1B = 0;

    TCCR1A |= (1 << COM1A0);  // Toggle OC1A on compare match
    TCCR1B |= (1 << WGM12);   // CTC mode
    TCCR1B |= (1 << CS10);    // No prescaler (prescaler = 1)

    OCR1A = 39;               // SEÑAL DE 200 kHz (50%)
    TCNT1 = 0;                // Inicializa el contador en 0
}

// PRIVATE TIMER
void FreqGen::startTimer() {
    TCCR1A |= (1 << COM1A0);     // Habilita toggle en OC1A
    TCCR1B |= (1 << CS10);       // Asegura prescaler 1 (activar Timer)
}

// PRIVATE TIMER
void FreqGen::stopTimer() {
    TCCR1A &= ~(1 << COM1A0);     // Desactiva toggle
    TCCR1B &= ~(1 << CS10);       // Detiene el Timer
    PORTB &= ~(1 << PB1);         // Fuerza pin OC1A a LOW
}
```

```cpp
// FSM PNP
bool FreqGen::tick() {
    State prev = _state;

    // LECTURA DEL SENSOR INDUCTIVO (1 = NO DETECTA OBJETO, 0 = DETECTA OBJETO)
    bool valuePNP = (*_PINx & (1 << _OUT_PNP)) ? 1 : 0;

    // FSM LOGIC
    switch (_state) {
        case State::SLEEP:
            if (!_sleep_mode) {
                _state = valuePNP ? State::RUN : State::WAIT;
                if(valuePNP) {
                    _delta_current = 0;
                    _need_upload = true;
                    uploadFreq();
                }
            }
        break;

        case State::WAIT:
            if (_sleep_mode) {
                _state = State::SLEEP;
                stopTimer();
            } else if (valuePNP) {
                _state = State::RUN;
                _delta_current = 0;
                _need_upload = true;
                uploadFreq();
            }
        break;

        case State::RUN:
            if (_sleep_mode) {
                _state = State::SLEEP;
                stopTimer();
            } else if (!valuePNP) {
                _state = State::WAIT;
                stopTimer();
            } else if (_delta_target == 0 && _delta_current == 0) {
                stopTimer();
            } else {
                if(millis() - _last_step_time >= _step_delay_ms) {
                    if (_delta_current < _delta_target) {
                        _delta_current++;
                        _need_upload = true;
                        uploadFreq();
                    } else if (_delta_current > _delta_target) {
                        _delta_current--;
                        _need_upload = true;
                        uploadFreq();
                    }
                    _last_step_time = millis();
                }
            }
            break;
    }

    // Activar timer si acabamos de entrar a RUN
    if (_state == State::RUN && prev != State::RUN) {
        if (_delta_target > 0 || _delta_current > 0) startTimer();
    }

    return (_state != prev);
```

```
}

// INTERFAZ DE CONTROL
void FreqGen::setTargetDelta(uint8_t delta) {
    _delta_target = constrain(delta, 0, 100);
}

// INTERFAZ DE CONTROL
void FreqGen::setSleepMode(bool sleep_mode) {
    _sleep_mode = sleep_mode;
}

// INTERFAZ DE CONSULTA
uint8_t FreqGen::plotState() {
    return static_cast<uint8_t>(_state);
}

// INTERFAZ DE CONSULTA
bool FreqGen::needUpload() {
    return _need_upload;
}

// INTERFAZ DE CONSULTA
void FreqGen::uploadFreq() {
    if (_delta_current > 0) startTimer();

    if (LUT[_delta_current] == 0) OCR1A = 2000;
    else OCR1A = LUT[_delta_current];    // ACTUALIZAR OCR1A
    TCNT1 = 0;       // CONTADOR A 0

    // EVENTO MARCADO!
    _need_upload = false;
}
```

SerialR.h

```
/*
 * Proyecto: PulseLink
 * Class File: SerialR.h
 */

#ifndef SERIALR_H
#define SERIALR_H

#include <Arduino.h>

class SerialR {
    private:
        // ADDRESS PORT
        volatile uint8_t *_DDRx;
        volatile uint8_t *_PINx;
        volatile uint8_t *_PORTx;

        // ADDRESS PINS
        uint8_t _CLK;
        uint8_t _LAT;
        uint8_t _DS;

        // FUNCIONES PRIVADAS INTERNAS
        void _clockUp()    { *_PORTx |= (1 << _CLK); }
        void _latchUp()    { *_PORTx |= (1 << _LAT); }
        void _clockLow()   { *_PORTx &=~(1 << _CLK); }
        void _latchLow()   { *_PORTx &=~(1 << _LAT); }

    public:
        SerialR(
            // ADDRESS PORT
            volatile uint8_t *DDRx,
            volatile uint8_t *PINx,
            volatile uint8_t *PORTx,

            // ADDRESS PINS
            uint8_t pinCLK,
            uint8_t pinLAT,
            uint8_t pinDS
        );

        // SETUP FUNCTIONS
        void begin();

        // MAIN FUNCTIONS
        void sendVect(const bool array[]);
        void sendByte(const byte dataByte);
};

#endif // SERIALR_H
```

SerialR.cpp

```cpp
#include "SerialR.h"

// CONSTRUCTOR
SerialR::SerialR(
    // ADDRESS PORT
    volatile uint8_t *DDRx,
    volatile uint8_t *PINx,
    volatile uint8_t *PORTx,

    // ADDRESS PINS
    uint8_t pinCLK,
    uint8_t pinLAT,
    uint8_t pinDS
) {
    // ASIGNATION PORT
    _DDRx = DDRx;
    _PINx = PINx;
    _PORTx = PORTx;

    // ASIGNATION PINS
    _CLK = pinCLK;
    _LAT = pinLAT;
    _DS = pinDS;
}

// SETUP DEL SERIAL (PORTS)
void SerialR::begin() {
    *_DDRx |= (1 << _CLK) | (1 << _LAT) | (1 << _DS); // Configurar como salida
    *_PORTx &= ~((1 << _CLK) | (1 << _LAT) | (1 << _DS)); // Inicializar en bajo
    _clockLow();
    _latchLow();
    // Limpiar Registro
    sendByte(0x00);
}

// MAIN FUNCTION - SEND ARRAY
void SerialR::sendVect(const bool array[]) {
    for(uint8_t bit = 0; bit < 8; bit++) {
        _clockLow();    // Init Transmision
        if(array[bit]) *_PORTx |= (1 << _DS);   // Enviar '1'
        else           *_PORTx &=~(1 << _DS);   // Enviar '0'
        _clockUp();     // End Transmision
    }
    _latchUp();
    _latchLow();
}

// MAIN FUNCTION - SEND BYTE
void SerialR::sendByte(const byte dataByte) {
    bool BitVector[8] = { };

    // CONVERSIÓN DE BYTE A VECTOR...
    for(uint8_t bit = 0; bit < 8; bit++) {
      BitVector[7-bit] = (dataByte >> bit) & 0x01;
    }
    sendVect(BitVector);
}
```