Diego Arcos Sapena

DISEÑO Y DESARROLLO DE UN CONTROLADOR LÓGICO PROGRAMABLE MODULAR: UNA SOLUCIÓN IOT PARA LA INDUSTRIA 4.0

Grado Ingeniería Informática

Trabajo de Fin de Grado

Dirigido por
Dr. Carlos Molina Clemente

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2025

Resumen.

Este Trabajo de Fin de Grado tiene como objetivo el diseño y desarrollo de un controlador lógico programable (PLC) modular, orientado a los estándares de la Industria 4.0. El proyecto surge de la necesidad de crear soluciones tecnológicas que sean altamente adaptables, escalables y capaces de integrarse en entornos industriales modernos, donde la eficiencia, la interoperabilidad y la conectividad son esenciales.

La metodología empleada incluye el diseño e implementación de una arquitectura modular y escalable, aprovechando tecnologías modernas como Docker para el despliegue de servicios, MariaDB para la gestión centralizada de datos en tiempo real y protocolos de comunicación como MQTT y Modbus TCP/IP.

Finalmente, los resultados obtenidos demuestran que el sistema cumple con todos los requisitos funcionales y no funcionales definidos, logrando el desarrollo de un prototipo con un funcionamiento robusto y fiable.

Resum.

Aquest Treball de Fi de Grau té com a objectiu el disseny i el desenvolupament d'un controlador lògic programable (PLC) modular, orientat als estàndards de la Indústria 4.0. El projecte sorgeix de la necessitat de crear solucions tecnològiques que siguin altament adaptables, escalables i capaces d'integrar-se a entorns industrials moderns, on l'eficiència, la interoperabilitat i la connectivitat són essencials.

La metodologia emprada inclou el disseny i la implementació d'una arquitectura modular i escalable, aprofitant tecnologies modernes com Docker per al desplegament de serveis, MariaDB per a la gestió centralitzada de dades en temps real i protocols de comunicació com MQTT i Modbus TCP/IP.

Finalment, els resultats obtinguts demostren que el sistema compleix tots els requisits funcionals i no funcionals definits, aconseguint el desenvolupament d'un prototip amb un funcionament robust i fiable.

Abstract.

This Final Degree Project aims at the design and development of a modular programmable logic controller (PLC), oriented to Industry 4.0 standards. The project arises from the need to create technological solutions that are highly adaptable, scalable and capable of being integrated into modern industrial environments, where efficiency, interoperability and connectivity are essential.

The methodology used includes the design and implementation of a modular and scalable architecture, taking advantage of modern technologies such as Docker for the deployment of services, MariaDB for centralized management of real-time data and communication protocols such as MQTT and Modbus TCP/IP.

Finally, the results obtained demonstrate that the system meets all the defined functional and non-functional requirements, achieving the development of a prototype with robust and reliable operation.

Agradecimientos.

Quiero comenzar expresando mi gratitud, en primer lugar, a mi mejor amigo: yo mismo. A mi resiliencia, que me ha permitido mantenerme firme en los momentos más desafiantes. A mi ambición, esa fuerza que me impulsa a soñar en grande y a fijarme metas que me desafían constantemente. Y, sobre todo, a mi inconformismo, esa chispa que muchas veces se ha visto como un defecto, pero que hoy reconozco como una de mis mayores virtudes. Este inconformismo domado me permite cuestionar los límites establecidos y perseguir siempre la mejora constante. Hoy agradezco esta fortaleza interior que me guía hacia la excelencia y me acerca, paso a paso, a la mejor versión de mí mismo.

También, quiero agradecer a mi abuelo Antonio, quien me presentó con ilusión a la empresa Borrell. A mis padres y hermano, por brindarme siempre las facilidades, el apoyo y el cariño necesarios para avanzar.

A Vicente y Jose, por abrirme las puertas de la empresa y confiar plenamente en mí desde el principio. A Jorge y Jordi, por su incansable paciencia y por transmitirme cada día su conocimiento y pasión por el trabajo. Y a todos los trabajadores de Borrell, quienes me acogieron y me hicieron sentir parte del equipo desde el primer día.

A mi tutor de TFG, el Dr. Carlos Molina Clemente, por su confianza y orientación impecable a lo largo de este camino.

Y, por último, a la buena música, que me ha acompañado fielmente en todas las horas dedicadas a este proyecto, convirtiéndose en mi aliada silenciosa.

Índice

Índice de tablas

Índice de figuras

1 Introducción

Para comprender en su totalidad el alcance y la repercusión del presente proyecto, es imprescindible contextualizarlo dentro de la evolución histórica de la automatización industrial. Por ello, vamos a introducir este proyecto con una visión amplia del paradigma de la automatización industrial.

1.1 Contexto de la Automatización Industrial

La automatización industrial se refiere a la integración de tecnologías avanzadas, como máquinas, software y sistemas de control, para gestionar y operar procesos industriales con mínima intervención humana. Este enfoque ha revolucionado la producción de bienes y servicios, estableciendo nuevos estándares de eficiencia operativa, precisión y seguridad en los entornos laborales.

1.1.1 Historia de la Automatización Industrial

La evolución de la automatización industrial [1] ha transformado radicalmente la forma en que se producen y gestionan los bienes y servicios, marcando hitos significativos que han impulsado avances tecnológicos y económicos a lo largo del tiempo. Esta evolución puede entenderse a través de las cuatro grandes revoluciones industriales (ver *Figura 1.1*), cada una de las cuales ha representado un cambio de paradigma en los métodos de producción y en el uso de tecnologías.

- **Industria 1.0, Mecanización:** La Primera Revolución Industrial (finales del siglo XVIII) marcó la transición de la producción manual a la mecanización, impulsada por máquinas alimentadas por **energía hidráulica y vapor**. El telar mecanizado de Edmund Cartwright (1785) destacó como un avance clave, optimizando la producción textil y sentando las bases de la industria mecanizada.
- **Industria 2.0, Electrificación:** A finales del siglo XIX, la electricidad revolucionó la industria al superar las limitaciones de las máquinas de vapor, permitiendo una energía más eficiente. En 1913, **Henry Ford** introdujo la línea de ensamblaje flexible, reduciendo costos y tiempos de fabricación, y estableciendo la producción en masa como un estándar. Este período también trajo avances en comunicación, como el teléfono y el telégrafo.
- **Industria 3.0, Digitalización y Automatización:** La Tercera Revolución Industrial (mediados del siglo XX) integró tecnologías digitales en la industria, comenzando con **la primera computadora programable**, la Z3 de Konrad Zuse (1941). En los años 70, los microprocesadores y ordenadores personales impulsaron la automatización avanzada, incluyendo robots industriales y sistemas de control como controladores lógicos programables (PLCs) y SCADA[1], aumentando la precisión y la eficiencia en los procesos.

[1] SCADA (Supervisory Control and Data Acquisition): Sistema de control y adquisición de datos utilizado para supervisar y gestionar procesos industriales en tiempo real.

- **Industria 4.0, Conectividad y Redes Inteligentes:** Actualmente, nos encontramos en la era de la Industria 4.0, un paradigma industrial que combina tecnologías avanzadas como **IoT (Internet of Things)**, **IA**, **computación en la nube** y **sistemas ciberfísicos**. Este estado actual de la industria se caracteriza por la implementación de fábricas inteligentes, donde las **máquinas conectadas** optimizan procesos, ejecutan tareas de manera autónoma y mejoran la eficiencia operativa. Esta etapa permite una producción personalizada, adaptable y altamente eficiente, marcando un punto de inflexión en la transformación digital global.

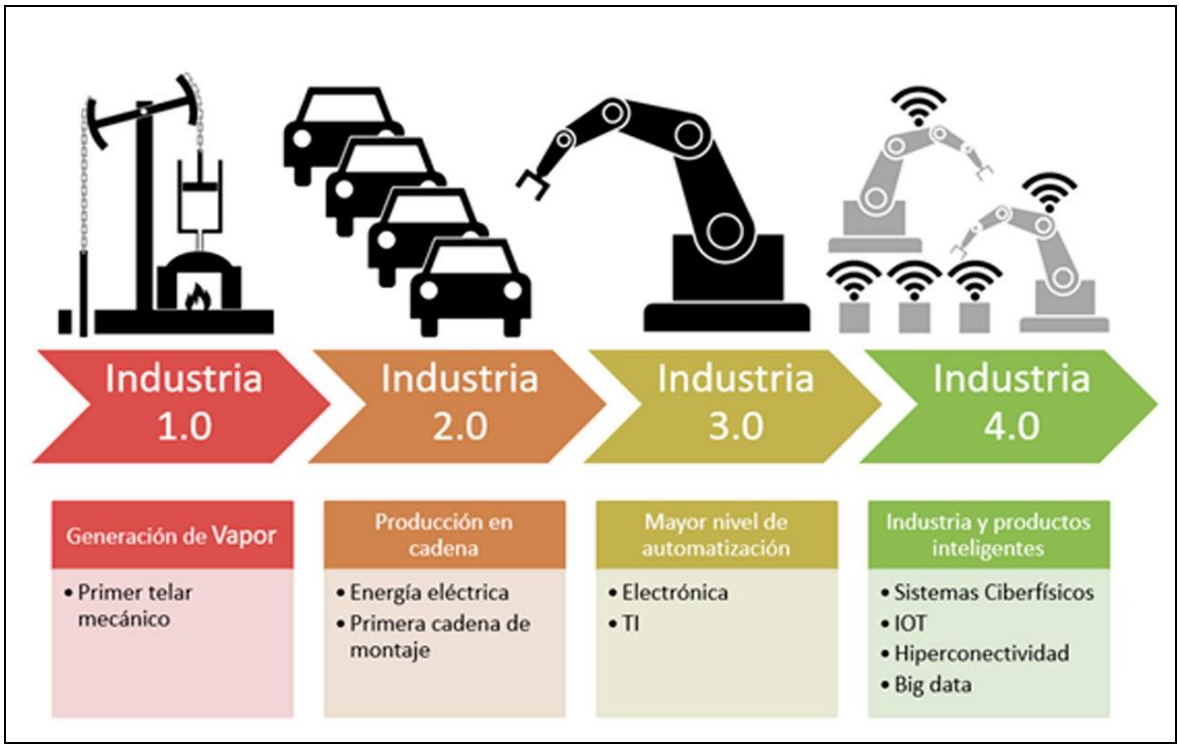

Figura 1.1: Las 4 grandes revoluciones industriales

1.1.2 Industria 4.0: Paradigma Actual

La Industria 4.0 representa un cambio de paradigma en la manufactura y producción industrial, centrado en la **conectividad** y la **digitalización** de los procesos. En este contexto, las fábricas inteligentes utilizan redes de dispositivos interconectados, como sistemas **IoT**, que recopilan, procesan y analizan datos en tiempo real. Esta interconexión permite a las máquinas comunicarse entre sí, tomar decisiones basadas en datos y optimizar procesos de manera autónoma o semiautónoma, mejorando significativamente la eficiencia, la productividad y la capacidad de personalización.

Un componente crucial dentro de la Industria 4.0 es el **Internet de las Cosas Industrial (IIoT)**, que extiende los principios del IoT al entorno industrial. A través de sensores y dispositivos conectados, el IIoT facilita el monitoreo, análisis y control de activos industriales en tiempo real, permitiendo una integración sin precedentes entre los sistemas físicos y digitales. Esto está transformando sectores como la manufactura, la energía y la logística, al proporcionar visibilidad completa de las operaciones y habilitar respuestas rápidas a cambios en el entorno.

La **inteligencia artificial (IA)** también desempeña un papel esencial, posibilitando el análisis de grandes volúmenes de datos (**Big Data**) para identificar patrones y generar insights[2] que optimizan la toma de decisiones. Con la IA, las empresas pueden anticipar fallos en los equipos, implementar mantenimientos predictivos y optimizar sus cadenas de suministro, reduciendo costos y minimizando tiempos de inactividad.

Otro elemento clave es la **computación en la nube**. Esta proporciona la infraestructura necesaria para almacenar y procesar grandes cantidades de datos de forma eficiente, lo cual permite el acceso en **tiempo real** a la información desde **cualquier ubicación**, facilitando la monitorización y el control remoto de las operaciones industriales, así como la integración con servicios externos.

En este marco, la Industria 4.0 fomenta el desarrollo de **sistemas ciberfísicos (CPS)**, otro elemento fundamental, en los que los entornos físicos están estrechamente integrados con modelos digitales. Esta interacción habilita simulaciones y optimizaciones en tiempo real, creando entornos dinámicos y adaptativos. Un ejemplo de ello lo son los **gemelos digitales (digital twins)** [2], réplicas virtuales de activos físicos que permiten monitorear el estado de los equipos, predecir su comportamiento y realizar ajustes antes de aplicar cambios en el sistema real.

En conclusión, la Industria 4.0 no solo está transformando los procesos industriales mediante la integración de tecnologías avanzadas como el IIoT, la inteligencia artificial, la computación en la nube y los sistemas ciberfísicos, sino que también impulsa la sostenibilidad y la competitividad. Ofrece un nivel sin precedentes de optimización, flexibilidad y personalización en la producción, mientras minimiza el desperdicio de recursos y el impacto ambiental. Este paradigma posiciona a las empresas para enfrentar los desafíos globales de sostenibilidad y adaptarse a un mercado cada vez más dinámico y exigente, consolidándose como la base para el futuro de la manufactura.

---

[2] Insights: Se refiere a conocimientos o descubrimientos valiosos derivados del análisis de datos, que permiten identificar patrones, tendencias o información clave para la toma de decisiones.

1.1.3 Protocolos de Comunicación a Bajo Nivel: SPI

En los sistemas embebidos, los protocolos de comunicación a bajo nivel desempeñan un papel esencial para garantizar la interacción eficiente entre los componentes electrónicos. Entre los más utilizados se encuentran **SPI (Serial Peripheral Interface)** e **I²C (Inter-Integrated Circuit)**, ambos diseñados para la comunicación serial entre microcontroladores y periféricos como sensores, memorias y pantallas.

Este apartado se centra especialmente en el **protocolo SPI** [3], dado que es el estándar seleccionado para la comunicación entre el núcleo del PLC y sus tarjetas periféricas.

SPI es un protocolo serial síncrono desarrollado por Motorola a principios de los años 80, especialmente para aplicaciones que requieren velocidades de transferencia altas y baja latencia. Utiliza un enfoque **maestro-esclavo**, empleando líneas dedicadas para datos y señales de control, lo que garantiza una comunicación rápida y eficiente. Debido a su arquitectura simple y alta velocidad, el SPI es ampliamente utilizado en aplicaciones donde se prioriza el rendimiento.

La comunicación se realiza a través de **cuatro líneas principales** (ver *Figura 1.2*):

- **SCLK (Serial Clock):** Genera la señal de reloj para sincronizar la transmisión de datos.
- **MOSI (Master Out, Slave In):** Línea donde el maestro envía datos al esclavo.
- **MISO (Master In, Slave Out):** Línea donde el esclavo envía datos al maestro.
- **SS (Slave Select):** Línea utilizada por el maestro para seleccionar el esclavo con el que desea comunicarse.

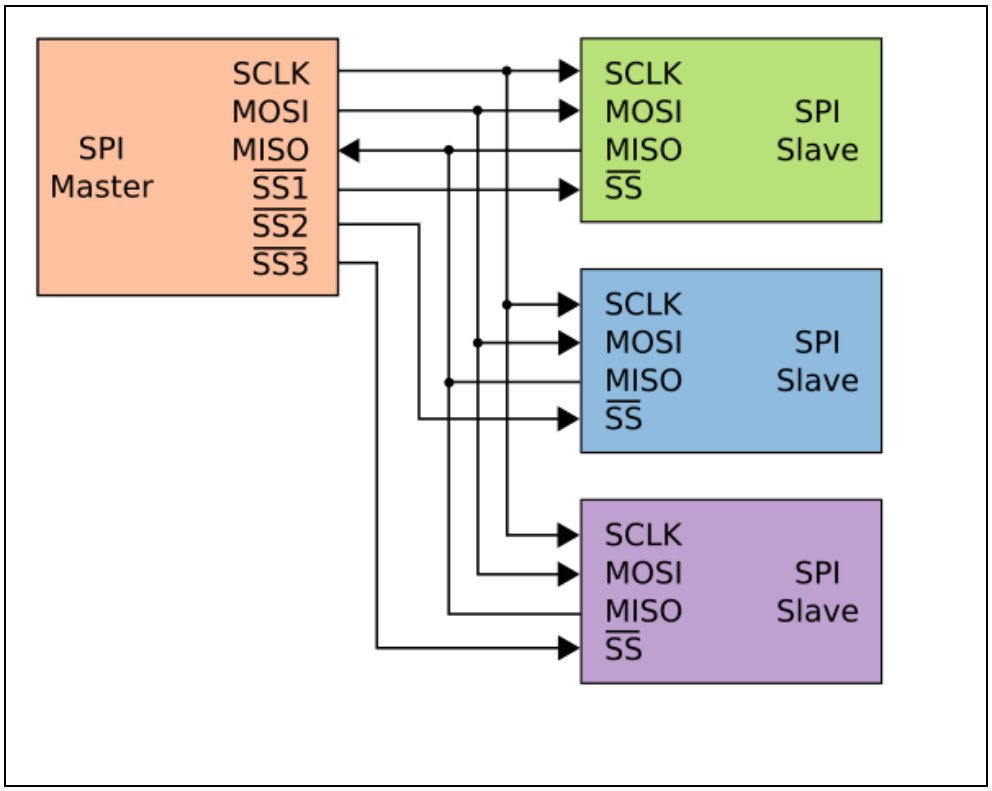

Figura 1.2: Diagrama de bloques del protocolo SPI

Este diseño permite la comunicación simultánea en ambas direcciones, característica conocida como **Full-Duplex**, lo que aumenta significativamente la eficiencia.

En cuanto a la comunicación, esta se realiza de manera **síncrona**, lo que significa que el maestro genera una señal de reloj para sincronizar la transferencia de datos. El proceso básico de comunicación incluye las siguientes etapas:

- **Selección del Esclavo:** El maestro selecciona el esclavo deseado activando su línea SS (poniéndola en bajo). Si hay varios esclavos, cada uno tendrá su propia línea SS.
- **Configuración del Reloj:** El maestro configura la frecuencia del reloj, la polaridad (CPOL) y la fase (CPHA) según los requisitos del esclavo.
- **Transferencia de Datos:** Los datos se transmiten bit a bit a través de las líneas MOSI y MISO, sincronizados con el reloj SCLK. Cada byte enviado desde el maestro se acompaña de un byte recibido desde el esclavo, permitiendo la comunicación Full-Duplex.
- **Finalización de la Transferencia:** Una vez completada la comunicación, el maestro desactiva la línea SS, finalizando la interacción con el esclavo.

El protocolo SPI destaca por una serie de características que lo convierten en una opción preferida en aplicaciones de sistemas embebidos:

- **Alta Velocidad:** El SPI es uno de los protocolos más rápidos para comunicación serial, con velocidades que pueden superar los **50 Mbps**. Es ideal para aplicaciones donde la rapidez es fundamental.
- **Comunicación Full-Duplex:** Permite la transmisión y recepción de datos simultáneamente, maximizando la eficiencia en la comunicación.
- **Topología Maestro-Esclavo:** Un maestro controla la comunicación y puede interactuar con múltiples esclavos mediante líneas SS dedicadas.
- **Flexibilidad:** El protocolo es altamente configurable en términos de velocidad de reloj, polaridad y fase del reloj, adaptándose a una amplia variedad de dispositivos.
- **Simplicidad:** El protocolo es fácil de implementar, lo que lo hace ideal para aplicaciones que requieren una configuración rápida y directa.

Gracias a estas características, el protocolo SPI se utiliza en una amplia variedad de aplicaciones donde la velocidad y la eficiencia son esenciales. Por ejemplo, es común en dispositivos como tarjetas SD, donde facilita la transferencia de datos entre un controlador y la memoria de almacenamiento. También se utiliza en pantallas LCD y OLED para enviar gráficos y comandos de control desde microcontroladores, y en sensores industriales, como acelerómetros y giroscopios.

1.1.4 Protocolos de Comunicación Industriales: Modbus

En el ámbito industrial, los protocolos de comunicación son fundamentales para garantizar la interoperabilidad entre dispositivos y sistemas. Entre los más utilizados se encuentran Modbus [4].

Modbus es un protocolo de comunicación industrial ubicado en la **capa de aplicación** del modelo OSI. Fue desarrollado en 1979 por Modicon (ahora parte de Schneider Electric) y diseñado para facilitar la transmisión de datos entre dispositivos **maestros y esclavos**, como PLCs, sensores y actuadores. Su simplicidad y eficiencia lo convirtieron en un estándar en aplicaciones de automatización y control.

Inicialmente, operando sobre RS-232[3] y RS-485[4], Modbus permitió comunicaciones eficientes en redes industriales. En la década de 1990, la extensión Modbus TCP/IP adaptó el protocolo para redes Ethernet, mejorando la velocidad y fiabilidad en aplicaciones modernas, como el control remoto y distribuido.

Desde 2004, bajo la supervisión de la Modbus Organization, el protocolo ha seguido evolucionando para integrarse en tecnologías actuales como IoT, consolidándose como una herramienta esencial en la automatización industrial. A continuación, se explica detalladamente su funcionamiento.

El protocolo Modbus organiza la comunicación en un modelo maestro-esclavo:

- **Maestros:** El dispositivo que inicia todas las comunicaciones. Normalmente es un PLC, HMI[5] o sistema SCADA.
- **Esclavos:** Dispositivos periféricos que responden a las solicitudes del maestro. Por ejemplo, sensores, actuadores o controladores.

El maestro envía una solicitud a un esclavo específico, que responde con los datos requeridos o confirma la acción ejecutada. Este modelo asegura que no haya colisiones en el bus de comunicación.

En cuanto a los tipos de datos, en Modbus están organizados en cuatro categorías principales, cada una asociada a un área de memoria específica:

- **Coils:**
    - **Descripción:** Representan salidas digitales que se pueden activar o desactivar (encendido/apagado, 1/0).
    - **Acceso:** Lectura y escritura.
    - **Uso típico:** Control de dispositivos binarios como relés, luces o motores.

---

3 RS-232: Protocolo de comunicación serie asíncrona, ideal para cortas distancias, con un solo dispositivo por línea.

4 RS-485: Protocolo de comunicación serie multipunto, diseñado para largas distancias y redes de múltiples dispositivos.

5 HMI (Human-Machine Interface): Interfaz gráfica que permite la interacción entre los usuarios y sistemas automatizados, facilitando el monitoreo y control de procesos industriales.

- **Discrete Inputs:**
    - **Descripción:** Representan entradas digitales que solo se pueden leer.
    - **Acceso:** Solo lectura.
    - **Uso típico:** Monitorización de sensores de estado como interruptores de límite o botones de parada de emergencia.
- **Holding Registers:**
    - **Descripción:** Representan registros de 16 bits que se utilizan para almacenar datos configurables o salidas analógicas.
    - **Acceso:** Lectura y escritura.
    - **Uso típico**: Ajuste de valores como velocidad de un motor, posición de un actuador o configuración de un dispositivo.
- **Input Registers:**
    - **Descripción:** Representan registros de 16 bits que solo se pueden leer, utilizados para datos analógicos de sensores.
    - **Acceso:** Solo lectura.
    - **Uso típico:** Lectura de valores como temperatura, presión o nivel.

A parte de los tipos de datos, Modbus soporta diferentes modos de comunicación adaptados a diversas aplicaciones (ver *Figura 1.3*):

- **RTU (**Remote Terminal Unit**):** Transmite datos en formato binario, compacto y eficiente.
- **TCP/IP:** Extiende Modbus para redes modernas, permitiendo la comunicación sobre Ethernet. **Es la versión de Modbus que se utiliza en este proyecto**.

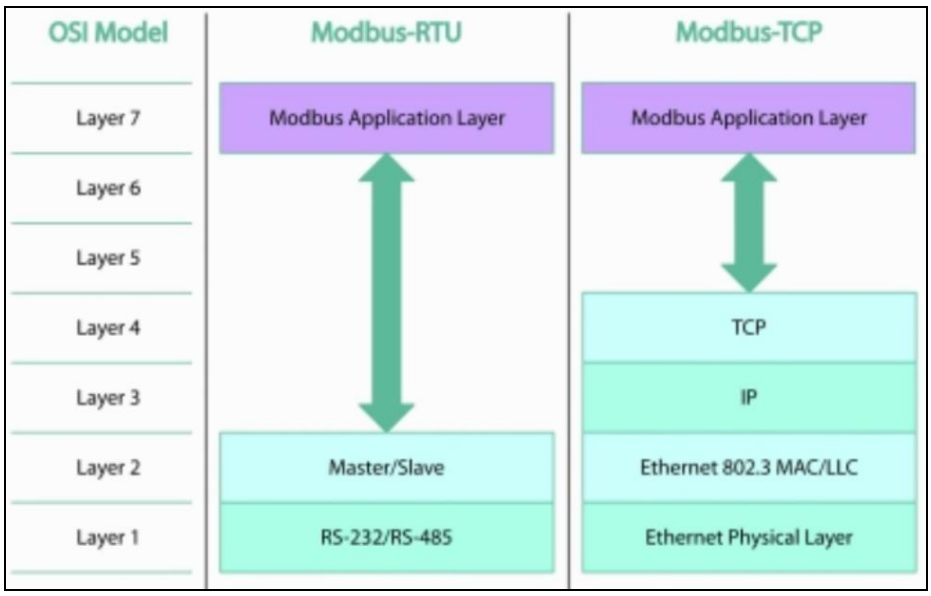

Figura 1.3: Ubicación de los protocolos Modbus TCP/IP y Modbus RTU en el modelo OSI

En resumen, el protocolo Modbus es un pilar en las comunicaciones industriales, utilizado en dispositivos como PLCs, sistemas SCADA, medidores inteligentes, etc. Su capacidad para operar en redes simples o complejas lo hace indispensable en industrias como la manufactura, la energía, la automatización de edificios y el transporte. Su flexibilidad y fiabilidad garantizan una integración efectiva en una amplia variedad de aplicaciones.

1.1.5 Protocolos de Comunicación Industriales: MQTT

MQTT (Message Queuing Telemetry Transport) [5] es un protocolo ligero de mensajería, ubicado en la **capa de aplicación** del modelo OSI, diseñado para transmitir datos en redes con ancho de banda limitado o alta latencia. Desarrollado en 1999 por IBM y Eurotech para monitorizar oleoductos, utiliza un modelo de **publicación/suscripción** que facilita la comunicación eficiente entre dispositivos.

Estandarizado por OASIS en 2013, MQTT se ha convertido en un pilar en IoT, automatización industrial y sistemas embebidos. Sus características avanzadas, como **calidad de servicio** (**QoS**) y persistencia de mensajes, lo hacen ideal para entornos con recursos limitados y redes poco confiables.

MQTT sigue un modelo de comunicación publicación/suscripción organizado en tres roles principales:

- **Publicador:** Envía mensajes a un servidor intermedio (**broker**) etiquetados con un tema o **tópico** (**topic**) específico.
- **Suscriptor:** Se registra en un tema en particular para recibir los mensajes que se publican en él.
- **Broker:** Actúa como intermediario, gestionando la distribución de mensajes entre publicadores y suscriptores.

El funcionamiento básico, ilustrado en la *Figura 1.4* incluye:

- **Conexión:** Los dispositivos cliente establecen una conexión TCP/IP con el broker, que gestiona la interacción entre ellos.
- **Publicación/Suscripción:** Los publicadores envían mensajes etiquetados con un tema, y los suscriptores reciben solo los mensajes relacionados con los temas que les interesan.
- **Calidad de Servicio (QoS):** Define la garantía de entrega de los mensajes en tres niveles:
    - **QoS 0:** Entrega al mejor esfuerzo, sin garantías.
    - **QoS 1:** Garantiza que el mensaje se entregue al menos una vez.
    - **QoS 2:** Garantiza que el mensaje se entregue solo una vez.
- **Persistencia de Mensajes:** Los mensajes pueden almacenarse en el broker para que los nuevos suscriptores reciban el último mensaje publicado en un tema relevante.

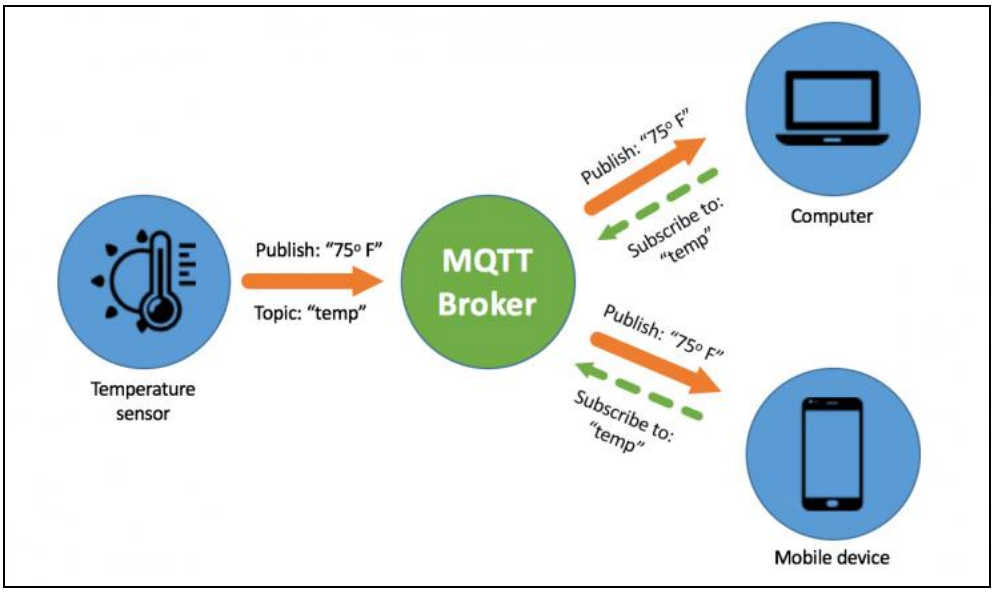

Figura 1.4: Funcionamiento del protocolo MQTT

En resumen, la capacidad de MQTT para operar en entornos con restricciones lo posiciona como una solución esencial en el ecosistema IoT moderno y más allá. Su diseño ligero y flexible lo hace ideal para conectar sensores, actuadores y dispositivos inteligentes en aplicaciones de IoT. Ejemplos de uso incluyen hogares inteligentes (control de luces, termostatos), agricultura de precisión (monitorización de cultivos), automóviles conectados (gestión de datos telemáticos) y automatización industrial (control de maquinaria).

1.1.6 Historia del PLC

Un **Controlador Lógico Programable (PLC, por sus siglas en inglés)** es un dispositivo electrónico diseñado para controlar procesos industriales y todo tipo de máquinas mediante programación. Este tipo de controlador permite automatizar tareas de forma precisa, sencilla y funcional, simplificando la gestión de sistemas complejos y mejorando la eficiencia en los entornos industriales. Los PLCs destacan por su **facilidad de programación**, **rápida ejecución** y capacidad para **manejar múltiples entradas** y salidas, lo que los hace aplicables tanto en proyectos pequeños como en sistemas avanzados.

El origen del PLC se remonta a finales de la década de 1960, como respuesta a la creciente necesidad de modernizar los procesos industriales y optimizar los sistemas de control. Antes de la aparición del PLC, los sistemas de automatización dependían de tecnologías basadas en circuitos de relés electromecánicos, que eran costosos, difíciles de mantener y extremadamente inflexibles frente a cambios en los procesos productivos.

La evolución del PLC comenzó con el desarrollo de la primera unidad funcional, el **Modicon 084**, creado en 1968 por **Dick Morley**, considerado el "padre del PLC" (ver *Figura 1.5*). Este dispositivo fue diseñado específicamente para la industria automotriz, con el propósito de sustituir los complicados sistemas de cableado de los relés utilizados en las líneas de ensamblaje de General Motors. El Modicon 084 revolucionó la automatización industrial al introducir una arquitectura basada en microprocesadores, que permitió realizar modificaciones en la lógica de control mediante programación, en lugar de reconfigurar físicamente el hardware.

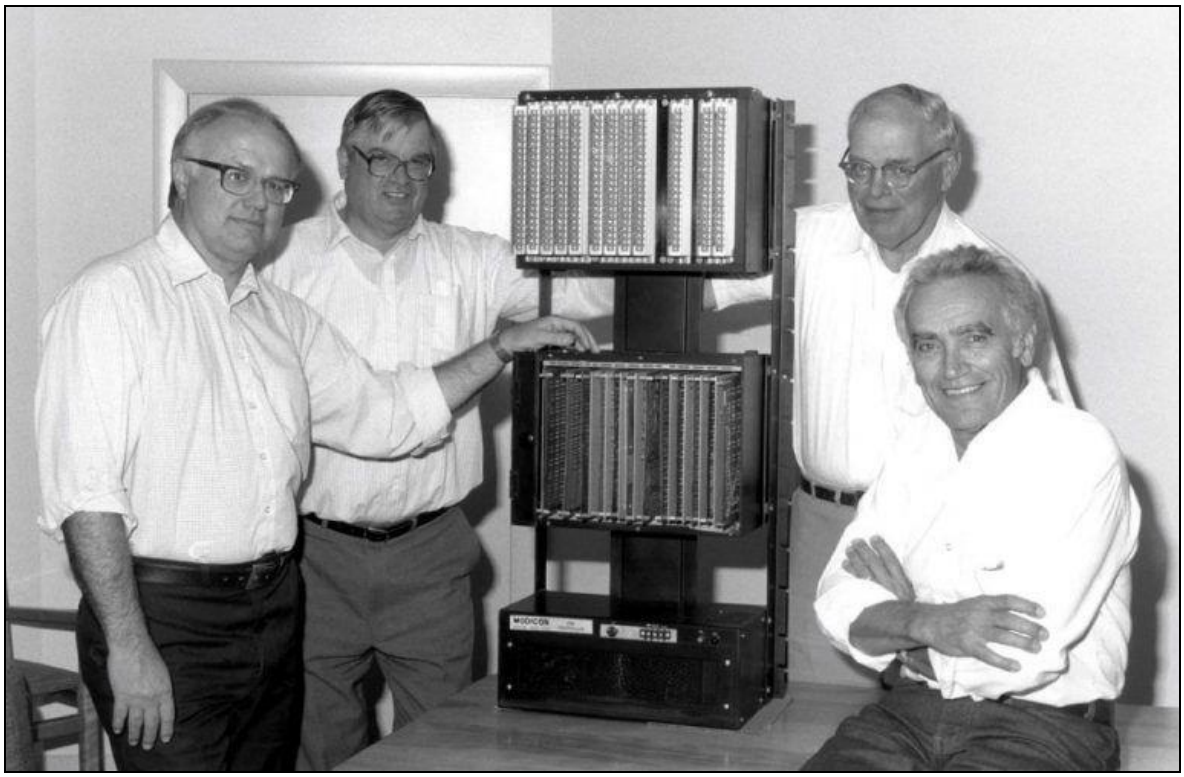

Figura 1.5: Fotografía histórica. El equipo de diseño del MODICON 084 junto a su creación. De izq. a dcha.: Dick Morley, Tom Boissevain, George Schwenk y Jonas Landau

Desde entonces, el PLC ha ido evolucionando hasta la actualidad:

- **Primera Generación (1968-1970s):** Los PLCs iniciales se basaban en una lógica de programación binaria similar a los esquemas de relés. Utilizaban lenguajes de programación rudimentarios como el lenguaje de escalera (Ladder Logic), que facilitaba la transición desde los esquemas eléctricos tradicionales hacia un modelo digital. Estos dispositivos estaban limitados en capacidad de memoria y procesamiento, y su principal uso era la sustitución directa de relés en tareas específicas de control.
- **Segunda Generación (1980s):** Durante esta etapa, los avances en la electrónica y la computación permitieron la integración de microprocesadores más potentes en los PLCs. Esto dio lugar a un aumento en la capacidad de memoria, la incorporación de módulos de entrada y salida modulares, y la posibilidad de comunicación con otros dispositivos mediante protocolos industriales. En esta etapa, surgieron estándares como el Protocolo Modbus, que marcó el inicio de la conectividad en redes industriales.
- **Tercera Generación (1990s):** Con el auge de la automatización distribuida y la adopción de redes de comunicación más avanzadas como Profibus y Ethernet Industrial, los PLCs evolucionaron hacia sistemas más inteligentes e interconectados. También se introdujeron capacidades de programación más versátiles, con lenguajes estándar definidos por la norma IEC 61131-3, como el Texto Estructurado y el Diagrama de Bloques Funcionales.

- **Cuarta Generación (2000s):** En el contexto de la digitalización, los PLCs comenzaron a incorporar funcionalidades propias de sistemas embebidos, integrando capacidades avanzadas de procesamiento, almacenamiento y conectividad inalámbrica. Además, se consolidaron las plataformas **multifunción**, que combinan control lógico, monitoreo y gestión de datos en un único dispositivo. Esta generación marcó el inicio de la integración de PLCs con sistemas SCADA y el IIoT.
- **Quinta Generación (Actualidad):** Los PLCs actuales son componentes fundamentales de los sistemas ciberfísicos en la Industria 4.0, caracterizados por su alta capacidad de procesamiento, conectividad en tiempo real y compatibilidad con tecnologías emergentes como **inteligencia artificial**, **analítica predictiva** y **gemelos virtuales**. Estos dispositivos están diseñados para operar en entornos altamente exigentes, con capacidad de integración en redes industriales avanzadas y compatibilidad con estándares abiertos como OPC UA y TSN (Time-Sensitive Networking).

El desarrollo del PLC transformó radicalmente los paradigmas de la industria manufacturera y de procesos, estableciendo las bases de la automatización moderna. Desde su invención, los PLCs han evolucionado para convertirse en herramientas versátiles y escalables, capaces de abordar desde tareas simples de control secuencial hasta aplicaciones complejas en sectores como la robótica, la energía renovable y la manufactura avanzada.

Hoy en día, los PLCs no solo representan un puente entre los sistemas físicos y digitales, sino que también desempeñan un papel esencial en la transición hacia fábricas inteligentes, donde la automatización, la conectividad y la inteligencia artificial son pilares fundamentales para la competitividad industrial.

1.1.7 Innovaciones Destacadas en los Últimos PLCs

En la Industria 4.0, los Controladores Lógicos Programables (PLCs) han evolucionado significativamente para adaptarse a los retos de la digitalización y la automatización avanzada. Estas innovaciones abarcan aspectos clave como conectividad, inteligencia, seguridad y sostenibilidad, posicionándolos como piezas esenciales en los sistemas industriales modernos, donde se busca maximizar la eficiencia y la integración tecnológica.

Una de las principales transformaciones en los PLCs recientes es su compatibilidad con protocolos de comunicación modernos como 5G, OPC UA, MQTT, y EtherNet/IP, lo que facilita su integración con plataformas IoT y **servicios en la nube**. Gracias a esta conectividad avanzada, los PLCs pueden interactuar con dispositivos de diferentes fabricantes, recolectar y transmitir datos en tiempo real, y garantizar la interoperabilidad entre sistemas.

Además de la conectividad, los PLCs más avanzados han incorporado **capacidades de procesamiento local (edge computing)**, lo que reduce la dependencia de la nube y minimiza la latencia. Estas capacidades permiten a los dispositivos analizar y actuar sobre los datos directamente en el lugar donde se generan, optimizando la toma de decisiones en tiempo real. Por ejemplo, en una línea de producción, un PLC con capacidades de edge computing puede identificar y resolver problemas de forma inmediata, evitando tiempos de inactividad y mejorando la eficiencia operativa, lo que resulta crucial en entornos industriales dinámicos.

Junto a estas mejoras, algunos PLCs modernos han sido diseñados para ejecutar **algoritmos de inteligencia artificial (IA)** y **machine learning (ML)**, permitiendo tareas como mantenimiento predictivo, análisis de patrones operativos y optimización de procesos. Estas capacidades los convierten en herramientas inteligentes capaces de prever fallos y detectar anomalías en equipos antes de que generen interrupciones. Por ejemplo, un PLC con IA puede analizar los datos provenientes de sensores en un motor industrial y predecir con precisión el momento ideal para realizar mantenimiento, minimizando costos y tiempos de inactividad.

Por otra parte, con el incremento en la conectividad de los sistemas industriales, la **seguridad cibernética** se ha convertido en una prioridad en el diseño de los PLCs modernos. Los últimos modelos incluyen hardware dedicado para cifrado, autenticación segura y protección contra accesos no autorizados, cumpliendo con estándares internacionales como IEC 62443. Estas medidas no solo garantizan la integridad y confidencialidad de los datos, sino que también protegen los procesos industriales de amenazas cibernéticas como accesos remotos no autorizados o manipulaciones de sistemas críticos.

Finalmente, los avances en el diseño de los PLCs también han priorizado la **eficiencia energética** y la sostenibilidad. Los dispositivos actuales son más compactos y consumen menos energía, sin sacrificar rendimiento. Esto no solo contribuye a la reducción de costos operativos, sino que también responde a las exigencias globales de sostenibilidad. Por ejemplo, en aplicaciones industriales, estos dispositivos optimizan el uso de energía en procesos como climatización y gestión de iluminación, ayudando a las empresas a cumplir con regulaciones medioambientales y objetivos de sostenibilidad.

En conjunto, estas innovaciones posicionan a los PLCs como herramientas fundamentales para la modernización industrial, garantizando que las empresas puedan enfrentar los desafíos de un entorno altamente digitalizado y competitivo, mientras optimizan sus operaciones de manera segura y sostenible.

1.1.8 Relevancia de los PLCs y los Gemelos Virtuales en la Industria 4.0

Una de las innovaciones más destacadas en el marco de la Industria 4.0 es la integración de los **Gemelos Virtuales (Digital Twins)**. Un gemelo virtual se define como una réplica digital de un sistema físico que facilita la simulación, el análisis predictivo y la optimización en tiempo real. Aplicada a los PLCs, esta tecnología permite no solo modelar su comportamiento, sino también interactuar con el entorno operativo en el que están integrados, generando una **sincronización bidireccional** entre los mundos físico y digital.

La combinación de PLCs y gemelos virtuales ofrece beneficios transformadores para la industria, entre los que se incluyen:

- **Simulación y validación previa**: Permite probar y validar procesos antes de implementarlos en el entorno físico, reduciendo errores y riesgos.
- **Optimización de procesos**: Facilita encontrar configuraciones más eficientes, mejorando la productividad.
- **Mantenimiento predictivo**: Predice fallos y permite planificar mantenimientos preventivos, reduciendo costos y riesgos.
- **Capacitación segura**: Proporciona un entorno virtual para entrenar operadores y técnicos sin riesgo de dañar equipos reales.

1.1.9 Competencia de los PLCs

Los PLCs son elementos esenciales en la automatización industrial, reconocidos por su robustez, flexibilidad y adaptabilidad en una amplia variedad de entornos. Sin embargo, a pesar de su liderazgo en automatización industrial, merecido por los avances anteriormente mencionados, **los PLCs enfrentan una creciente competencia** debido a la aparición de tecnologías alternativas y plataformas digitales:

- **Microprocesadores y Microcontroladores:** Microprocesadores como **Raspberry Pi** y microcontroladores como el **ATmega328P**, ofrecen soluciones más económicas y flexibles para aplicaciones específicas. Aunque no igualan la robustez de un PLC en entornos industriales exigentes, su bajo costo y capacidad de personalización los hacen atractivos para proyectos pequeños y medianos.
- **PCs Industriales (Industrial PCs, IPCs):** Los IPCs están diseñados para tareas más complejas y demandantes, como el procesamiento de datos y el análisis en tiempo real. Su potencia y compatibilidad con software estándar los hacen ideales para entornos que requieren alta capacidad de cálculo, pero pueden carecer de la resistencia inherente de un PLC en condiciones extremas.
- **Sistemas Embebidos con Edge Computing:** Soluciones con capacidad de procesamiento local y conectividad avanzada compiten directamente con los PLCs al ofrecer mayor flexibilidad para aplicaciones en tiempo real. Su integración con IA y análisis local reduce la dependencia de la nube y mejora la latencia en operaciones críticas.
- **Plataformas de Automatización en la Nube:** Servicios como **AWS IoT**, **Azure IoT** y **Google Cloud IoT** están redefiniendo la automatización industrial al integrar hardware y software en una arquitectura basada en la nube. Ofrecen escalabilidad y capacidades de análisis avanzado, aunque dependen de la conectividad estable.

1.1.10 Principales Marcas de PLCs Actuales

El mercado de los Controladores Lógicos Programables (PLCs) está liderado por empresas que han sabido adaptarse a las demandas de la Industria 4.0 mediante soluciones avanzadas y escalables. **Siemens**, con su serie SIMATIC S7, es una de las marcas más reconocidas, ofreciendo PLCs modulares como el mostrado en la *Figura 1.6* con integración en herramientas como TIA Portal, conectividad IoT mediante Profinet y OPC UA, y capacidades avanzadas de procesamiento en tiempo real. **Allen-Bradley** (Rockwell Automation) se posiciona con sus líneas ControlLogix y CompactLogix, que destacan por su alta velocidad de procesamiento, conectividad mediante EtherNet/IP y compatibilidad con plataformas IoT como FactoryTalk.

Figura 1.6: PLC actual, de la compañía SIEMENS

Beckhoff, por su parte, ha revolucionado el mercado con su enfoque en la automatización basada en PC y su serie CX (Embedded PCs), que combina las capacidades de un PLC con la potencia de un PC industrial. Sus soluciones destacan por la integración de TwinCAT, un entorno de software que permite implementar control en tiempo real, IoT y análisis de datos avanzados. Esta flexibilidad hace de Beckhoff una opción líder en aplicaciones que requieren alta precisión, procesamiento intensivo y conectividad avanzada. Además, **Schneider Electric**, con su serie Modicon, sobresale en sostenibilidad y análisis en la nube mediante su plataforma EcoStruxure, siendo una opción preferida para aplicaciones críticas en energía y automatización de edificios.

Estas marcas continúan liderando la evolución de la automatización industrial, integrando tecnologías como inteligencia artificial, edge computing y conectividad IoT en sus PLCs. Su capacidad para ofrecer soluciones que van desde sistemas compactos hasta fábricas inteligentes permite a las empresas optimizar procesos, mejorar la eficiencia operativa y adaptarse a los retos de la transformación digital

1.2 Presentación de la Empresa

Este proyecto ha sido desarrollado trabajando directamente con **Borrell S.A.** [6], una empresa ubicada en Denia, Alicante (ver la *Figura 1.7*), líder en el sector del procesamiento de frutos secos y un referente en la automatización industrial. La colaboración con Borrell ha permitido que este proyecto se enfoque en las necesidades reales del mercado, alineándose con las demandas de la Industria 4.0 y garantizando soluciones prácticas y eficientes tanto para uso interno como para el mercado global.

Para entender el contexto en el que se enmarca este proyecto, resulta esencial conocer más sobre la trayectoria de Borrell, sus hitos más relevantes y su impacto en la automatización y control industrial, aspectos que se detallarán a continuación.

1.2.1 Historia Borrell

La historia de **Borrell** comienza en **1922**, cuando su fundador, **Ángel Borrell Ivars**, diseñó e inventó la primera máquina partidora de almendras, conocida como **"La Ideal Almendrera"**. Este dispositivo marcó un punto de inflexión en la mecanización del procesamiento de frutos secos, al simplificar una tarea que hasta entonces se realizaba manualmente. En **1925**, este avance fue patentado, consolidando la posición de Borrell como un referente en la innovación tecnológica del sector.

Durante los primeros años, la empresa se enfocó en perfeccionar sus diseños. En los años **30**, introdujo importantes mejoras técnicas, como el sistema de eje de excéntricas, que reducía vibraciones en las máquinas, mejorando su eficiencia y aumentando su durabilidad. Estas innovaciones destacaron en un momento en que la mecanización industrial era incipiente, posicionando a Borrell como líder en el mercado español.

La década de **1950** marcó un hito significativo en la historia de la empresa. Bajo la dirección de **José Borrell Collado**, la empresa inició sus primeras exportaciones, llevando sus soluciones a mercados internacionales. Este período también estuvo caracterizado por la diversificación de su catálogo y la consolidación de su marca comercial, lo que fortaleció su identidad corporativa. A partir de entonces, Borrell no solo se centró en España, sino que también comenzó a expandir su influencia en otros países, adaptándose a las demandas específicas de cada mercado.

En las décadas posteriores, Borrell continuó evolucionando. Su compromiso con la innovación quedó demostrado con la obtención de más de **65 patentes**, reflejo de su liderazgo en el desarrollo de soluciones tecnológicas avanzadas para el procesamiento de frutos secos. Estas patentes abarcan desde sistemas para descascarado y calibrado hasta tecnologías para el secado, tostado y pasteurización.

El auge de la empresa también se reflejó en su capacidad para adaptarse a las transformaciones de la industria. A medida que surgían nuevas tecnologías, Borrell incorporó sistemas de automatización y control que optimizaban los procesos de producción, garantizando la calidad y eficiencia de sus máquinas. Esta visión estratégica le permitió mantenerse relevante en un mercado altamente competitivo.

En **2022**, Borrell celebró su **centenario**, un logro que subraya su solidez y legado en la industria del procesamiento de frutos secos. Este aniversario no solo destacó su trayectoria histórica, sino también su capacidad para innovar y responder a las exigencias de la Industria 4.0, con soluciones que integran IoT, inteligencia artificial y análisis de datos en tiempo real.

Hoy en día, Borrell sigue siendo un referente global, con una sólida presencia en mercados internacionales gracias a su filial **Borrell USA Corp.**, ubicada en **Merced, California**. Esta filial ha reforzado su capacidad de ofrecer soporte técnico, mantenimiento y maquinaria adaptada a las necesidades específicas del mercado norteamericano, consolidando su posición como líder en el sector.

La historia de Borrell es un testimonio de su compromiso con la excelencia, la innovación y la satisfacción del cliente, atributos que han guiado a la empresa desde su fundación hasta convertirse en un **actor clave en el ámbito del procesamiento de frutos secos a nivel internacional**.

Figura 1.7: Instalaciones Borrell, Denia

1.2.2 Rol de la Empresa en el Ámbito de la Automatización y Control Industrial

Borrell ha desempeñado un papel fundamental en la automatización y control industrial, especialmente en el procesamiento de frutos secos como almendras, avellanas y pistachos. A lo largo de su trayectoria, la empresa ha desarrollado soluciones integrales que abarcan desde maquinaria individual hasta líneas completas de producción, integrando tecnologías avanzadas para optimizar la eficiencia y calidad de los procesos industriales.

Para lograr este nivel de innovación, la empresa ha incorporado sistemas de control de línea y procesos que permiten la supervisión y gestión en tiempo real de las operaciones. Su plataforma de software ofrece funcionalidades HMI/SCADA, captura y análisis de datos, facilitando la integración con sistemas existentes y mejorando la toma de decisiones basadas en información precisa.

Además, Borrell ha obtenido la certificación UL 508A como fabricante de paneles de control y automatización, asegurando que sus soluciones cumplen con los más altos

estándares de seguridad y fiabilidad en la industria. Este reconocimiento refuerza su compromiso con la excelencia y la confianza en sus sistemas automatizados.

Una de las innovaciones más destacadas de Borrell en la actualidad es la **SmartSorter® Vision A** (ver *Figura 1.8*), una máquina diseñada específicamente para la detección y separación de almendras amargas. Esta tecnología utiliza un **sistema de visión artificial** avanzado combinado con espectrometría para analizar cada almendra en tiempo real. La detección se basa en la presencia de **amigdalina**, un compuesto característico de las almendras amargas, que se identifica mediante el análisis de la luz reflejada y absorbida por las almendras en diferentes espectros, permitiendo distinguir de manera precisa las almendras dulces de las amargas. Este innovador desarrollo fue posible gracias a la colaboración entre Borrell y el **Grupo de Espectrometría Atómica de la Universidad de Alicante**, cuya investigación académica aportó las bases científicas para perfeccionar el sistema de análisis espectral. El resultado es una solución altamente eficaz y patentada que no solo garantiza un producto final de alta calidad libre de almendras amargas, sino que también mejora significativamente la eficiencia operativa al reducir la intervención manual y optimizar los tiempos de producción en la línea.

Figura 1.8: Máquinas Borrell SmartSorter® integradas en los grupos de descascaración

1.3 Motivación del Proyecto

La motivación de este proyecto nace de una visión clara: **aportar un avance significativo en la automatización industrial**, ofreciendo un PLC innovador, versátil y altamente personalizable que responda a las demandas actuales de la Industria 4.0. La elección de este tema surge de la observación de un **hueco de mercado con un enorme potencial**; durante mi análisis, identifiqué que los PLC tradicionales, aunque siguen siendo esenciales en los entornos industriales, se han quedado anticuados en varios aspectos, especialmente en términos de conectividad, escalabilidad y capacidad para integrar tecnologías avanzadas como el IoT y el análisis masivo de datos. Este proyecto busca cerrar esa brecha y proporcionar una solución que no solo atienda las necesidades actuales, sino que también impulse la transición hacia sistemas más avanzados y conectados.

El interés de Borrell en este proyecto es doble: ofrecer una solución al mercado que se alinee con las exigencias de la Industria 4.0 y, al mismo tiempo, adoptar este PLC para uso interno, lo que representa beneficios significativos en términos de eficiencia operativa y reducción de costos. Con este PLC, la empresa logra centralizar todas las funciones de control y supervisión, eliminando la necesidad de tarjetas de entradas y salidas individuales y reduciendo considerablemente el cableado eléctrico, cables Ethernet, alimentación y switches, que de otro modo serían necesarios para cada tarjeta. Esta centralización no solo simplifica la configuración al eliminar la asignación de direcciones IP y otras configuraciones individuales, sino que también optimiza el espacio físico requerido en las instalaciones.

Adicionalmente, este PLC permite prescindir de una NUC[6], ya que todo el sistema se programa directamente en una **Orange Pi**, donde se implementa el PLC. Esto no solo reduce los costos asociados al hardware, sino que también simplifica la arquitectura del sistema, mejorando su mantenibilidad y escalabilidad a futuro.

Con esta propuesta, este proyecto no solo busca innovar en el mercado, sino también generar un impacto tangible dentro de la empresa, proporcionando un enfoque tecnológico que eleva la eficiencia operativa y marca un nuevo estándar en el ámbito del control y la automatización industrial.

6 NUC (Next Unit of Computing): Miniordenador compacto desarrollado por Intel, diseñado para ofrecer un alto rendimiento en un formato reducido, ideal para aplicaciones de escritorio, entretenimiento y entornos industriales.

1.4 Previsión de Uso

Este PLC se proyecta como una herramienta innovadora para la automatización industrial, diseñada con un enfoque claro: **permitir a usuarios sin experiencia en programación realizar tareas complejas de control y supervisión mediante interfaces gráficas intuitivas**. La previsión de uso abarca múltiples sectores y aplicaciones, ofreciendo soluciones accesibles y versátiles para diversas necesidades.

Por una parte, se pretende posicionar este PLC como una solución práctica tanto para la propia empresa desarrolladora como para cualquier otra del sector. Su diseño busca facilitar su adopción por usuarios sin conocimientos avanzados en programación, permitiendo:

- Configurar procesos industriales mediante herramientas visuales.
- Simplificar la implementación en tareas de automatización, reduciendo la curva de aprendizaje.
- Adaptar el PLC a diferentes requerimientos operativos, desde el control de líneas de producción hasta la integración con sistemas IoT, de manera eficiente y sin complejidad.

Por otra parte, un objetivo importante del proyecto es que este PLC sea **utilizado en centros educativos y universidades como herramienta de aprendizaje**. Su facilidad de programación y enfoque visual permitirán a estudiantes y docentes:

- Experimentar con conceptos de automatización sin necesidad de conocimientos de programación avanzada.
- Desarrollar proyectos prácticos de control y monitorización con un enfoque didáctico y accesible.
- Explorar aplicaciones en áreas emergentes como IoT, robótica y sistemas inteligentes, fomentando una formación práctica y adaptable.

Así pues, este proyecto busca posicionar el PLC como **una solución accesible y eficiente para usuarios no técnicos de centros educativos y empresas**, incluyendo a Borrell, ofreciendo un enfoque práctico y adaptado a las demandas de la Industria 4.0.

1.5 Contexto de Este Trabajo de Fin de Grado Dentro del Proyecto del PLC

Este Trabajo de Fin de Grado (TFG) se centra en una parte específica del desarrollo total del **PLC**, un proyecto altamente ambicioso que requiere un largo tiempo de desarrollo y múltiples etapas para su implementación completa. En particular, el alcance de este TFG abarca la **arquitectura de más bajo nivel**, incluyendo el diseño y desarrollo del **núcleo del PLC**, que constituye la columna vertebral técnica del sistema, la **base de datos** y **2 módulos de comunicación**.

En conjunto, estas áreas de trabajo establecen una base sólida para que el PLC pueda evolucionar en futuras fases del proyecto, donde se implementarán interfaces gráficas y capacidades de personalización.

Es importante destacar que el punto de partida de este TFG fue un “hola mundo” de la parte del maestro SPI (sin funcionar perfectamente), sin documentación, ni comentarios descriptivos. El firmware de las tarjetas periféricas (esclavos SPI) ya estaba completado. Por tanto, el PLC solo era capaz de comunicarse, con errores, con sus tarjetas periféricas.

2 Palabras Clave

Estas son las palabras clave que engloba este Trabajo de Fin de Grado:

PLC modular, Industria 4.0, controlador lógico programable, sistemas embebidos, escalabilidad, interoperabilidad, conectividad, eficiencia energética, sistemas de tiempo real, Docker, MariaDB, bases de datos, MQTT, Modbus TCP/IP, SPI, GPIO, Orange Pi Zero 3, tarjetas periféricas, C, programación multihilo, gestión de hilos, sincronización con mutex, entradas y salidas digitales, entradas y salidas analógicas, automatización industrial, innovación industrial, IoT, IIoT, open-source.

3 Objetivos del Trabajo de Fin de Grado

3.1 Objetivos Principales

El objetivo principal de este Trabajo de Fin de Grado es **diseñar y desarrollar un controlador lógico programable modular para la industria 4.0**. Además, se busca crear una solución tecnológica con impacto real en el mercado, destinada a cubrir una necesidad concreta en la sociedad y aportar valor práctico.

3.2 Objetivos Secundarios

Al seleccionar este tema para el Trabajo de Fin de Grado, me propuse los siguientes objetivos personales, los cuales han orientado mi dedicación al proyecto y alimentado mi motivación:

- Explorar las posibilidades que ofrece el IoT.
- Familiarizarme con los protocolos de comunicación industrial y de bajo nivel para optimizar la interacción entre dispositivos.
- Formarme en la **programación a bajo nivel** y en **sistemas embebidos**, ya que siempre me ha interesado entender cómo funcionan los sistemas desde su nivel más fundamental, abarcando aspectos como la interacción cercana entre software y hardware.
- Desarrollar un proyecto ambicioso que represente un reto significativo, permitiéndome aplicar y ampliar mis conocimientos aprendidos a lo largo de mis estudios.
- Generar un impacto tangible en la empresa donde trabajo, integrando un enfoque innovador que beneficie tanto a la organización como al sector industrial en general.
- Ganar experiencia desarrollando proyectos reales con proyección de mercado.

4 Requisitos del Proyecto

Este apartado detalla los requisitos funcionales y no funcionales que guían el desarrollo del PLC en el contexto de este Trabajo de Fin de Grado. Los requisitos funcionales abarcan las características y capacidades específicas que el PLC debe cumplir para satisfacer las necesidades del proyecto, como la comunicación interna, la gestión de tarjetas periféricas y la integración con protocolos industriales. Por otro lado, los requisitos no funcionales se enfocan en atributos de calidad, como la escalabilidad, mantenibilidad, compatibilidad y tolerancia a fallos, esenciales para garantizar el rendimiento y la fiabilidad del sistema en entornos industriales complejos y dinámicos.

4.1 Requisitos Funcionales

A continuación, se detallan los requisitos funcionales del PLC, definidos con base en los objetivos del proyecto y en el alcance establecido para este Trabajo de Fin de Grado.

Comunicación Interna

1. El PLC debe implementar una comunicación eficiente entre el núcleo y las tarjetas periféricas utilizando el protocolo SPI.
2. El sistema debe garantizar la bidireccionalidad de la comunicación entre tarjetas periféricas y tarjeta maestro, permitiendo el envío de comandos y la recepción de datos en tiempo real.
3.

Gestión de Tarjetas Periféricas

4. El PLC debe permitir la conexión, desconexión e intercambio de tarjetas periféricas en caliente, sin necesidad de reiniciar el sistema.
5. Las tarjetas periféricas deben configurarse automáticamente al ser añadidas, reconociendo su tipo y funcionalidad sin intervención manual.
6. El PLC debe poder controlar y modificar el estado de las entradas y salidas de las tarjetas desde el núcleo del PLC.

Base de Datos Centralizada

7. El PLC debe disponer de una base de datos centralizada**.**
8. La base de datos debe registrar eventos críticos y responder a ellos de manera automática.
9. La base de datos debe mantener un espejo actualizado del estado del PLC para:
   9.1. Controlar y gestionar las salidas del PLC en tiempo real.
   9.2. Monitorizar en tiempo real el estado de las entradas y salidas.
10. La base de datos debe permitir la integración de sistemas externos mediante consultas en tiempo real y actualizaciones, garantizando su compatibilidad con módulos superiores como Modbus TCP/IP y MQTT.

Módulo Modbus TCP/IP

El PLC debe incluir un módulo Modbus TCP/IP que funcione como servidor, capaz de:

11. Atender peticiones de clientes Modbus TCP.
12. Transformar dichas peticiones en operaciones sobre la base de datos que permitan la lectura y escritura de datos en el PLC.
13. Gestionar múltiples clientes simultáneamente, garantizando una operación estable y eficiente.

Módulo MQTT

El PLC debe incluir un puente MQTT que permita:

14. Volcar la configuración de tarjetas conectadas en tiempo real mediante la publicación de datos en topics de configuración.
15. Volcar el estado de las entrades y salidas de las tarjetas periféricas mediante la publicación de datos en topics de lectura.
16. Escuchar a los clientes MQTT en los topics de escritura para modificar valores en la base de datos y permitir así controlar las salidas del PLC desde topics MQTT.

Configuración y Control

17. Permitir la configuración de visibilidad de tarjetas para MQTT.
18. Asegurar la configuración de permisos de lectura/escritura para los diferentes tipos datos almacenados en la base de datos.

4.2 Requisitos No Funcionales

Escalabilidad

1. El diseño del sistema debe permitir la ampliación del número de tarjetas periféricas sin necesidad de reestructurar el núcleo.
2. Debe ser posible añadir nuevos módulos de comunicación y protocolos industriales sin afectar la operación del sistema base y sin necesidad de reestructurar el núcleo.

Mantenibilidad

3. El sistema debe estar diseñado de forma modular, facilitando la actualización o sustitución de componentes como tarjetas periféricas, módulos de comunicación y núcleo.
4. El código fuente debe seguir estándares de desarrollo, como nombres descriptivos, comentarios claros y estructura modular, para simplificar el mantenimiento y la evolución del proyecto.

Compatibilidad

5. El PLC debe ser compatible con los estándares industriales más utilizados, como Modbus TCP, MQTT, para integrarse fácilmente en entornos industriales heterogéneos.
6. Debe ser funcional sobre hardware basado en Linux, como Orange Pi.

Manejo de Errores y Tolerancia a Fallos

7. El sistema debe ser tolerante a fallos, garantizando que la funcionalidad general no se vea interrumpida por errores en alguno de sus componentes.
8. Ante fallos en la comunicación SPI, desconexión de tarjetas periféricas o interrupciones en los módulos de comunicación, el sistema debe ser capaz de aislar automáticamente el componente afectado y continuar operando con los demás elementos.
9. El PLC debe generar notificaciones detalladas para que el operador pueda identificar rápidamente la causa del fallo y tomar medidas correctivas.

5 Diseño

5.1 Arquitectura del Sistema

La arquitectura del sistema desarrollado para este PLC se basa en una **estructura modular** que combina componentes de hardware y software, permitiendo flexibilidad, escalabilidad y eficiencia en entornos industriales. Tal y como se observa en la *Figura 5.1*, todo el sistema se divide en las siguientes capas y componentes:

Hardware

- **Tarjetas Periféricas:** El PLC cuenta con 8 tarjetas periféricas responsables de manejar entradas y salidas analógicas y digitales. Estas tarjetas, en su firmware, están configuradas como esclavos SPI y se encargan de interactuar directamente con los sensores y actuadores conectados al PLC.
- **Maestro SPI:** La tarjeta con la Orange Pi Zero 3 actúa como el maestro SPI del sistema. Este dispositivo ejecuta un sistema operativo Debian 5.4 y todo el software del PLC, incluyendo la base de datos.

Software

- **Base de Datos:** La base de datos centralizada constituye el pilar fundamental en la arquitectura del PLC, actuando como un **gemelo virtual del PLC**. Su propósito principal es mantener un registro actualizado y accesible de las entradas y salidas del PLC, así como de las configuraciones específicas de las tarjetas periféricas y eventos críticos. Este diseño asegura que toda la información relevante esté disponible de forma inmediata para los demás módulos, estableciendo un flujo continuo, robusto y coordinado de datos. De esta manera, no solo almacena datos, sino que también actúa como un medio para que los módulos trabajen de forma sincronizada y consistente con el estado actual del PLC en tiempo real.
- **Núcleo del PLC:** El núcleo del PLC es el encargado de gestionar la comunicación SPI desde el maestro, permitiendo la interacción directa con las tarjetas periféricas (esclavos SPI), responsables de manejar las entradas y salidas del sistema. Además, actúa como un **puente esencial entre estas tarjetas y la base de datos centralizada**, asegurando que toda la información sobre el sistema esté siempre actualizada en ambos lados. Esta comunicación bidireccional, sincronizada y eficiente, garantiza un control preciso y confiable de las entradas y salidas del PLC desde las capas superiores, asegurando una integración fluida entre los componentes físicos y digitales del sistema.
- **Módulo Modbus TCP/IP:** Este módulo permite la interacción con sistemas SCADA y otros dispositivos industriales mediante el protocolo Modbus TCP/IP. Funciona como **un servidor multicliente** que:
    - Traduce las solicitudes Modbus TCP/IP en consultas a la base de datos.
    - Devuelve los resultados en el formato adecuado para el cliente.
- **Módulo MQTT:** Este módulo permite la comunicación del PLC con sistemas externos a través de enlazar el protocolo MQTT con la base de datos. Está organizado en tres grandes topics, que están gestionados por un broker:
    - **Lectura:** Publica el estado de las entradas y salidas del PLC en tiempo real para que los clientes suscritos puedan consultarlo.

- **Escritura:** Recibe comandos de los clientes del broker para modificar el valor de los registros asociados a las salidas de las tarjetas periféricas.
- **Configuración de tarjetas:** Publica en tiempo real la configuración de las tarjetas periféricas del PLC, incluyendo información sobre su orden de conexión, estado (conectadas o desconectadas) y los parámetros específicos de cada tarjeta.

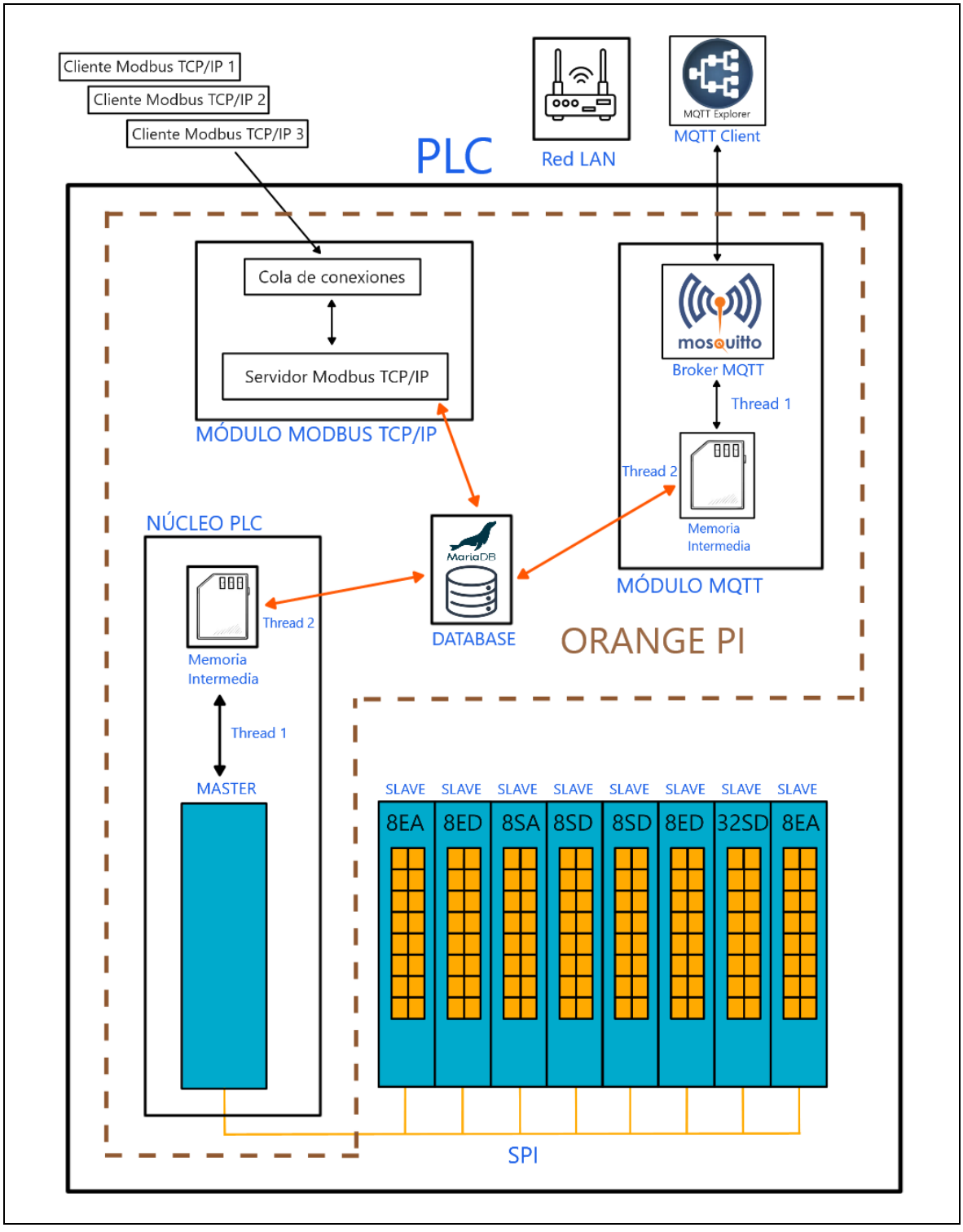

Figura 5.1: Arquitectura del PLC

5.2 Diseño del Hardware

El hardware del PLC (ver *Figura 5.2*) está compuesto por una **estructura modular** que incluye 8 tarjetas periféricas de entradas y salidas, y una tarjeta maestro SPI (ver *Figura 5.4*), que es donde se encuentra la Orange Pi Zero 3 y donde se ejecuta todo el software.

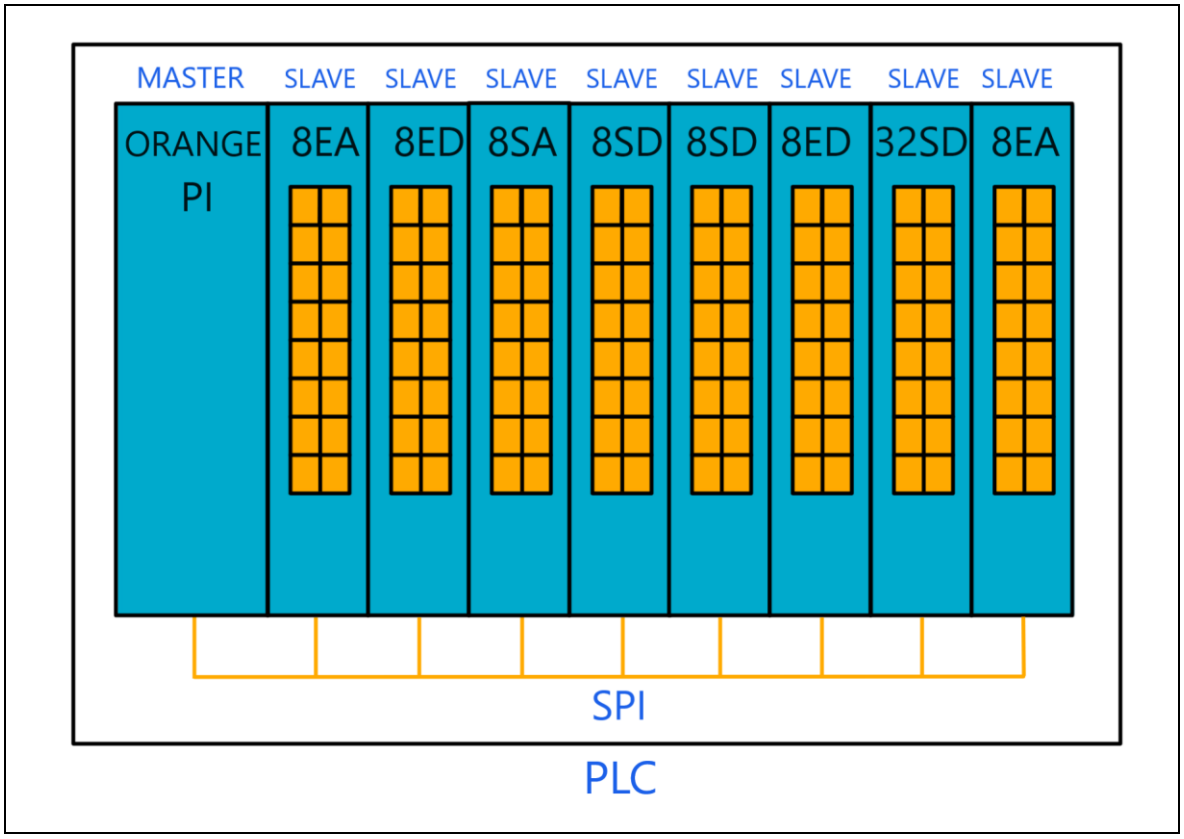

Figura 5.2: Hardware del PLC y conexión SPI. Las direcciones de los esclavos SPI van de 0-7, de izq. a dcha.

Estas tarjetas periféricas, diseñadas y fabricadas por la empresa Ibercomp[7] [7], son de 5 tipos diferentes, cada una con características específicas:

- **8SD (8 Salidas Digitales):** Proporciona 8 registros de tipo *bit*, asociados a 8 salidas digitales, y 16 registros de tipo *half*.
- **8ED (8 Entradas Digitales):** Proporciona 8 registros de tipo *bit*, asociados a 8 entradas digitales, y 16 registros de tipo *half*. En la *Figura 5.3* se puede ver un ejemplo real de este tipo de tarjeta conectada a 8 entradas digitales.
- **8SA (8 Salidas Analógicas):** Proporciona 8 registros de tipo *half*, asociados a 8 salidas analógicas, y 16 registros de tipo *half* adicionales.

[7] Ibercomp S.A.: Empresa de ingeniería ubicada en Palma de Mallorca, dedicada al diseño y fabricación de equipos electrónicos basados en microcontroladores desde 1988.

- **8EA (8 Entradas Analógicas):** Proporciona 8 registros tipo *half*, asociados a 8 entradas analógicas, y 16 registros de tipo half adicionales.
- **32SD (32 Salidas Digitales):** Proporciona 32 registros de tipo *bit*, asociados a 32 salidas digitales, y 16 registros de tipo *half*.

En la *Tabla 5.1* se puede ver la relación entre el tipo de entrada/salida y el tipo de registro asociado.

Tipo Entrada/Salida	Tipo de Registro E/S Asociado
Digital	*Bit*
Analógica	*Half*

Tabla 5.1: Relación entre el tipo de entrada/salida y el tipo de registro asociado

Cabe destacar que los 16 registros de tipo *half* de cada tarjeta periférica se utilizan como registros de configuración, almacenando información crítica sobre la tarjeta, como el número de entradas y salidas digitales, el número entradas y salidas analógicas, y un identificador único que representa el tipo de tarjeta. Tal y como se detallará más adelante, esta información es fundamental para las capas superiores.

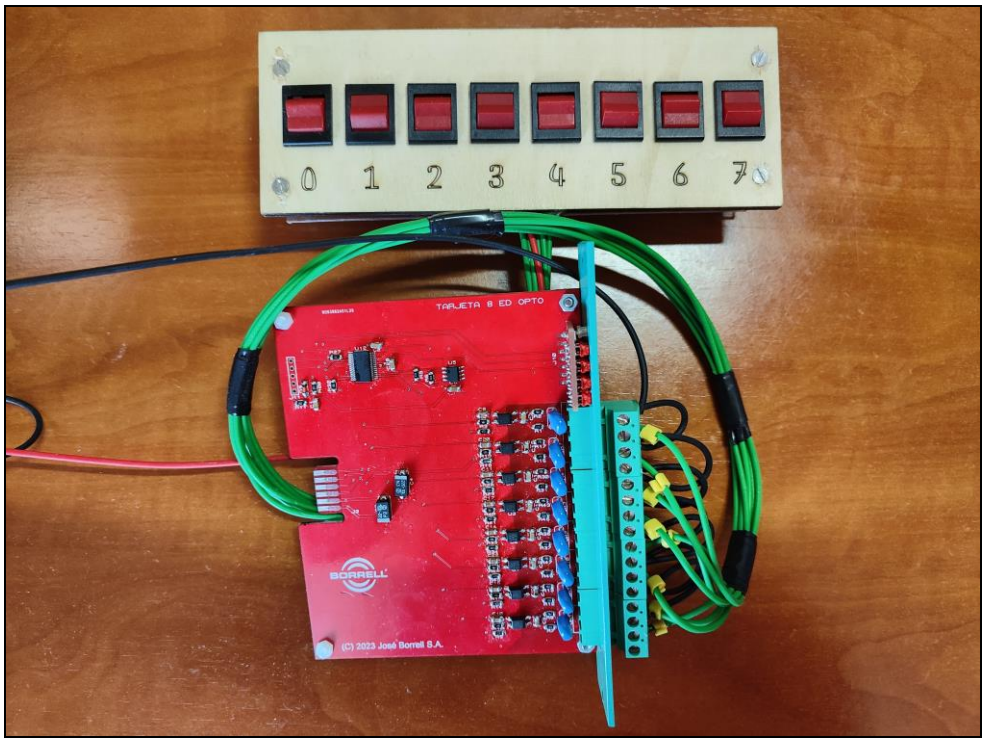

Figura 5.3: Tarjeta de tipo 8ED conectada a 8 interruptores

Para garantizar un acceso estructurado y eficiente a los datos, cada tarjeta periférica está conectada al sistema mediante una **dirección SPI única**, lo que permite al núcleo del PLC identificar y comunicarse fácilmente con ellas. Dentro de cada tarjeta, los registros E/S de tipo *bit* y tipo *half* también están organizados en un **sistema de direcciones**.

- **Nivel de tarjeta:** Cada tarjeta tiene una dirección única en el PLC.
- **Nivel de tipo de datos:** Dentro de cada tarjeta, los registros E/S están organizados en direcciones específicas para cada tipo (*bit* o *half*).

Este sistema de direcciones en capas estará mapeado directamente en la **base de datos centralizada**, estableciendo la base sobre la cual todos los módulos del PLC podrán acceder uniformemente a cualquier tipo de dato. Esta arquitectura garantiza que los módulos, como Modbus TCP/IP o MQTT, puedan interactuar con los registros asociados a las entradas y salidas y los registros de configuraciones de las tarjetas, de manera eficiente y consistente, proporcionando un control centralizado y una interoperabilidad completa entre el hardware y el software.

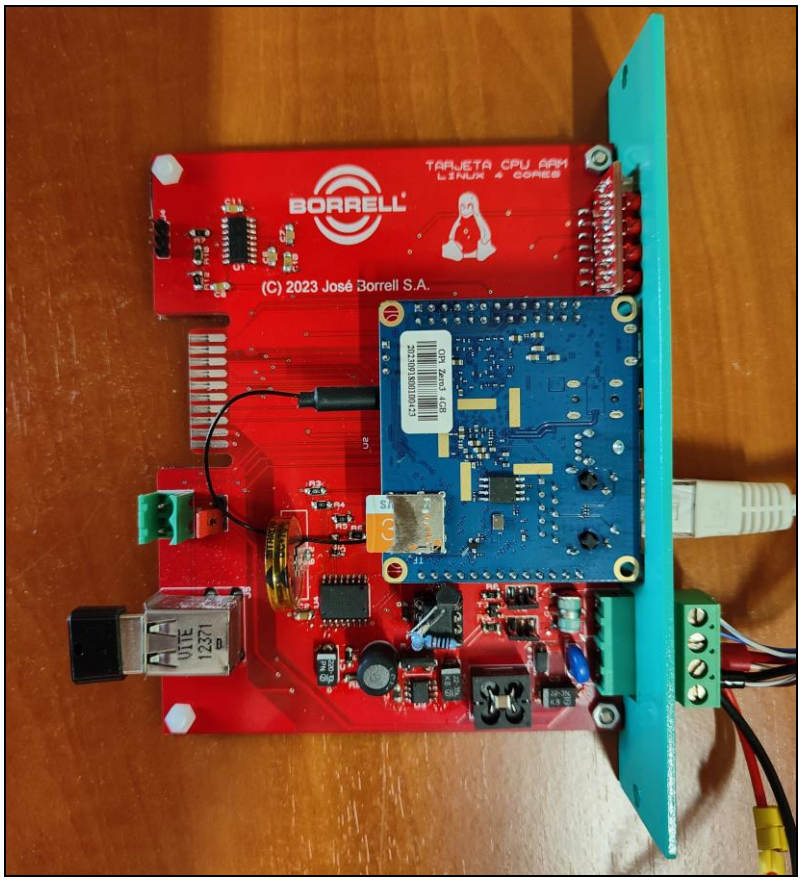

Figura 5.4: Tarjeta master SPI, con Orange Pi Zero 3

5.3 Diseño de la Base de Datos

Tal y como se ha mostrado anteriormente, la base de datos centralizada es el **núcleo de comunicación** del sistema, actuando como el punto de conexión entre todos los módulos que conforman el PLC y el núcleo. En ella se **almacena toda la información esencial** en tiempo real que permite la interacción de estos componentes y su sincronización con el PLC físico. A continuación, se detalla cómo se ha organizado todos estos datos en tablas.

5.3.1 Tablas de la Base de Datos

En este apartado se describen las principales tablas que conforman la base de datos del sistema desarrollado para el PLC modular. Cada tabla ha sido diseñada meticulosamente para garantizar la correcta gestión y organización de la información, reflejando fielmente el estado físico del sistema y permitiendo la interacción con los módulos superiores.

rtmirror: La tabla *rtmirror* es uno de los componentes centrales de la base de datos del PLC. Actúa como un gemelo virtual del PLC, reflejando tanto el estado de las entradas y salidas como los valores deseados para las salidas de cada tarjeta.

- **Campos de la Tabla:**
    - **module:** Representa el identificador único del slot del PLC donde está conectada la tarjeta. Este valor corresponde a la dirección SPI de la tarjeta (0-7).
    - **address:** Dirección específica de un registro E/S dentro de la tarjeta, que corresponde a un *half* o *bit* individual en SPI.
    - **value:** Almacena el valor actual (el que hay realmente en el PLC en ese instante) de un registro E/S. Este valor es actualizado en tiempo real por el núcleo del PLC.
    - **required_value:** Representa el valor que se desea poner en un registro asociado a una salida específica del PLC. Es utilizado por los módulos superiores (como MQTT o Modbus TCP/IP) para indicar el valor que se desea que tome dicha salida.
    - **timestamp:** Indica la última vez que se actualizó el valor. Es útil para realizar análisis históricos o diagnósticos.
    - **type_value:** Define si el *value* corresponde a un registro *bit* (valor digital) o a un registro *half* (valor analógico). Esto es esencial para establecer los permisos de lectura y escritura de cada dato, ya que los registros asociados a entradas solamente tendrán permisos de lectura (por lo que no se podrán modificar de ninguna manera) y los registros asociados a salidas tendrán permisos de lectura y escritura.
- **Clave primaria:** Se compone por (*module*, *address*, *type_value*). Esta combinación de campos permite una búsqueda rápida para cada dirección, de cada tipo de valor, de cada tarjeta periférica.

- **Ejemplo:** Supongamos 2 tarjetas conectadas al slot 2 y 3 del PLC, que tienen los siguientes datos:
    - Un *bit* en la tarjeta del slot 2, con dirección 3, con un valor actual de 1, actualizado por última vez el 7 de enero de 2025 a las 12:30:45, y un valor deseado de 0 (este valor puede ser enviado por un módulo superior, como el módulo Modbus TCP/IP o MQTT).
    - Un *half* en la tarjeta del slot 3, con dirección 5, con un valor actual de 150, actualizado por última vez el 7 de enero de 2025 a las 12:31:00, y un valor deseado de 200.

  El contenido de *rtmirror* sería el representado en la *Tabla 5.2*.

module	address	value	required_value	timestamp	type_value
2	3	1	0	2025-01-07 12:30:45	bit
3	5	150	200	2025-01-07 12:31:00	half

Tabla 5.2: Ejemplo explicativo de *rtmirror*

connected_modules: La tabla *connected_modules* es crucial para gestionar la conexión y configuración de las tarjetas periféricas en tiempo real. Permite al sistema mantener un registro actualizado de las tarjetas conectadas al PLC y proporciona información sobre las capacidades de cada una.

- **Campos de la Tabla:**
    - **connected:** Indica si una tarjeta periférica está actualmente conectada al PLC.
    - **timestamp:** Registra la última vez que se cambió el estado de conexión de la tarjeta. Esto es útil para rastrear eventos de conexión o desconexión.
    - **module:** Representa la dirección del slot del PLC donde está conectada la tarjeta. Este valor corresponde a la dirección SPI de la tarjeta.
    - **bits:** Indica el número de registros E/S de tipo *bit* que tiene la tarjeta. Es decir, el número de entradas y salidas digitales físicas que tiene la tarjeta conectada a ese slot. Esta información se extrae de uno de los 16 registros *half* de configuración que hay en cada tarjeta.
    - **halfs:** Especifica la cantidad de registros E/S de tipo *half* que tiene la tarjeta. Es decir, el número de entradas y salidas analógicas físicas que tiene la tarjeta conectada a ese slot. Esta información se extrae de uno de los 16 registros *half* de configuración que hay en cada tarjeta.
    - **id_type:** Identifica el tipo específico de tarjeta periférica conectada. Este campo ayuda a distinguir entre los diferentes modelos de tarjetas (por ejemplo, 8SD, 8ED, etc.) y permite, junto con la tabla *permission_map*, saber cuántas salidas de cada tipo tiene la tarjeta. Esta información se extrae de uno de los 16 registros *half* de configuración que hay en cada tarjeta.
- **Clave primaria:** Está compuesta por el campo (*module*). Garantiza que cada tarjeta conectada al sistema tenga una dirección SPI única.

- **Ejemplo:** En la *Tabla 5.3* se muestran 2 registros de la tabla *conected_modules*, con la siguiente información:
    - En el slot 7 del PLC hay conectada una tarjeta **8ED**, que cuenta con **8 entradas/salidas digitales** (en este caso, entradas, por el tipo de tarjeta). El estado de conexión de la tarjeta se estableció como conectado (TRUE) en la fecha y hora indicada.
    - En el slot 2 no hay ninguna tarjeta conectada desde la fecha indicada.

connected	timestamp	module	bits	halfs	id_type
TRUE	2025-01-07 10:15:30	7	8	0	17
FALSE	2025-01-05 11:23:03	2	NULL	NULL	NULL

Tabla 5.3: Ejemplo explicativo de *connected_modules*

configuration: La tabla *configuration* está destinada a almacenar datos de configuración global del sistema en formato **clave-valor**. Su propósito principal es centralizar la gestión de parámetros operativos y de estado. Por ejemplo, una fila de esta tabla almacena el número de tarjetas que hay conectadas en el PLC en cada instante.

- **Campos de la Tabla:**
    - **key:** Identifica de manera única cada parámetro de configuración. Este campo actúa como una etiqueta descriptiva que define el propósito del valor asociado.
    - **value:** Almacena el valor asociado a la clave, representando el estado o configuración específica del sistema en ese momento.
- **Clave Primaria:** Se compone por el campo (*key*).
- **Ejemplo:** En el ejemplo de la *Tabla 5.4* se representa que hay 7 tarjetas (del máximo, 8) conectadas actualmente en el PLC.

key	value
num_modules	7

Tabla 5.4: Ejemplo explicativo de *configuration*

permission_map: La tabla *permission_map* es fundamental para **reflejar de manera digital el funcionamiento físico real del PLC**, especialmente en la gestión de entradas y salidas. Define los permisos de acceso para cada tipo de registro E/S, reproduciendo digitalmente el funcionamiento de estos registros a nivel físico del PLC. La relación entre los registros E/S con los permisos de acceso, representada en la *Tabla 5.5*, es la siguiente:

- **Registros asociados a entradas:** Representan datos provenientes de dispositivos externos, como sensores o equipos de monitoreo, conectados al PLC. Estos registros son utilizados únicamente para la lectura, ya que almacenan señales que

llegan al PLC desde el exterior. Por su naturaleza, **ya desde bajo nivel no es posible escribir ni modificar directamente estos registros**, pues están diseñados exclusivamente para capturar información del entorno físico que llega por el pin de entrada al que están asociados. Por ello, y para preservar la consistencia, **se asignan permisos de solo lectura en la base de datos.**

- **Registros asociados a salidas:** Representan señales generadas por el PLC para controlar dispositivos externos, como actuadores, motores o luces, conectados al sistema. Estos registros permiten tanto la lectura como la escritura, ya que almacenan los valores que el PLC envía al pin de salida correspondiente. **Desde bajo nivel, es posible leer y modificar estos registros** porque están diseñados específicamente para gestionar el control de las señales salientes. Por esta razón, y para reflejar fielmente su comportamiento físico, **se asignan permisos de lectura y escritura en la base de datos**.

Entrada/Salida	Tipo de Señal	Tipo de Registro E/S Asociado	Permiso de Acceso en la Base de Datos
Entrada	Digital	*Bit*	Lectura
	Analógica	*Half*	Lectura
Salida	Digital	*Bit*	Lectura/Escritura
	Analógica	*Half*	Lectura/Escritura

Tabla 5.5: Relación entre la naturaleza de entrada/salida de una señal, el tipo de señal, el tipo de registro E/S asociado y el permiso de acceso en la base de datos.

Con esto, esta tabla garantiza que el comportamiento global del sistema (incluyendo módulos superiores) respete y reproduzca el funcionamiento físico y real de las tarjetas y sus pines de entradas y salidas. Módulos como Modbus TCP/IP y MQTT **utilizan esta tabla para validar las operaciones solicitadas, asegurándose de que cumplan con las restricciones de acceso definidas**. Por ejemplo, cualquier intento de un cliente de modificar un valor de entrada será rechazado automáticamente, dado que no está permitido modificar datos configurados con permisos de solo lectura.

Para que todo este sistema sea robusto y fiable, esta tabla es **inmutable** y está ***hardcodeada* desde el principio**, lo que significa que sus valores no cambian durante el tiempo de ejecución. Esto asegura que las reglas de acceso estén definidas de manera estática y consistente, proporcionando una base sólida y confiable para la operación del sistema.

- **Campos de la Tabla:**
    - **id_type:** Identifica el tipo de tarjeta periférica conectada (por ejemplo, 8SD, 8ED, etc.), asociando cada tipo de tarjeta con un set de configuración de permisos.
    - **type_value:** Clasifica si el permiso corresponde a un registro E/S digital (*bit*) o analógico (*half*).
    - **address:** Especifica la dirección dentro de la tarjeta que identifica de forma única el registro (es decir, la entrada/salida).

  - **permissions_rw:** Define el tipo de acceso permitido. Hay 2 tipos:
    - **read:** Para registros de entrada, que solo pueden ser leídos.
    - **readwrite:** Para registros de salida, permitiendo que el sistema pueda tanto leerlos como escribirlos.

- **Clave primaria:** Se compone por (*id_type*, *type_value*, *address*): Esta combinación de campos permite saber, para cada tipo de tarjeta, el permiso de acceso asociado a cada una de las direcciones de cada tipo de registro E/S (*bit* o *half*).
- **Ejemplo:** En la *Tabla 5.6* se muestra el contenido de la tabla *permission_map* correspondiente a la tarjeta 8ED (8 entradas digitales). En este caso, el identificador *id_type* asignado a esta tarjeta es 17, y cuenta con 8 registros E/S de tipo *bit* (en este caso, asociados a 8 entradas digitales). Como era de esperar, dado que estos registros representan el estado de las entradas del sistema, todos ellos tienen permisos de **solo lectura** (*read*), lo que garantiza que no puedan ser modificados por el PLC.

id_type	type_value	address	permissions_rw
17	bit	0	read
17	bit	1	read
17	bit	2	read
17	bit	3	read
17	bit	4	read
17	bit	5	read
17	bit	6	read
17	bit	7	read

Tabla 5.6: Ejemplo explicativo de *permission_map* para la tarjeta 8ED

visibility_config: La tabla *visibility_config* desempeña un papel clave en la gestión de qué datos del PLC son accesibles para los módulos superiores y bajo qué condiciones. Su principal función es establecer la visibilidad y los parámetros de actualización de cada *value* de la tabla *rtmirror*, proporcionando **control granular** sobre qué información debe ser publicada y cómo debe actualizarse. Esto es especialmente útil en un escenario distribuido donde los estados de las entradas/salidas del PLC son consumidos por clientes externos, como clientes MQTT.

- **Campos de la Tabla:**
  - **module:** Representa el identificador único del slot del PLC donde está conectada la tarjeta. Este valor corresponde a la dirección SPI de la tarjeta.
  - **address:** Dirección específica dentro de la tarjeta, que corresponde a un registro E/S concreto (*bit* o *half*).
  - **type_value:** Indica si el registro E/S es de tipo *bit* o un *half*.
  - **visibility:** Define si el dato es **visible** u **oculto** para los módulos superiores, permitiendo un control sobre qué información está disponible externamente.

  - **visibility_mode:** Especifica cómo se actualiza la visibilidad del dato, con dos opciones:
    - **periodically:** El dato se publica a intervalos regulares. Ideal para monitorización.
    - **changes:** El dato solo se publica cuando ocurre un cambio en su valor. Esto minimiza el tráfico de datos al publicar únicamente cuando hay un cambio, adecuado para aplicaciones sensibles a eventos.
  - **refresh_rate:** Indica el intervalo de tiempo, en milisegundos, para la actualización de datos cuando el modo de visibilidad es *periodically*.
- **Clave primaria:** Compuesta por (*module*, *address*, *type_value*). Garantiza que cada registro E/S del PLC tenga una configuración de visibilidad.
- **Ejemplo:** En la *Tabla 5.7* se muestra el contenido de la tabla *visibility_config* con 3 configuraciones de visibilidad diferentes:
  - La primera fila indica que el registro E/S *bit* de la dirección 5 de la tarjeta 3 es visible y se publica únicamente cuando cambia.
  - La segunda fila muestra que el registro E/S *half* de la dirección 2 de la tarjeta 4 es visible y se publica cada 2 segundos.
  - La tercera fila define que el registro E/S *bit* de la dirección 8 de la tarjeta 2 está oculto y, por lo tanto, no se publica.

module	address	types_value	visibility	visibility_mode	refresh_rate
3	5	bit	visible	changes	NULL
4	2	half	visible	periodically	2000
2	8	bit	hidden	NULL	NULL

Tabla 5.7: Ejemplo explicativo de *visibility_config*

5.3.2 Optimización de la Base de Datos y Nivel de Aislamiento

La optimización del rendimiento es un aspecto crucial en el diseño de la base de datos del PLC, especialmente dado el entorno **embebido** en el que opera. El PLC utiliza una **microSD como almacenamiento principal**, lo que introduce limitaciones importantes relacionadas con el número de operaciones de lectura y escritura que puede soportar este medio antes de deteriorarse. Para mitigar este problema y garantizar un funcionamiento eficiente, se han tomado las siguientes decisiones de diseño:

Uso de la RAM para Tablas de Alta Frecuencia

Todas las tablas críticas para el funcionamiento en tiempo real del PLC, como *rtmirror*, *connected_modules*, *configuration* y *visibility_config*, **están configuradas para residir en la RAM**. Esto permite:

- **Alta velocidad de acceso y actualización:** La RAM ofrece tiempos de respuesta significativamente más rápidos que la microSD, lo que es esencial para un sistema que requiere actualizaciones en tiempo real.
- **Reducción del desgaste de la microSD:** Al evitar operaciones frecuentes de escritura y lectura en la microSD, **se prolonga la vida útil** de este medio de almacenamiento.

Uso de la microSD para tablas inmutables

La única tabla que se encuentra almacenada de manera persistente en la microSD es ***permission_map***. Esta decisión se basa en las siguientes razones:

- **Naturaleza inmutable:** La tabla *permission_map* no cambia durante el tiempo de ejecución del sistema, ya que está *hardcodeada* con los permisos de acceso para cada tipo de registro E/S de las tarjetas periféricas.
- **Bajo impacto en el rendimiento:** Dado que esta tabla no requiere actualizaciones frecuentes, su almacenamiento en la microSD no afecta negativamente al rendimiento del sistema ni al deterioro de esta.
- **Persistencia garantizada:** Al estar en ROM, esta tabla asegura que **los permisos definidos estén disponibles incluso tras reinicios o cortes de energía**.

Nivel de Aislamiento SERIALIZABLE

Para garantizar la coherencia y la integridad de los datos en un sistema que opera con múltiples módulos y servicios simultáneamente, se ha configurado el nivel de aislamiento de las transacciones en **SERIALIZABLE**, el nivel más estricto disponible. Este nivel de aislamiento asegura que:

- **No ocurran interferencias entre transacciones:** Cada transacción se ejecuta como si fuera la única en el sistema, eliminando posibles conflictos de lectura y escritura en este entorno concurrente.
- **Integridad total de los datos:** Se evita la aparición de inconsistencias causadas por lecturas no repetibles, lecturas fantasma o condiciones de carrera.
- **Fiabilidad en sistemas distribuidos:** Dado que múltiples módulos (como Modbus TCP/IP y MQTT) y el núcleo acceden a la base de datos simultáneamente, este nivel de aislamiento garantiza que todas las operaciones sean seguras y confiables.

En conjunto, estas decisiones aseguran un equilibrio entre rendimiento, fiabilidad y longevidad del sistema, adaptándose a las limitaciones del hardware mientras se maximizan las capacidades del PLC.

5.4 Núcleo del PLC

El núcleo del PLC es un elemento fundamental del sistema, encargado de integrar las tarjetas periféricas conectadas al PLC con el sistema global. Coordina la comunicación mediante tres componentes principales, que trabajan conjuntamente para formar una **cola de mensajería bidireccional asíncrona** entre las tarjetas periféricas y la base de datos. Además, asume la responsabilidad de gestionar el protocolo SPI desde el lado del maestro. En los apartados siguientes se detallan las funciones y características de estos tres componentes.

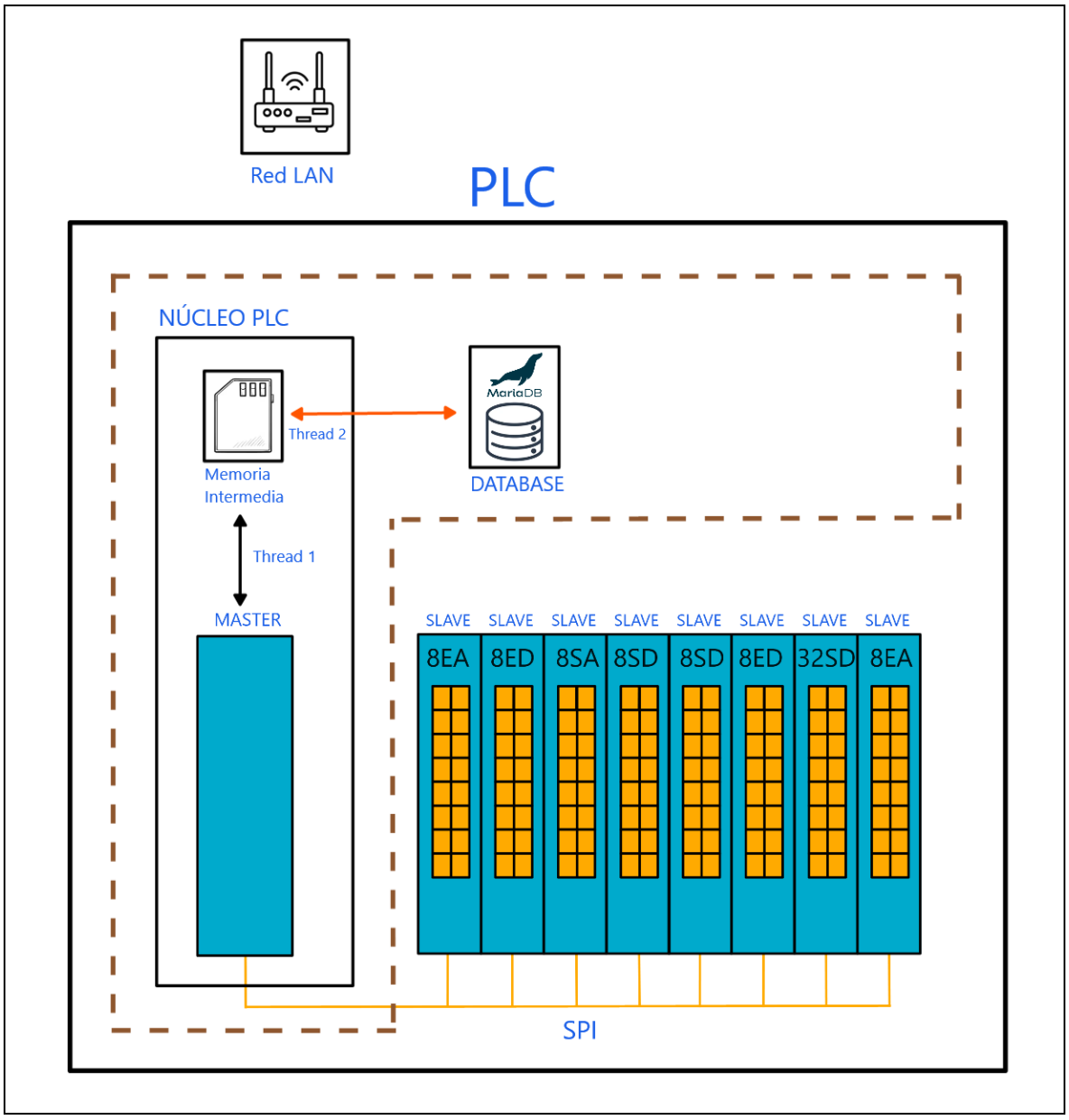

Figura 5.5: Esquema de diseño del núcleo del PLC

5.4.1 Memoria Intermedia

El componente central de este diseño es la memoria intermedia (ver *Figura 5.5*), una compleja estructura de datos diseñada específicamente para reflejar en tiempo real el estado de las tarjetas periféricas conectadas al PLC y los cambios deseados en las salidas por las capas superiores. Esta memoria actúa como un **puente entre el hardware y la base de datos**, almacenando de manera temporal y ordenada la información de las entradas, salidas, y datos de configuración. Es decir, **se trata de una memoria compartida que conecta los dos threads**. Por tanto, estamos hablando de que esta estructura de datos es la **sección crítica** del núcleo, por lo que su acceso está minuciosamente protegido por **mutexes**. Esto garantiza que el acceso concurrente a esta estructura se gestione de manera segura, evitando conflictos o inconsistencias durante las operaciones de lectura y escritura realizadas por los threads.

Su propósito principal es, por una parte, permitir una **comunicación bidireccional y asíncrona** entre hardware y base de datos. Por otra parte, minimizar el número de accesos directos al hardware y el número de operaciones contra la base de datos, optimizando el rendimiento del sistema y permitiendo un flujo de datos controlado y eficiente entre ambas capas.

Estos son todos los campos de la estructura de memoria intermedia:

- **idSlot:** Identifica de manera única la tarjeta periférica conectada al slot.
- **maxBits:** Define el número máximo de valores digitales (bits) que la tarjeta puede manejar.
- **maxHalfs:** Especifica la cantidad máxima de valores analógicos que la tarjeta puede manejar (sin tener en cuenta los 16 *halfs* de configuración que tienen todas las tarjetas).
- **bits:** Almacena los valores actuales de las entradas y salidas **digitales** de la tarjeta (en realidad, de los registros E/S asociados), actuando como un reflejo del estado físico del hardware.
- **halfs:** Este campo contiene los valores actuales de las entradas y salidas **analógicas** de la tarjeta (en realidad, de los registros E/S asociados).
- **required_bits:** almacena los valores deseados para las **salidas digitales**, que el sistema debe enviar a la tarjeta. Esto permite que clientes externos de los módulos de capas superiores puedan definir el comportamiento de las salidas digitales del hardware.
- **required_halfs:** almacena los valores deseados para las **salidas analógicas**, que el sistema debe enviar a la tarjeta. Esto permite que clientes externos de los módulos de capas superiores puedan definir el comportamiento de las salidas analógicas del hardware.
- **connected:** indica el estado actual del slot respecto a su conexión con alguna tarjeta. Incluye los diferentes estados:
    - **0:** Slot sin ninguna tarjeta conectada.
    - **1:** Slot desconectado e información modificada en la base de datos.
    - **2:** Slot con una tarjeta conectada.
    - **3:** Slot conectado e información modificada en la base de datos.

- **bits_modified:** Este campo utiliza un array de indicadores para señalar si un registro E/S *bit* ha cambiado su valor, antes de actualizarlo en la base de datos. Al registrar únicamente los cambios, el sistema optimiza la sincronización con la base de datos, reduciendo la carga de trabajo y mejorando la eficiencia general al evitar actualizaciones innecesarias.
- **halfs_modified:** Similar al campo anterior, este campo utiliza indicadores para señalar si algún registro E/S *half* ha sido modificado, antes de actualizarlo en la base de datos.
- **required_bits_modified:** Este campo registra qué *required_value* de tipo *bit* de la tabla *rtmirror* ha cambiado en la base de datos antes de ser enviado a la tarjeta. Esto permite al sistema actualizar las salidas digitales del hardware solo cuando sea necesario, garantizando un uso eficiente del bus SPI.
- **required_halfs_modified:** Similar al campo anterior, indica qué *required_value* de tipo *half* de la tabla *rtmirror* ha cambiado en la base de datos antes de ser enviado a la tarjeta. Esto asegura que las salidas analógicas sean actualizadas de manera eficiente, sin hacer operaciones redundantes sobre el bus SPI.
- **delete_rtmirror:** Sirve como un indicador para borrar la información relacionada con el slot en la tabla *rtmirror* de la base de datos. Esto es útil en casos de desconexión o reconfiguración de tarjetas, asegurando que los datos obsoletos se eliminen y no interfieran con el funcionamiento del sistema.

Los campos anteriores representan toda la información de una sola tarjeta periférica en el PLC. **Para representar el PLC en su totalidad, se utiliza un array que contiene 8 de estas estructuras, una por tarjeta**.

5.4.2 Thread 1: Volcado de Datos Entre las Tarjetas Periféricas y la Memoria Intermedia

El diseño de esta parte del sistema se centra en **gestionar el canal de comunicación bidireccional** entre las tarjetas periféricas y la memoria intermedia del PLC (ver *Figura 5.5*). Este proceso se lleva a cabo en un **thread dedicado**, que opera de manera continua para garantizar que los datos entre el hardware y la memoria intermedia se mantengan sincronizados en tiempo real a través del bus SPI. A continuación, se detalla el flujo del sistema:

Main

- Se inicializa el hardware (pines GPIO).
- Se leen parámetros de configuración de ficheros.
- Se lanza un thread que maneja el refresco de los leds del PLC.
- Se lanza un thread que se encarga de sincronizar el parpadeo los leds de todas las tarjetas periféricas con el parpadeo de sus leds.
- Se inicializan las estructuras de la memoria intermedia.
- Se lanza el thread 2.
- Comienza la ejecución del thread 1 (utiliza el thread del main).

Ciclo Principal

- **Barrido continuo:** Se realiza un bucle infinito donde, en cada iteración, se ejecuta un barrido sobre los slots, verificando su estado y actualizando los datos según corresponda.
- **Ajuste dinámico del tiempo:** Entre iteraciones, el sistema calcula el tiempo de espera necesario para mantener una frecuencia constante, cada 300 ms reales.

Barrido de Slots

- **Verificación de conexión:** Para cada slot, se verifica el estado actual de la tarjeta mediante la función *identifyCard*, que se ejecuta cada 10 segundos. Este intervalo está diseñado teniendo en cuenta que el cambio de tarjetas en caliente no es un evento muy frecuente en el sistema. Al espaciar las verificaciones en este lapso, se evita sobrecargar el bus SPI innecesariamente, preservando su eficiencia para la transferencia de datos esenciales y garantizando un equilibrio entre rendimiento y capacidad de respuesta ante cambios en el hardware.
- **Manejo de desconexiones:** Si se detecta que un slot ya no tiene una tarjeta conectada, se libera la memoria asociada al slot utilizando *free_slot_memory*, se marca como desconectado y se marca el flag *delete_rtmirror* para que la base de datos borre las filas de *rtmirror* y *visibility_config* asociadas a este slot.
- **Manejo de conexiones:** Si hay una tarjeta conectada al slot, se realiza una lectura por SPI de los registros E/S *bits* y *halfs* de la tarjeta. Seguidamente, si se detecta que la tarjeta se ha conectado a un slot previamente vacío, se inicializan sus datos en la memoria intermedia mediante *slot_connected*. Luego (o, directamente, si la tarjeta ya estaba conectada en la iteración anterior): se sincronizan los registros E/S *bits* y *halfs* entre las tarjetas y la memoria intermedia en ambas direcciones utilizando *slot_still_connected*.

Seguidamente, se describen las principales funciones utilizadas en este proceso:

- **identifyCard:** Identifica una tarjeta en un canal SPI, leyendo la información desde los registros *half* de configuración (ID del tipo, número de *bits* y número de *halfs*) y almacenándola en la memoria intermedia. Devuelve un código de error si la respuesta es incorrecta o incompleta.
- **slot_connected:** Inicializa la memoria intermedia para una tarjeta recién conectada, reservando memoria dinámica para los campos necesarios en función del tipo de tarjeta e inicializándolos. Finalmente, marca el slot como "conectado" en el flag correspondiente.
- **slot_still_connected:** Realiza 2 acciones:
    - Vuelca a la memoria intermedia los valores actuales de las entradas y salidas de la tarjeta mediante la función *slot2struct*.
    - Realiza la operación inversa, actualizando las salidas de la tarjeta con los cambios en la memoria intermedia mediante *struct2slot*.
- **slot2struct**: Compara los valores de los registros E/S leídos en la tarjeta con los valores almacenados en la memoria intermedia y, si son diferentes (o si en la iteración anterior no estaba conectada la tarjeta), los vuelca a memoria. Después,

marca como modificados los valores que han cambiado, optimizando las futuras actualizaciones hacia la base de datos.

- **struct2slot**: Verifica si existen cambios en los valores de los registros asociados a salidas almacenados en la memoria intermedia, provenientes de la base de datos (*required_value* de la tabla *rtmirror*). Para cada dirección de salida que haya sufrido un cambio, se envía el nuevo valor actualizado al registro *bit*/*half* asociado a esa salida a través del bus SPI. Este enfoque optimiza el uso del bus SPI al evitar transmisiones innecesarias. Una vez enviado el valor, se desactiva el flag correspondiente, indicando que el cambio ha sido procesado.
- **free_slot_memory:** Libera la memoria dinámica asociada a un slot desconectado, asegurando la correcta gestión de recursos. También, marca el slot como "desconectado" y limpia cualquier dato relacionado con la tarjeta previa. Por último, marca a TRUE el flag *delete_rtmirror* para indicar a la base de datos que en la siguiente iteración ha de eliminar las filas de *rtmirror* y *visibility_config* asociadas a este slot.

5.4.3 Thread 2: Volcado de Datos Entre la Memoria Intermedia y la Base de Datos

El diseño de este módulo se centra en la gestión del **canal de comunicación bidireccional entre la memoria intermedia y la base de datos** (ver *Figura 5.5*), en un thread dedicado. A continuación, se describe el flujo general del sistema:

Inicialización

- **Conexión inicial:** Se establece una conexión con la base de datos. Si la conexión falla, el sistema termina para evitar inconsistencias.
- **Inicialización de tablas:** Inserta registros iniciales en la tabla *connected_modules*, marcando como vacíos los slots sin tarjetas conectadas. También, borra datos obsoletos de ejecuciones anteriores.

Ciclo Principal

- **Iteración continua:** Se ejecuta un bucle infinito que recorre todos los slots del PLC. Por cada slot, se realizan las siguientes tareas.
- **Volcado de datos prioritarios:** Los valores de los registros E/S, marcados como modificados, se vuelcan de la memoria intermedia a la base de datos con *struct2db_prior* cada 300 ms.
- **Volcado de datos secundarios:** Datos de baja prioridad, como configuraciones específicas, se vuelcan a la base de datos cada 600 ms mediante *struct2db_less*.
- **Sincronización inversa:** Los valores deseados (*required_value*) en la base de datos se vuelcan a la memoria intermedia con *db2struct* y se marcan como modificados para que se acaben actualizando en las tarjetas.
- **Ajuste dinámico del tiempo:** Calcula el tiempo restante de la iteración para garantizar un ciclo constante de 300 ms reales.

A continuación, se describen las funciones principales:

- **struct2db_prior:** Vuelca, desde la memoria intermedia a la base de datos (columna *value* de la tabla *rtmirror*), el valor actual de los registros E/S de un slot que están marcados como modificados en memoria intermedia. Una vez completado, se reinician los flags de modificación, marcándolos como sincronizados con la base de datos.
- **struct2db_less:** Maneja la sincronización de datos secundarios, como configuraciones específicas o registros E/S de baja prioridad.
- **db2struct:** Recupera los valores deseados (*required_value*) desde la base de datos y los vuelca en la memoria intermedia. Una vez recuperados, compara estos valores con los actuales en la memoria intermedia y, si hay discrepancias, los actualiza y marca los flags correspondientes para que sean aplicados al hardware en iteraciones futuras. Después de la sincronización, los *required_value* en la base de datos se ponen a NULL para evitar redundancias.
- **connected_modules:** Gestiona los cambios en el estado de conexión de un slot actualizando las tablas relevantes en la base de datos. Cuando una tarjeta se conecta, inserta un registro en la tabla *connected_modules*, estableciendo los valores de configuración (id, número de *bits* y número de *halfs*). Además, añade la clave primaria en las tablas *rtmirror* y *visibility_config* para la nueva tarjeta, junto con los valores predeterminados de visibilidad en la tabla *visibility_config*. Si se detecta una desconexión, actualiza el estado de la tarjeta en la tabla *connected_modules*, mientras que elimina todos los datos relacionados en las tablas *rtmirror* y *visibility_config*. Estas operaciones aseguran que las tablas reflejen el estado actual de las tarjetas conectadas al PLC en tiempo real.

5.5 Modbus TCP/IP

El módulo Modbus TCP/IP es una pieza fundamental del sistema que permite la comunicación entre múltiples clientes Modbus y el PLC, tal y como se representa en la *Figura 5.6*. Este módulo asegura una gestión eficiente, concurrente y segura de las operaciones de lectura y escritura de datos **mediante el protocolo estándar Modbus TCP/IP** a través de un servidor Modbus TCP/IP.

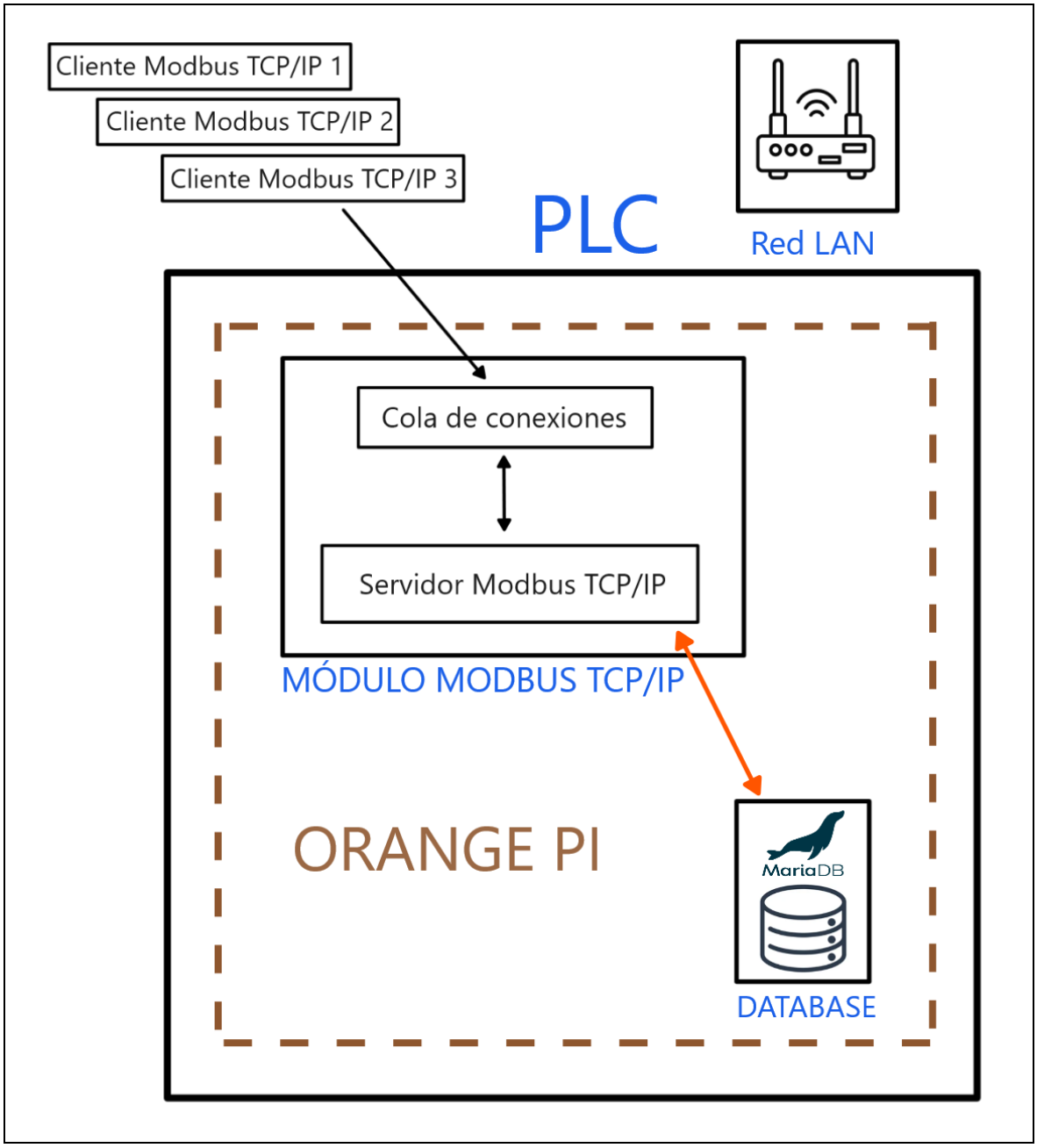

Figura 5.6: Diseño del módulo Modbus TCP/IP

Para entender correctamente cómo se relacionan las capas inferiores (el núcleo del PLC y la base de datos) con este módulo, es imprescindible explicar la correspondencia de los tipos de datos del núcleo y la base de datos con los tipos de datos propios del protocolo Modbus, resumida en la *Tabla 5.8*.

- **Bit Lectura/Escritura → Coil:**
    - Un **coil** en Modbus corresponde a una posición que puede ser leída o escrita como un único bit. En la base de datos, los registros E/S de tipo *bit* con permiso de lectura/escritura se mapean como un coil en este módulo.
- **Bit Lectura → Discrete Input:**
    - Un **discrete input** en Modbus es un bit de solo lectura. En la base de datos, los registros E/S de tipo *bit* con permiso exclusivo de lectura se mapea como un discrete input en este módulo.
- **Half Lectura/Escritura → Holding Register:**
    - Un **holding register** en Modbus es un registro de 16 bits que puede ser leído y escrito. En la base de datos, los registros E/S de tipo *half* con permiso de lectura/escritura se mapea como un holding register en este módulo.
- **Half Lectura → Input Register:**
    - Un **input register** en Modbus es un registro de 16 bits de solo lectura. En la base de datos, los registros E/S de tipo *half* con permiso exclusivo de lectura se mapean como un input register en este módulo.

Núcleo del PLC			**Base de Datos**	**Modbus**
Entrada/Salida	**Tipo de Señal**	**Tipo de Registro E/S Asociado**	**Permiso de Acceso en la Base de Datos**	**Tipo de Dato**
Entrada	Digital	*Bit*	Lectura	Discrete Input
	Analógica	*Half*	Lectura	Input Register
Salida	Digital	*Bit*	Lectura/Escritura	Coil
	Analógica	*Half*	Lectura/Escritura	Holding Register

Tabla 5.8: Relación entre el protocolo de datos de las capas inferiores y el protocolo de datos Modbus

A continuación, se describe el flujo de ejecución de este módulo:

Inicialización del Servidor

- **Creación del contexto:** Se configura un contexto Modbus TCP en el puerto 502 (estándar).
- **Creación del socket del servidor:** Se inicializa un socket que escucha conexiones entrantes.
- **Configuración inicial:** Se configura una estructura de monitoreo para rastrear eventos en los sockets del servidor y los clientes.

Ciclo Principal

- **Monitoreo de eventos en los sockets:** Se utiliza una función bloqueante que **detiene el thread** hasta detectar eventos en el socket del servidor (para detectar conexiones entrantes) y en los sockets de los clientes (para detectar peticiones Modbus TCP/IP entrantes).
- **Gestión de nuevas conexiones:** Si el socket del servidor detecta un evento, se acepta una nueva conexión de cliente. Si no se ha alcanzado el límite máximo de clientes, el nuevo cliente se agrega a la cola de sockets monitoreados. Si el límite se ha alcanzado, la conexión del cliente es rechazada.
- **Monitoreo de solicitudes de clientes conectados:** Se iteran los sockets de los clientes en la cola para verificar si han enviado una petición Modbus TCP/IP. Si un cliente cierra la conexión, su socket es eliminado de la cola. Si un cliente envía una petición válida, se procesa la solicitud.

Procesamiento de Solicitudes del Cliente

- **Recepción y validación de solicitudes:** Se recibe un marco Modbus. Si el cliente envía datos incorrectos o se desconecta, su socket es cerrado y eliminado de la cola.
- **Validación de direcciones y conexión de slot:** Se verifica que la dirección del slot solicitado exista y que tenga una tarjeta conectada, a través de una consulta en la base de datos.
- **Creación de un mapa Modbus TCP/IP:** Se genera un mapa Modbus específico para la tarjeta solicitada por el cliente. Este mapa sirve como una estructura temporal en memoria que almacena los datos (coils, discrete inputs, holding registers e input registers) asociados a la tarjeta.
- **Lectura y escritura de datos:** Primeramente, se valida que las direcciones de los tipos de datos involucrados existan en la tarjeta conectada al slot. Después, se maneja la solicitud en base a las funciones contempladas (ver *Tabla 5.9*):
    - Si el cliente solicita leer entradas y salidas del PLC, (código de función: 0x01, 0x02, 0x03, 0x04), se obtienen los valores de la base de datos y se cargan en el mapa Modbus.
    - Si el cliente solicita escribir salidas del PLC (código de función: 0x05, 0x06, 0x0F, 0x10), los datos enviados por el cliente se escriben en la base de datos.
- **Envío de respuesta al cliente:** Se envía la respuesta al cliente, que puede incluir los datos solicitados de la base de datos o un mensaje de error.

Código de Función (Hex)	Descripción
0x01	Leer Coils
0x02	Leer Discrete Inputs
0x03	Leer Holding Registers
0x04	Leer Input Registers
0x05	Escribir un Coil
0x06	Escribir un Holding Register
0x0F	Escribir múltiples Coils
0x10	Escribir Múltiples Holding Registers

Tabla 5.9: Códigos de Función Modbus TCP/IP contemplados en el módulo

A parte de ver el flujo de ejecución, es imprescindible describir las funciones principales involucradas:

- **create_modbustcp_map:** Esta función es responsable de crear un mapa Modbus, que representa la memoria de una tarjeta específica dentro del sistema. Este mapa permite que el servidor gestione las operaciones Modbus TCP/IP sobre los tipos de datos del propio protocolo. Estos son los pasos principales:
    - La función hace diferentes consultas a la base de datos para obtener el número de elementos de cada tipo (coils, discrete inputs, holding registers e input registers) asociados a la tarjeta.
    - Una vez se conoce la cantidad de valores de cada tipo de dato, se crea el mapa Modbus, asignando la memoria correspondiente.
- **get_real_starting_address:** Esta función determina la dirección real en la base de datos para un valor específico solicitado por un cliente Modbus. Esto es necesario porque los tipos de datos de Modbus y, por tanto, el sistema de direcciones, no coinciden con los tipos de datos del núcleo del PLC y la base de datos (ver relación entre tipos en la *Tabla 5.8*).
    - Se ejecuta una consulta SQL para obtener la dirección del primer valor de la tupla: tipo de registro E/S y permisos de acceso.
    - Una vez obtenida la dirección base, se suma el desplazamiento relativo proporcionado en la solicitud Modbus. Este cálculo produce la dirección real en la base de datos de los datos solicitados por el cliente Modbus TCP/IP, la cual es necesaria para las siguientes funciones.
- **db_modbustcp:** Esta función se utiliza para obtener datos de la base de datos según las solicitudes de un cliente Modbus.
    - Se ejecuta una consulta SQL para obtener el rango de valores (columna *value* de la tabla *rtmirror*) solicitados por el cliente Modbus TCP/IP.
- **modbustcp_db:** Esta función escribe los valores enviados por un cliente Modbus en la base de datos.
    - Se recorren los valores proporcionados por el cliente y se actualizan en el campo *required_value* de la tabla *rtmirror*.

5.6 MQTT

El módulo MQTT es otra pieza fundamental del sistema, diseñada para **gestionar la comunicación entre la base de datos del PLC y los clientes MQTT**. Este módulo permite la transmisión **eficiente**, **asíncrona** y **bidireccional** de datos, garantizando la integración entre las capas internas del PLC y el ecosistema de IoT. Similar al núcleo del PLC, su arquitectura está compuesta por **cuatro componentes** (ver *Figura 5.7*) que trabajan en conjunto para constituir una **cola de mensajería bidireccional asíncrona** entre la base de datos, la memoria intermedia del módulo MQTT, el broker MQTT y los clientes MQTT. Para facilitar la identificación de la funcionalidad de cada combinación de: registro E/S con tipo de registro (*bit*/*half*) con permisos de lectura/escritura, se han utilizado los tipos de datos definidos en el protocolo Modbus. Esta decisión permite una mayor claridad en la representación y gestión de los datos en los topics MQTT.

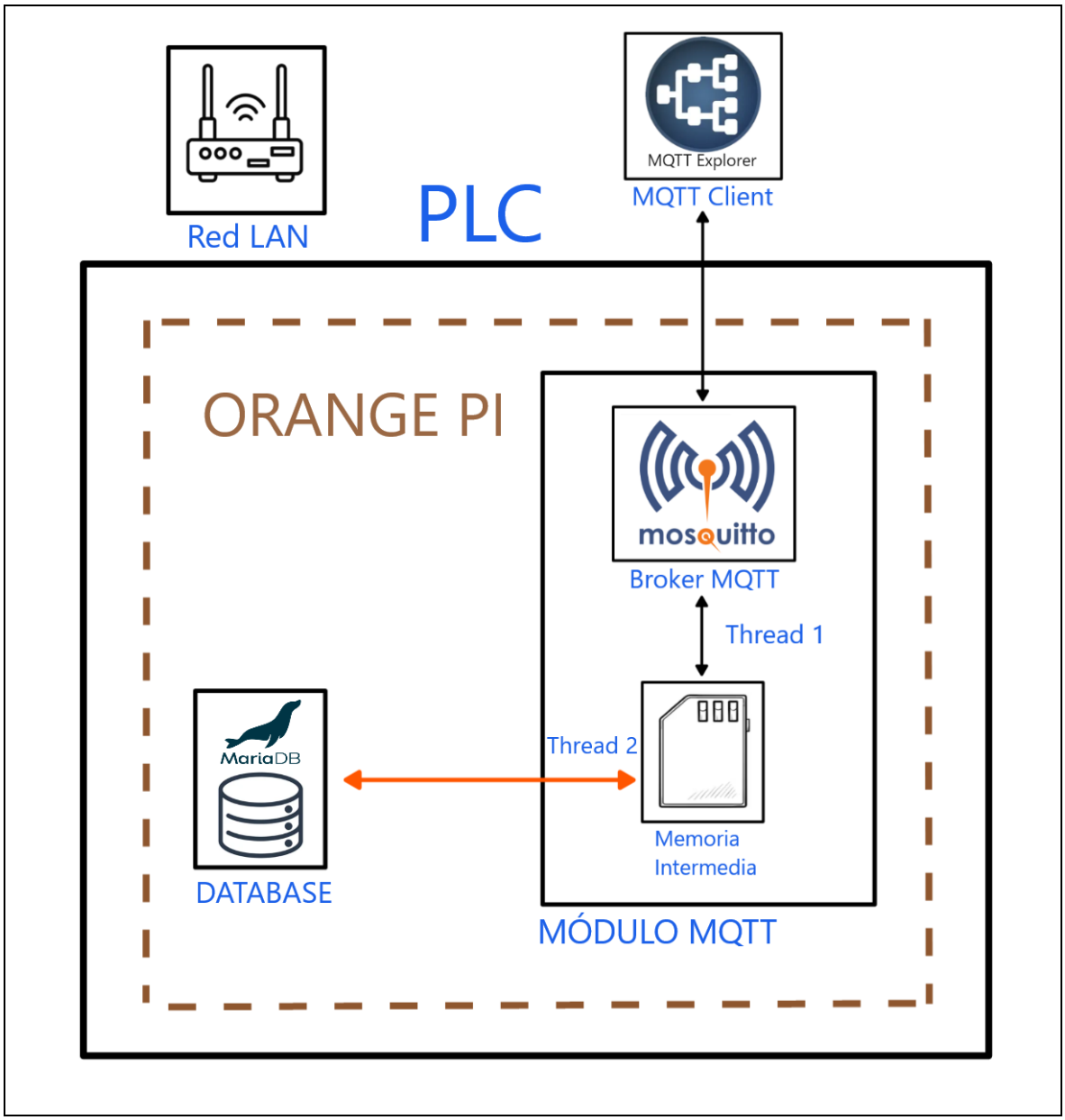

Figura 5.7: Diseño del módulo MQTT

5.6.1 Broker

El broker MQTT es uno de los 4 componentes del módulo MQTT, que actúa como un intermediario entre los clientes MQTT y el thread 1. Su principal función es gestionar la **publicación y suscripción de topics** automáticamente, permitiendo una comunicación estandarizada y eficiente en el sistema.

En cuanto a los topics que gestiona el broker, se ha decidido utilizar una nomenclatura organizada y coherente, dividiéndolos en tres categorías principales:

Topics de lectura (data topic)

Estos topics permiten a los clientes obtener información en tiempo real sobre el estado de las entradas y salidas de las tarjetas conectadas al PLC.

- **Estructura del *topic*:** /plc/data/module_<n>/<tipo>/<direccion>. Donde:
  - <**n**>: Número de la tarjeta. Es decir, dirección SPI. (ej. module_0).
  - <**tipo**>: Tipo de dato (ej. coils, discrete_inputs, holding_registers, input_registers).
  - <**direccion**>: Dirección específica del dato (ej. 0, 1, etc.).
- **Ejemplo de topic:** Con el siguiente *topic*, el cliente puede leer el valor del coil (en capas inferiores, registro de tipo *bit* con permiso de lectura/escritura asociado a una salida digital) número 5 de la tarjeta conectada al slot 2 del PLC: /plc/data/module_2/coils/5.

Topics de escritura (write topic)

Estos topics permiten a los clientes enviar mensajes en un formato determinado para modificar los *required_value* de la base de datos y poder modificar así las salidas de las tarjetas del PLC.

- **Estructura del topic:** /plc/write/module_<n>/<tipo>/<direccion>. Donde los componentes tienen la misma nomenclatura que los topics de lectura.
- **Ejemplo de topic:** Enviando un payload con valor 100 en el siguiente *topic*, el cliente envía el valor 100 al holding register (en capas inferiores, registro de tipo *half* con permiso de lectura/escritura asociado a una salida analógica) número 3 de la tarjeta conectada al slot 1 del PLC: /plc/write/module_1/holding_registers/3.

Topics de configuración (modules_config topic)

Estos topics contienen información sobre la configuración de las tarjetas en tiempo real, como el número de elementos (coils, discrete inputs, holding registers e input registers) y el estado de conexión.

- **Estructura del *topic*:** /plc/modules_config/module_<n>/<config>. Donde:
  - <**n**>: Número de la tarjeta.
  - <**config**>: Campo de configuración (ej. coils, discrete_inputs, holding_registers, input_registers, module_id, connected).

- **Ejemplo de topic:** Escribiendo el siguiente *topic*, el cliente puede consultar el número de coils de la tarjeta conectada actualmente en el slot 3: /plc/modules_config/module_3/coils.

5.6.2 Memoria Intermedia

La memoria intermedia en el módulo MQTT es una estructura centralizada que gestiona en tiempo real los datos de las tarjetas conectadas al PLC y su interacción con el broker MQTT. Esta memoria actúa como un **puente entre la base de datos y el broker**, permitiendo una **comunicación bidireccional asíncrona y eficiente** entre estas capas. Al igual que en el núcleo del PLC, se trata de una memoria compartida que **conecta los dos threads del módulo**, siendo, por tanto, la **sección crítica** del programa. Esto implica que su acceso está minuciosamente protegido por **mutexes**, para garantizar que el acceso concurrente a esta estructura se gestione de manera segura y evitar así conflictos o inconsistencias durante las operaciones de lectura y escritura realizadas por ambos threads.

Su propósito principal es, por una parte, permitir una **comunicación bidireccional y asíncrona** entre la base de datos y el broker. Por otra parte, minimizar el número de accesos a estos dos componentes, optimizando el rendimiento del sistema y permitiendo un flujo de datos controlado y eficiente entre ambos.

A continuación, se describen los campos que conforman esta memoria intermedia:

- **module_id:** Almacena el ID del tipo de tarjeta conectada.
- **connected:** Indica el estado de conexión del slot.
- **initilize:** Determina si la tarjeta requiere inicialización en la iteración actual.
- **num_c, num_di, num_hr, num_ir:** Cantidades configuradas de cada tipo de dato (coils, discrete inputs, holding registers, input registers) por tarjeta.
- **previous_num_c, previous_num_di, previous_num_hr, previous_num_ir:** Almacenan los valores anteriores para detectar cambios en la configuración de la tarjeta. Esto es útil para detectar el inusual caso en que una tarjeta se intercambia rápidamente por otra sin que el núcleo detecte la desconexión de la primera.
- **c, di, hr, ir:** Contienen los *value* leídos desde la base de datos.
- **c_to_update, di_to_update, hr_to_update, ir_to_update:** Identifican qué *value* ha cambiado y debe ser publicado en el broker en la iteración actual.
- **c_required, di_required, hr_required, ir_required:** Almacenan los valores enviados por los clientes MQTT al topic de escritura para actualizar los *required_value* y, eventualmente, las salidas del PLC.
- **c_required_to_update, di_required_to_update, hr_required_to_update, ir_required_to_update:** Señalan qué valores enviados por los clientes MQTT al topic de escritura han cambiado y deben ser actualizados en el *required_value* de la base de datos.
- **c_visibility, di_visibility, hr_visibility, ir_visibility:** Determinan qué *value* son visibles para los clientes MQTT.
- **c_visibility_mode, di_visibility_mode, hr_visibility_mode, ir_visibility_mode:** Definen cómo se presenta la visibilidad para cada tipo de dato.
- **c_period, di_period, hr_period, ir_period:** Especifican los intervalos con los que los *value* deben ser publicados en el bróker

- **update_write_topic:** Indica si los topics de escritura de la tarjeta deben ser actualizados en esta iteración.
- **update_config_topic:** Identifica si la configuración del slot cambió, requiriendo la actualización de sus topics de configuración en el broker.
- **data_prepared:** Garantiza que los datos estén listos para ser publicados, evitando el envío de información no inicializada o incorrecta.

Estos campos representan la información necesaria para manejar una tarjeta del PLC. **Para representar el PLC completo, se utiliza un array que contiene 8 estructuras de estas, una por cada tarjeta.**

5.6.3 Thread 1: Volcado de Datos Entre la Memoria Intermedia y el Broker

El diseño de este thread se centra en gestionar la comunicación y **sincronización entre la memoria intermedia del módulo MQTT y el broker MQTT**. Es responsable de leer mensajes desde el broker, actualizarlos en la memoria intermedia y viceversa. El flujo de ejecución del thread se detalla a continuación:

Main

- Crea un cliente para conectarse al broker MQTT.
- Conecta el cliente al broker.
- Subscribe el cliente a la raíz de los 3 tipos de topics.
- Establece una conexión con la base de datos.
- Lanza el thread 2.
- Empieza la ejecución del thread 1 (se ejecuta en el thread del main).

Inicialización

- Elimina datos residuales en los topics del broker de la ejecución anterior con la función *delete_previous_topics*.

Bucle principal

- **Lectura desde el broker:** Procesa mensajes entrantes de los topics de escritura con la función *read_mqtt* y actualiza los valores deseados en la memoria intermedia.
- **Publicación en el broker:** Recorre las tarjetas conectadas y publica las configuraciones y los *value* en los topics correspondientes con *update_topics*.
- **Ajuste dinámico del tiempo:** Ajusta el tiempo de espera del bucle en base al tiempo real transcurrido para tener un refresco de 1 Hz.

A continuación, se describe el funcionamiento de las funciones principales de este thread:

- **delete_previous_topics:** Elimina los datos almacenados en los topics de la ejecución anterior para evitar inconsistencias. Para ello, recibe mensajes pendientes en el broker relacionados con datos previos y **publica un mensaje vacío** en los topics involucrados.
- **read_mqtt:** Procesa los mensajes recibidos desde los topics de escritura y actualiza la memoria intermedia. Para ello, identifica la tarjeta y la dirección del mensaje recibido y verifica que esta tarjeta esté conectada y que la dirección sea válida para ese tipo de dato (coil, input register, etc.). Entonces, si el valor recibido es diferente

al actual, actualiza el valor deseado en la memoria intermedia y marca el topic como listo para ser publicado.

- **update_topics:** Publica configuraciones y datos actualizados en el broker.
    - Llama a *update_modules_config_topics* para publicar configuraciones (e.g., número de coils, discrete inputs, etc.).
    - Llama a *update_data_topics* para publicar los *value* de las tarjetas conectadas en función de su visibilidad y periodicidad establecidas en la base de datos.
- **update_modules_config_topics:** En caso de que el flag de actualización para ese slot esté activado:
    - Se eliminan los topics de datos relacionados con configuraciones previas del slot publicando mensajes vacíos en el broker, asegurando que los datos obsoletos de la tarjeta anterior sean eliminados.
    - Si la tarjeta está conectada, se publican los valores de configuración de la tarjeta actual; si está desconectado, se publican mensajes vacíos.
    - Se actualiza a 0 el flag que indica que se han de refrescar los valores de configuración de ese slot, para que no se realice esta operación en siguientes iteraciones
- **update_data_topics:** Consulta la visibilidad (visible u oculto), el refresco y el modo de publicación (por cambios o periódicamente) de cada *value*. Entonces, publica desde memoria intermedia los valores que cumplen con las condiciones de visibilidad y periodicidad en esa iteración en el topic data.
- **publish_topic:** Construye el nombre del topic en base a la tarjeta, dirección y tipo de valor. Por último, publica un mensaje con el valor correspondiente como payload, o un mensaje vacío si se trata de una desconexión.

5.6.4 Thread 2: Volcado de Datos Entre la Memoria Intermedia y la Base de Datos

El diseño de este thread se centra en gestionar la comunicación y **sincronización entre la memoria intermedia del módulo MQTT y la base de datos**. Es responsable de leer y actualizar la configuración y los datos de las tarjetas conectadas al PLC en la memoria intermedia, así como de volcar los valores requeridos desde esta hacia la base de datos. El flujo de ejecución del thread se detalla a continuación:

Inicializar las tarjetas en memoria intermedia

- Asigna valores iniciales a los 8 slots de la memoria intermedia (desconectado, sin valores previos, sin datos preparados, etc.).

Iniciar bucle infinito

- **Leer configuración de las tarjetas desde la base de datos:** Lee desde la base de datos la configuración actual de las tarjetas en el PLC (número de coils, ID del tipo de tarjeta, etc.) mediante la función *read_modules_config_db*.
- **Actualizar memoria intermedia:** Con los valores de configuración leídos, la función *initialize_module_data* los vuelca a memoria intermedia. Si se detecta una tarjeta desconectada, libera recursos y reestablece todos los datos de ese slot.

- **Leer datos de la tarjeta desde la base de datos:** Actualiza valores actuales (*value*, de la tabla *rtmirror*) y parámetros de visibilidad en la memoria intermedia mediante la función *read_modules_data_db*.
- **Escribir valores requeridos desde la memoria intermedia hacia la base de datos:** Actualiza solo las direcciones de los *required_value* de la base de datos que hayan cambiado en memoria intermedia (estos cambios provienen del topic write) mediante la función *write_to_db*.
- **Ajuste dinámico del tiempo:** Ajustar el tiempo de espera del bucle en base al tiempo real transcurrido para tener un refresco de 1 Hz.

A continuación, se describe el funcionamiento de las principales rutinas que intervienen en el flujo de ejecución:

- **read_modules_config_db:** Obtiene la configuración de una tarjeta conectada a un slot del PLC desde la base de datos. También verifica si el estado del slot ha sido actualizado. Basándose en esta información, ajusta y actualiza las estructuras de la memoria intermedia
- **initialize_module_data:** Libera y reasigna memoria para cada tipo de dato (coils, discrete inputs, holding registers e input registers) y para los flags de sincronización, según la configuración de la tarjeta. Luego, inicializa todas estas estructuras.
- **read_modules_data_db:** Consulta a la base de datos los *value* de un slot y su configuración de visibilidad. Entonces, se actualiza la configuración de visibilidad para cada *value* en memoria intermedia. En cuanto a los *value* leídos, estos se comparan con los existentes en la memoria intermedia, y si hay diferencias, se actualizan y marcan como listos para ser publicados al broker MQTT más adelante.
- **get_real_address:** Su función es idéntica a la versión del módulo Modbus TCP/IP. Se encarga de retornar la dirección real en la base de datos de un tipo de dato Modbus solicitado.
- **write_to_db:** Si el flag de actualización para ese slot está activado, recorre cada tipo de dato y verifica si es necesario realizar cambios para esa dirección. Para cada dirección con un valor modificado, calcula su dirección real en la base de datos con la función anteriormente mencionada *get_real_address* y ejecuta una consulta de actualización sobre el *required_value* de *rtmirror*. Finalmente, marca los valores como actualizados en la memoria intermedia para evitar redundancias.

5.7 Decisiones de Diseño Comunes a Todo el Software

En el desarrollo de todo el software del sistema, se han tomado diversas decisiones de diseño fundamentales para garantizar la **eficiencia**, **integridad** y **robustez** del sistema. Una de las decisiones clave ha sido la implementación de operaciones de escritura en la base de datos como **transacciones**, con el objetivo de asegurar la **atomicidad a nivel de slot**. Esto implica que todas las modificaciones relacionadas con un slot específico, como actualizaciones de valores de entradas, salidas y configuraciones, se ejecutan como una unidad de trabajo indivisible. En caso de que ocurra un error durante la ejecución de una operación, se realiza un **rollback**, garantizando que la base de datos no quede en un estado inconsistente. Este enfoque es esencial en un entorno con operaciones concurrentes provenientes de múltiples módulos, ya que asegura la integridad de los datos.

Además, todo el sistema implementa una **jerarquía de manejo de errores** mediante el **exclusivo uso de funciones que retornan valores enteros**. Esto permite establecer un control estructurado, donde cada función puede indicar el éxito o fracaso de su operación, facilitando la identificación y gestión de errores a lo largo de todo el flujo de ejecución. Este enfoque asegura que el sistema pueda verificar de manera constante el correcto funcionamiento de cada componente.

Por último, dada la extensión y complejidad del proyecto, desde un primer momento se decidió adoptar un enfoque de buenas prácticas de programación, centrado en reducir al mínimo la repetición de código. Esto se ha logrado mediante la creación de funciones reutilizables y estructuras comunes que permiten **agrupar la lógica compartida**, garantizando un diseño más limpio y eficiente. Este enfoque no solo mejora la legibilidad, eficiencia y mantenibilidad del código, sino que también facilita la incorporación de futuras ampliaciones o modificaciones, asegurando que el sistema sea escalable y más sencillo de depurar.

6 Implementación

6.1 Base de Datos

6.1.1 Sistema Gestor de Base de Datos y Docker

La base de datos del sistema está gestionada mediante **MariaDB**, un sistema gestor de bases de datos relacional ampliamente reconocido por su robustez y eficiencia. MariaDB fue seleccionado debido a varias razones clave que lo convierten en una solución idónea para este proyecto. En primer lugar, su compatibilidad con estándares SQL lo hace altamente versátil y fácil de integrar con otros componentes del sistema. Además, su rendimiento optimizado es especialmente valioso en dispositivos de baja potencia como la **Orange Pi**, que actúa como el núcleo de hardware en este proyecto. MariaDB es también una solución de **código abierto**, lo que permite flexibilidad para personalizar su uso sin costos de licencias, haciéndolo una opción económicamente viable y técnicamente adecuada para este proyecto.

Para asegurar un entorno de ejecución eficiente y aislado, la base de datos está **dockerizada** en la Orange Pi. Docker encapsula MariaDB junto con todas sus dependencias en un contenedor, creando un entorno virtualizado y estandarizado. Esta aproximación presenta múltiples beneficios. Por un lado, asegura la portabilidad, ya que el contenedor puede ser trasladado a otros dispositivos con facilidad (ideal para el estado actual de prototipo del PLC). Por otro lado, el aislamiento proporcionado por Docker reduce la probabilidad de conflictos con otros servicios que se ejecutan en la Orange Pi, como el broker MQTT y otros que se añadirán en el futuro, lo que mejora la estabilidad del sistema.

Además, los datos de la base de datos no se almacenan directamente dentro del contenedor de Docker, sino que se gestionan a través de un **volumen de Docker**. Este enfoque tiene varias ventajas importantes:

- **Persistencia de datos**: Incluso si el contenedor es eliminado o recreado, los datos permanecen intactos en el volumen, asegurando que no se pierda información crítica.
- **Facilidad de mantenimiento**: Los datos pueden ser respaldados, restaurados o **migrados** de manera independiente del contenedor, simplificando las tareas de administración.
- **Escalabilidad**: Los volúmenes son fácilmente escalables y pueden ser transferidos a otros sistemas en caso de que sea necesario ampliar o reconfigurar el entorno.

Un aspecto adicional que refuerza la eficiencia de esta implementación es que el contenedor de Docker que contiene MariaDB se configura para **iniciarse automáticamente al arrancar el PLC**. Esta decisión asegura que la base de datos esté disponible desde el momento en que el sistema comienza a operar, eliminando la necesidad de una intervención manual para iniciar el servicio. Esta característica es especialmente crucial en entornos donde se requiere alta disponibilidad y confiabilidad, como es el caso de este PLC.

En resumen, la implementación de la base de datos mediante MariaDB, alojada en un contenedor Docker y utilizando volúmenes para la persistencia de datos, proporciona una solución robusta, escalable y eficiente. Al integrar estas tecnologías, se logra garantizar un alto rendimiento, flexibilidad en la gestión de datos, y facilidad de despliegue y mantenimiento.

6.1.2 Creación de Tablas

A continuación, se detalla la creación de 2 de las tablas de la base de datos con código SQL, donde se reflejan las decisiones tomadas durante la etapa de diseño del sistema. Se puede apreciar que estas tablas estarán almacenadas en RAM gracias al motor de almacenamiento ENGINE.

```
-- Creates the rtmirror table
CREATE TABLE rtmirror (
    module SMALLINT UNSIGNED,
    `address` SMALLINT UNSIGNED,
    `value` SMALLINT UNSIGNED,
    required_value SMALLINT UNSIGNED,
    `timestamp` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    type_value ENUM('bit', 'half'),
    PRIMARY KEY (module, `address`, type_value)
)ENGINE = MEMORY;


-- Creates a visibility configuration table to indicate to the upper layer services which values should be published and which should not.
CREATE TABLE visibility_config (
    module SMALLINT UNSIGNED,
    `address` SMALLINT UNSIGNED,
    type_value ENUM('bit', 'half'),
    visibility ENUM('visible', 'hidden'),
    visibility_mode ENUM('periodically', 'changes'),
    refresh_rate SMALLINT UNSIGNED,
    PRIMARY KEY (module, `address`, type_value)
)ENGINE = MEMORY;
```

6.1.3 Triggers y nivel de aislamiento

Los triggers en la base de datos del PLC automatizan la actualización de información clave, garantizando la coherencia de los datos en tiempo real. En particular, están diseñados para mantener sincronizado el **número de tarjetas conectadas** al sistema en la tabla *configuration*, bajo la clave *num_modules*. Estos son los 3 triggers implementados:

- **INSERT:** Actualiza el conteo al añadir un registro en *connected_modules*.
- **DELETE:** Ajusta el conteo al eliminar un registro.
- **UPDATE:** Recalcula el número de tarjetas al eliminar o insertar tarjetas mediante la operación UPDATE.

La principal ventaja de estos triggers radica en su capacidad para automatizar la lógica de conteo de tarjetas conectadas en tiempo real directamente en la base de datos, eliminando la necesidad de añadir lógica adicional en otros módulos del sistema para mantenerlos actualizados. A continuación, se detalla la implementación del trigger encargado de actuar en el INSERT en código SQL.

```
    -- Creates the trigger to update the number of connected modules
when insert
    CREATE TRIGGER update_num_modules_insert
    AFTER INSERT ON connected_modules
    FOR EACH ROW
    BEGIN
        DECLARE num SMALLINT UNSIGNED;
        -- Gets the number of connected cards
        SELECT COUNT(*) INTO num FROM connected_modules WHERE connected
= TRUE;
        -- Inserts or updates the number of connected cards in the
configuration table
        INSERT INTO `configuration` (`key`, `value`)
        VALUES ('num_modules', num)
        ON DUPLICATE KEY UPDATE `value` = VALUES(`value`);
    END;
    //
```

Además, tal y como se estableció en el diseño de la base de datos, se configura el nivel de aislamiento global de las transacciones como **SERIALIZABLE**, garantizando que las transacciones se ejecuten de manera completamente aislada. Este nivel de aislamiento es esencial en sistemas críticos como este, donde múltiples threads y servicios interactúan con la base de datos en tiempo real.

```
    -- Sets the isolation level for the DB
    SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

6.1.4 Permisos de Lectura/Escritura para cada tipo de tarjeta

En este apartado se definen en la tabla ***permission_map*** los permisos de acceso para cada tipo de registro E/S de cada tipo de tarjeta, permitiendo que el sistema respete las capacidades físicas de cada una y garantice un control coherente y seguro a las entradas y salidas. Esta información, como se ha explicado con anterioridad, se encuentra en la microSD y es inmutable. Adicionalmente, cabe destacar que algunos tipos de tarjetas presentan más registros que los definidos en diseño. Esto se debe a que, en algunos casos, se utilizan **registros virtuales** para funcionalidades adicionales. Este es el caso de la tarjeta 8SD, que se ilustra a continuación.

```
    -- Hardcodes the permission_map table
    INSERT INTO permission_map (id_type, type_value, address,
permissions_rw)
    -- id 16: 8SD, 8 bits, 2 halfs
    VALUES (16,1,0,2), (16,1,1,2), (16,1,2,2), (16,1,3,2), (16,1,4,2),
(16,1,5,2), (16,1,6,2), (16,1,7,2), -- bits rw
(16,2,0,2), (16,2,1,2), -- halfs rw
```

6.2 Núcleo del PLC

El núcleo del PLC ha sido **implementado en su totalidad utilizando el lenguaje C puro** (junto al makefile), una elección cuidadosamente justificada por las ventajas que este lenguaje de bajo nivel ofrece. C proporciona un control directo sobre el hardware, una gestión eficiente de la memoria, y permite una alta optimización en términos de velocidad de ejecución. Estas características son cruciales en un **sistema embebido** como este, donde el rendimiento, la estabilidad y el control sobre los recursos son esenciales.

Para gestionar las diferentes tareas y funcionalidades del núcleo, se han integrado varias librerías especializadas que complementan la implementación:

- **MariaDB Connector/C**: Se utiliza el MariaDB Connector/C como librería para interactuar con la base de datos. Este conector permite realizar operaciones SQL de manera eficiente y segura desde C, garantizando una comunicación fluida entre el núcleo del PLC y la base de datos para almacenar, recuperar y sincronizar los datos de las tarjetas periféricas.
- **pthread**: La librería pthread es utilizada para manejar el Tread 1 y 2 y otros 2 secundarios más. Esto permite implementar múltiples tareas paralelas, como la comunicación con los periféricos, la actualización de la memoria intermedia y la sincronización con la base de datos. Además, se utilizan mutexes para proteger las secciones críticas (memoria intermedia) y garantizar la consistencia de los datos en este entorno multithread.
- **WiringPi**: Para interactuar con los **GPIO** de la Orange Pi, se utiliza la librería WiringPi. Esta librería simplifica el control de los pines GPIO, permitiendo configurar y gestionar entradas y salidas digitales con precisión.

En conjunto, estas decisiones técnicas aseguran que el núcleo del PLC sea altamente eficiente, fiable y adaptable a las necesidades del sistema. Su implementación en C, combinada con el uso de librerías especializadas, permite integrar de manera óptima las diferentes capas del sistema y garantizar un funcionamiento fluido en tiempo real con un rendimiento cuidadosamente pulido.

La memoria intermedia es la parte central de este componente del sistema. Por ello, es imprescindible detallar brevemente su implementación.

Se basa en un *struct* llamado *SlotType*, diseñado para representar de manera eficiente el estado y configuración de cada uno de los slots del PLC. Para representar al PLC en su totalidad, se utiliza un array de *SlotType* de tamaño *MAX_SLOT* (8). Para garantizar la integridad de estas estructuras en el entorno concurrente, se protege su acceso mediante un mutex global (*slots_mutex*), permitiendo que el thread 1 y 2 puedan acceder a ella sin provocar condiciones de carreras ni inconsistencias. A continuación, se muestra el código que define esta memoria intermedia.

```
#define MAX_SLOTS 8

typedef struct {
    unsigned char idSlot;
    unsigned char maxBits;
    unsigned char maxHalfs;

    unsigned char *bits;
    unsigned short int *halfs;

    unsigned char *required_ bits;
    unsigned short int *required_ halfs;
    unsigned char connected;    // 0 = disconnected,
1 = disconnected and modified in DB, 2 = connected, 3 = connected
and modified in DB

    // MODULE -> STRUCT -> DATABASE
    // Array of flags that indicate, based on their position, the
address in the slot that has been modified. Necessary because it is the
only way to mark modifications (NULL or negative values cannot be used to
indicate empty positions).
    unsigned char * bits_modified;
    unsigned char * halfs_modified;

    // DATABASE -> STRUCT -> MODULE
    unsigned char *required_bits_modified;
    unsigned char *required_halfs_modified;
    unsigned char delete_rtmirror;
} SlotType;

extern SlotType slots[MAX_SLOTS]; // Represents the entire PLC
extern pthread_mutex_t slots_mutex;
```

6.3 Módulo ModbusTCP

El módulo ModbusTCP **se ha implementado íntegramente en C** (junto al makefile), por las mismas razones expuestas en el núcleo del PLC: su bajo nivel permite un control preciso de los recursos, optimización del rendimiento, y una integración eficiente con el hardware y protocolos. Junto a este lenguaje, se han utilizado estas librerías y llamadas al sistema:

- **libmodbus:** Para gestionar el protocolo Modbus TCP/IP se utiliza la librería libmodbus, ampliamente conocida por su soporte para estándares como Modbus RTU y TCP.
- **poll ():** Se integró el uso de la función poll(), una llamada al sistema proporcionada por la biblioteca estándar de Linux a través de <poll.h>, para monitorizar eventos en el servidor y los clientes de forma eficiente. Esta función permite vigilar múltiples descriptores de archivo (en este caso, sockets) simultáneamente para detectar eventos, como la disponibilidad de datos para lectura o escritura. Al configurarse con un timeout de -1, poll() bloquea el hilo hasta que se detecta un evento, optimizando el consumo de recursos al **evitar un ciclo de espera activa**. Esto asegura una operación eficiente del servidor, ya que permite responder a nuevas conexiones o solicitudes sin sobrecargar la CPU.
- **MariaDB Connector/C:** Se utiliza el MariaDB Connector/C para interactuar con la base de datos desde el servidor.

Es importante mencionar que, durante la implementación, se identificó una limitación importante en libmodbus: no permite gestionar múltiples contextos Modbus de forma consistente en un entorno multithread. Debido a esta restricción, fue necesario adoptar un enfoque alternativo basado en un modelo **secuencial** para procesar las solicitudes de los clientes, utilizando una cola. A pesar de ello, el módulo conserva su capacidad **multicliente**, permitiendo la conexión de varios dispositivos al servidor de manera eficiente.

A continuación, se muestra una breve parte de código, encargada de iterar las posiciones de la cola sockets y verificar si algún cliente ha enviado una petición Modbus.

```
    for (int i = 1; i < nfds; i++){ //first socket belongs to the server
                if (fds[i].revents & POLLIN){
                    char buffer[MODBUS_TCP_MAX_ADU_LENGTH];
                    ssize_t bytes_received = recv(fds[i].fd, buffer,
sizeof(buffer), MSG_PEEK | MSG_DONTWAIT);
                    if (bytes_received == 0){ //client closed connection
                        printf("Client %d disconnected\n", i);
                        close(fds[i].fd);
                        // Remove this client from the list
                        fds[i] = fds[nfds - 1];
                        nfds--;
                        i--;} // Adjust index after removal
                    else if (bytes_received > 0){
                        printf("Handling request from client %d\n", i);
                        handle_client(ctx, fds[i].fd, i);}}}
```

6.4 Módulo MQTT

Al igual que en el módulo anterior, y por las mismas razones, el módulo MQTT se ha **implementado en su totalidad en el lenguaje C** (junto al makefile).

El broker utilizado en este módulo es un servidor **Mosquitto**, el cual está configurado para ejecutarse en un entorno **dockerizado**. Este diseño permite un despliegue aislado y portable, asegurando que el broker pueda operar de forma estable en el sistema. Además, este Docker y el servicio del broker están configurado para **iniciarse automáticamente durante el arranque del PLC**, garantizando que el sistema MQTT esté operativo desde el momento en que el PLC se enciende. Por otra parte, todos los datos relacionados con el broker se almacenan en **volúmenes Docker**, lo que asegura la persistencia de la información, incluso tras reinicios o actualizaciones del contenedor.

También, cabe destacar que el módulo MQTT se implementa como un **cliente del broker** que interactúa directamente con el sistema de topics para leerlos y modificarlos.

En cuanto a las librerías utilizadas, las más importantes son las siguientes:

- **Paho MQTT C Client Library**: Esta librería es fundamental para implementar la funcionalidad del cliente MQTT. Proporciona las herramientas necesarias para establecer conexiones con el broker Mosquitto, suscribirse a topics, publicar mensajes y mantener la conexión activa en el tiempo. Es clave para gestionar toda la comunicación basada en el protocolo MQTT desde el lenguaje C.
- **MariaDB Connector/C**: Utilizada para la interacción con la base de datos del sistema desde el lenguaje C.
- **pthread**: Empleada para gestionar los hilos de ejecución dentro del módulo MQTT. Esto permite que el thread 1 y 2 se ejecuten simultáneamente, logrando una ejecución concurrente y eficiente.

Además, para la publicación de topics del PLC, se utiliza un **QoS (Quality of Service) de nivel 0**, el cual asegura una entrega "al máximo esfuerzo" sin confirmación del receptor. Esta elección acelera significativamente la ejecución del sistema, ya que evita el sobrecoste asociado a niveles de QoS más altos (como QoS 1 o QoS 2), que implican confirmaciones y reenvíos en caso de errores. Aunque QoS 0 no garantiza la entrega del mensaje, es ideal en este contexto, donde la prioridad recae en la velocidad y el rendimiento del sistema.

Con el fin de mostrar una pequeña parte del código, a continuación se adjunta la función *publish_topic*, encargada de publicar topics en el broker.

```
void publish_topic(MQTTClient client, int module, int address, char
value_type[], int16_t empty, uint16_t value){
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    pubmsg.qos = 0;      // Quality of Service (QoS 0): NO confirmation
for fast execution
    pubmsg.retained = 1; // Retain the message
    MQTTClient_deliveryToken token;
    char topic_name[128];
    char payload[16];
    // Create the topic name
```

```
    if (address == -1) // It means the message has to be placed in a
config topic (because in this case there is no address needed so we put -
1 for convenience){
        snprintf(topic_name, sizeof(topic_name),
"/plc/modules_config/module_%d/%s", module, value_type);}
    else{
        snprintf(topic_name, sizeof(topic_name),
"/plc/data/module_%d/%s/%d", module, value_type, address);}
    // Check if module is connected or disconnected
    if (empty == 1){
        printf("MQTT: Module %d disconnected\n", module);
        pubmsg.payload = NULL;
        pubmsg.payloadlen = 0;
        printf("DATA TOPIC: %s DELETED\n", topic_name);}
    else{
        printf("MQTT: Module %d connected\n", module);
        snprintf(payload, sizeof(payload), "%d", value); // Convert value
to string
        pubmsg.payload = payload;
        pubmsg.payloadlen = strlen(payload);}
    // Publish the message
    MQTTClient_publishMessage(client, topic_name, &pubmsg, &token);
    int rc = MQTTClient_waitForCompletion(client, token, 10000); // 10
segundos de espera
    if (rc == MQTTCLIENT_SUCCESS){
        printf("MQTT: Topic %s published or deleted successfully\n",
topic_name);}
    else{
        printf("MQTT: Error publishing or deleting %s\n", rc,
topic_name);}}
```

7 Evaluación

7.1 Casos de Prueba y Resultados

A continuación, se presentan diferentes tablas que recogen las pruebas más importantes y representativas realizadas a cada componente del sistema, con la finalidad de verificar el correcto funcionamiento de cada uno de ellos.

7.1.1 Pruebas Unitarias: Núcleo del PLC

En primer lugar, en la *Tabla 7.1* se muestran las pruebas más relevantes realizadas al núcleo del PLC.

Número de Prueba	Prueba Realizada	Resultado Esperado	Resultado Obtenido
1	Activar un interruptor conectado a una entrada digital de una tarjeta.	El *value* de la dirección correspondiente de *rtmirror* pasa a 1 en la base de datos inmediatamente.	El mismo que el resultado esperado.
2	Girar un potenciómetro conectado a una entrada analógica.	El *value* de la dirección correspondiente de *rtmirror* modula en tiempo real su valor.	El mismo que el resultado esperado.
3	Se conecta una tarjeta en un slot vacío.	La tabla *connected_modules* actualiza los campos de configuración para ese tipo de tarjeta.	El mismo que el resultado esperado.
4	Se desconecta una tarjeta.	La base de datos borra los datos da la tarjeta recién desconectada.	El mismo que el resultado esperado.
5	Se inicia el PLC con slots vacíos.	El PLC inicia correctamente sin datos de otras ejecuciones.	El mismo que el resultado esperado.
6	Se inicia el PLC, se conectan tarjetas, se apaga el PLC, se cambia la disposición de las tarjetas y se vuelve a encender.	El PLC inicia correctamente sin datos de otras ejecuciones y detecta las nuevas posiciones de las tarjetas.	El mismo que el resultado esperado.
7	Se desconecta una tarjeta y se conecta otra de otro tipo.	La base de datos actualiza la nueva configuración de la tarjeta recién conectada	El mismo que el resultado esperado.
8	Se desconecta una tarjeta y, rápidamente, se conecta otra de otro tipo, sin que el núcleo	El núcleo, a pesar de no haber detectado la desconexión, detecta que los IDs han cambiado, por	El mismo que el resultado esperado.

	haya detectado la desconexión de la primera.	lo que responde correctamente y actualiza en la base de datos la nueva configuración de la tarjeta recién conectada	
9	Se desconecta una tarjeta y, rápidamente, se conecta otra **del mismo tipo**, sin que el núcleo haya detectado la desconexión de la primera.	El núcleo es capaz de detectar el cambio de tarjetas, borra de la base de datos los datos relacionados con la anterior y carga la configuración de la actual.	El núcleo no es capaz de detectar el cambio de tarjetas, por lo que en la base de datos se conservarán todos los datos de la anterior tarjeta.
10	Activar un *required_value* de tipo *bit* de la tabla *rtmirror* manualmente desde la base de datos.	Este cambio se refleja en la activación inmediata de la salida digital asociada. Por ejemplo, se enciende el led correspondiente.	El mismo que el resultado esperado.
11	Activar un *required_value* de tipo *bit* de la tabla *rtmirror* manualmente desde la base de datos escribiendo un valor mayor que 1.	El núcleo detecta que el valor es mayor que 1 para un registro de tipo *bit* y envía un 1 por SPI, manteniendo la consistencia. Se enciende el led correspondiente.	El mismo que el resultado esperado.
12	Modificar un *required_value* de tipo *half* de la tabla *rtmirror* manualmente.	Este cambio se refleja en la activación inmediata de la salida analógica asociada. Por ejemplo, se mueve la varilla de un voltímetro.	El mismo que el resultado esperado.

Tabla 7.1: Conjunto de pruebas realizadas para testear el correcto funcionamiento del núcleo del PLC

7.1.2 Pruebas Unitarias: Módulo Modbus TCP/IP

Seguidamente, en la *Tabla 7.2* se muestran las pruebas realizadas al módulo Modbus TCP/IP.

Número de Prueba	Prueba Realizada	Resultado Esperado	Resultado Obtenido
1	Conectar un cliente Modbus.	El servidor acepta la conexión y pone el socket del cliente a la cola.	El mismo que el resultado esperado.
2	Conectar 3 clientes modbus.	El servidor acepta las 3 conexiones y pone los 3 sockets a la cola.	El mismo que el resultado esperado.
3	Realizar todos los tipos de consultas Modbus manejadas por el servidor.	Ante cada una de las funciones permitidas, el servidor responde correctamente.	El mismo que el resultado esperado.
4	Realizar una consulta de escritura o lectura a una tarjeta desconectada.	El servidor detecta que la tarjeta no está conectada, rechaza la petición Modbus y lanza un mensaje de advertencia sin detener el flujo de ejecución.	El mismo que el resultado esperado.
5	Realizar una consulta de escritura o lectura a una dirección de tipo de dato inexistente de una tarjeta conectada.	El servidor detecta que esa dirección no existe para ese tipo de dato, rechaza la petición Modbus y lanza un mensaje de advertencia sin detener el flujo de ejecución.	El mismo que el resultado esperado.
6	Realizar consultas en bucle desde varios clientes simultáneamente.	El servidor responde a todas las consultas de los 3 clientes sin mermar el rendimiento.	El mismo que el resultado esperado.
7	Realizar consultas en bucle desde varios clientes simultáneamente e ir desconectando los clientes.	Mientras responde a los clientes, detecta las desconexiones y reorganiza la cola de sockets.	El mismo que el resultado esperado.
8	Realizar una consulta de una función Modbus no manejada por el servidor.	El servidor detecta que ese código de función no lo maneja, lanza un mensaje de advertencia y rechaza la solicitud, sin detener el flujo de ejecución.	El mismo que el resultado esperado.

Tabla 7.2: Conjunto de pruebas realizadas para testear el correcto funcionamiento del módulo Modbus TCP/IP

7.1.3 Pruebas Unitarias: Módulo MQTT

Por último, en la *Tabla 7.3* se muestran las pruebas realizadas a módulo MQTT.

Número de Prueba	Prueba Realizada	Resultado Esperado	Resultado Obtenido
1	Escribir un valor en el topic write desde un cliente MQTT.	El valor escrito se muestra en el *required_value* correspondiente de la base de datos.	El mismo que el resultado esperado.
2	Escribir el topic write incorrectamente.	El módulo detecta el error y no realiza ninguna operación, más que generar un mensaje de error sin detener el flujo de ejecución.	El mismo que el resultado esperado.
3	Escribir el topic write con una ruta de una tarjeta desconectada o inexistente.	El módulo detecta que no hay una tarjeta conectada en ese slot y lanza un mensaje de error sin detener el flujo de ejecución.	El mismo que el resultado esperado.
4	Escribir en el topic write un payload que no sea entero positivo.	El módulo lo detecta y lanza un mensaje de error, sin detener el flujo de ejecución.	El mismo que el resultado esperado.
5	Escribir el topic write con una dirección de un tipo de dato que no existe.	El módulo detecta que esta posición no existe y lanza un mensaje de error sin detener el flujo de ejecución.	El mismo que el resultado esperado.
6	Consultar valores en el topic modules_config.	El módulo muestra en tiempo real el estado de conexión de los slots del PLC, así como la configuración de cada tarjeta conectada.	El mismo que el resultado esperado.
7	Consultar valores en el topic data.	El módulo muestra en tiempo real el estado de todos los *value* de la base de datos.	El mismo que el resultado esperado.

Tabla 7.3: Conjunto de pruebas realizadas para testear el correcto funcionamiento del módulo MQTT

7.1.4 Pruebas de Integración

Para verificar el correcto funcionamiento global del sistema, se han realizado diversas pruebas. Una de las más representativas, que permite comprobar la integración completa del sistema, es la siguiente:

Desde un cliente MQTT, se publica el valor 20000 en el topic: /plc/write/module_2/holding_registers/3. Al mismo tiempo, se lanza una petición recursiva por parte del cliente Modbus TCP/IP n.º 2 para leer ese mismo holding register. El resultado esperado es que, en pocos milisegundos, el cliente Modbus lea el valor 20000 en ese holding register. Tras realizar la prueba, el resultado final coincide con lo esperado.

Sin embargo, detrás de este proceso aparentemente simple, han ocurrido múltiples acciones coordinadas. A continuación, y a través de la *Figura 7.1*, se describe en detalle el recorrido que ha realizado el valor 20000, desde su publicación en el cliente MQTT hasta su lectura por el cliente Modbus.

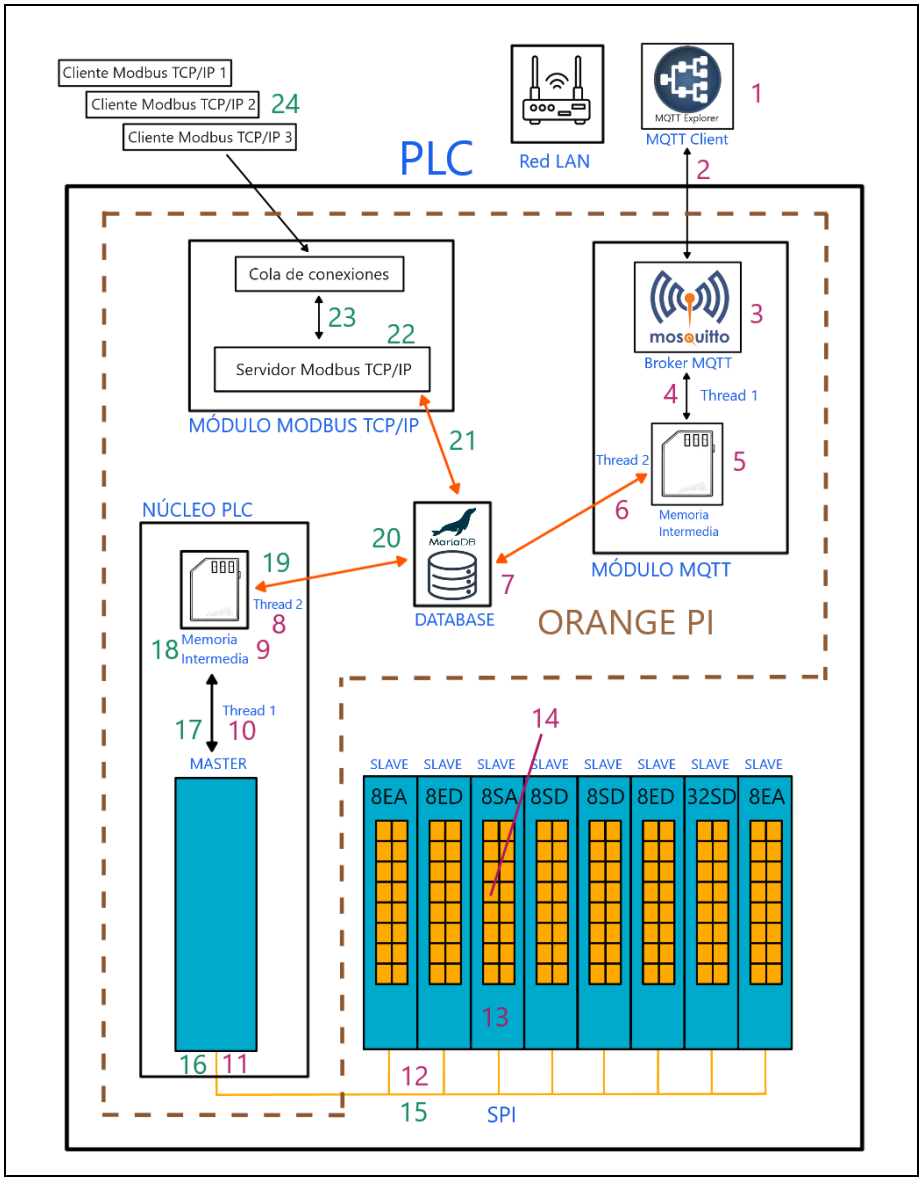

Figura 7.1: Recorrido de un valor publicado por un cliente MQTT hasta llegar a un cliente Modbus

En la *Figura 7.1* se numeran todos los pasos que realiza el valor 20000 antes de llegar a su destino. Este es el recorrido:

1. El cliente MQTT escribe el topic */plc/write/module_2/holding_registers/3* con el payload 20000.

2. El cliente MQTT publica este topic con el payload 20000 en el servidor Mosquitto (broker) dockerizado.

3. El bróker recibe el topic y lo envía al cliente MQTT del PLC.

4. El thread 1 recibe el payload desde el topic */plc/write/module_2/holding_registers/3*, al cual estaba suscrito, detecta que el valor ha cambiado con respecto a la memoria intermedia y lo vuelca.

5. La memoria intermedia almacena el nuevo valor.

6. El thread 2 detecta que ese valor es nuevo y lo envía a la base de datos.

7. La base de datos actualiza el valor 20000 en el *required_value* correspondiente de *rtmirror*.

8. El thread 2 del núcleo del PLC lee el *required_value* de la base de datos, detecta que ha cambiado y lo envía a memoria intermedia.

9. La memoria intermedia almacena el valor 20000.

10. El thread 1 detecta que ese valor es nuevo y lo envía a través del maestro SPI.

11. El valor sale de la Orange Pi.

12. El valor viaja a través de los buses que conectan la Orange Pi con las tarjetas periféricas.

13. El valor llega a la tarjeta periférica con dirección 2 de esclavo SPI.

14. La tarjeta procesa este nuevo valor y actualiza el registro *half* asociado a la salida analógica número 3. El valor 20000 sale por el GPIO de salida.

15. El thread 1, en iteraciones posteriores, hace una lectura del registro asociado y toma el valor por SPI para comprobar si es diferente.

16. El valor 20000 entra en la Orange Pi.

17. El thread 1 detecta que ese valor es nuevo y lo vuelca a memoria intermedia.

18. La memoria intermedia almacena el valor 20000.

19. El thread 2 detecta que ese valor es nuevo y lo envía a la base de datos.

20. La base de datos guarda el valor 20000 en el *value* correspondiente de la tabla *rtmirror*.

21. El servidor Modbus TCP/IP del módulo, cada segundo, está haciendo una consulta al registro *value* que se acaba de actualizar. Ahora que se ha actualizado, selecciona el 20000 y lo descarga.

22. El servidor actualiza el valor en el mapa temporal de tipos de datos Modbus.

23. El servidor empaqueta el valor siguiendo el protocolo Modbus TCP/IP y envía el paquete al cliente 2 de la cola de conexiones (cola de sockets).

24. El cliente 2 recibe el paquete de datos, lo desempaqueta y acaba obteniendo el valor 20000.

Si bien este recorrido puede asemejarse muy largo, tiene una explicación. El hecho de no acortarlo, actualizando el *value* en la base de datos en pasos anteriores, sirve para asegurar que en los *value* de *rtmirror* **solamente se tengan valores que ya estén en las tarjetas**. De no ser así, podría darse el caso en que haya un error en la transmisión SPI, que la tarjeta no actualice realmente el valor y que ya se haya actualizado el *value* en la base de datos, ocasionando una inconsistencia.

Además de esta prueba de integración, se han realizado diferentes pruebas para evaluar el rendimiento global del sistema. Entre ellas, se destaca la medición del uso de la CPU mientras el sistema opera con todos los módulos activos y en pleno funcionamiento. Para esta evaluación se utilizó la herramienta **top** en el sistema operativo Linux, que permitió monitorizar el porcentaje de uso de CPU en tiempo real.

Los resultados más representativos de estas pruebas indican que, incluso en escenarios de máxima carga, el porcentaje de CPU utilizado no supera el **15%**. Este dato refleja la eficiencia del diseño y la implementación del sistema, confirmando que puede manejar de forma holgada las operaciones previstas sin comprometer la estabilidad o el rendimiento global del hardware.

8 Evaluación de Costes

En lo relativo a los **costes de personal**, el desarrollo de este proyecto ha requerido la dedicación de un programador a tiempo completo (quien ha realizado este Trabajo de Fin de Grado), encargado de la implementación del núcleo, la base de datos, los módulos asociados y la integración de todas las partes del sistema. Además, se ha contado con la colaboración de una empresa externa, Ibercomp, especializada en la fabricación de hardware, que se encargó de diseñar y producir las placas periféricas, así como la comunicación SPI entre tarjetas y la Orange Pi.

En cuanto a la **evaluación de los costes materiales**, en total son poca cantidad, pues este proyecto se centra en el software. Concretamente, el proyecto ha necesitado una **Orange Pi Zero 3** como núcleo central de procesamiento, junto con **ocho tarjetas periféricas más una** donde conectar la Orange Pi, una **placa de conexión** donde conectar todas las tarjetas, y componentes adicionales (voltímetro, potenciómetro, interruptores, cables, fuente de alimentación, etc.). Además de una carcasa personalizada para alojar todos los componentes de manera segura y ordenada, diseñada y fabricada por la propia empresa.

9 Legislació y Protección de Datos

En el contexto de este Trabajo de Fin de Grado (TFG), la aplicación de normativas relacionadas con la protección de datos, como el **Reglamento General de Protección de Datos** (RGPD) de la Unión Europea, no resulta relevante. Esto se debe a que el proyecto no maneja datos personales ni información sensible de individuos.

El alcance del trabajo se limita al desarrollo de un sistema técnico centrado en la interacción entre hardware, bases de datos y protocolos de comunicación. Todos los datos gestionados en el proyecto corresponden exclusivamente a registros técnicos relacionados con el funcionamiento del sistema, como estados de sensores, valores de actuadores y configuraciones del PLC. Estos datos no están vinculados a personas físicas ni permiten su identificación directa o indirecta, por lo menos, en estas etapas tempranas de desarrollo en las que se encuentra el PLC.

Por lo tanto, no entra en el ámbito de aplicación de las legislaciones específicas de protección de datos personales.

10 Implicaciones Éticas, de igualdad y Medioambientales

En este Trabajo de Fin de Grado, todas las tecnologías empleadas son de código abierto, lo que garantiza el acceso libre y transparente a los recursos utilizados, fomentando la colaboración y la innovación en la comunidad tecnológica. Además, se pretende que una parte del software desarrollado también sea de código abierto, contribuyendo al ecosistema de software libre y promoviendo la reutilización y mejora de las soluciones propuestas por otros profesionales o investigadores. Este compromiso refleja una ética de trabajo basada en la transparencia, la accesibilidad y el respeto por las normativas de propiedad intelectual.

En cuanto a las implicaciones de igualdad, este proyecto no presenta implicaciones directas relacionadas con la igualdad, ya que se centra en el desarrollo técnico de un sistema de automatización industrial.

Finalmente, relacionado con las implicaciones medioambientales, se ha priorizado la eficiencia en el uso de recursos. Por una parte, se favorece el bajo consumo energético gracias al diseño eficiente del software. Por otra parte, el hardware ha sido diseñado de manera modular, lo que permite que, en caso de fallo o rotura de una tarjeta periférica, el resto del PLC continúe funcionando correctamente. Esta característica no solo mejora la sostenibilidad al evitar reemplazos innecesarios del sistema completo, sino que también prolonga la vida útil del equipo, minimizando el impacto ambiental del proyecto y alineándose con los principios de sostenibilidad y economía circular.

11 Planificación Temporal

A continuación, en la *Figura 11.1*, se presenta la planificación temporal seguida a lo largo del desarrollo del proyecto, mostrando de manera estructurada las actividades realizadas semana a semana en un diagrama de Gantt.

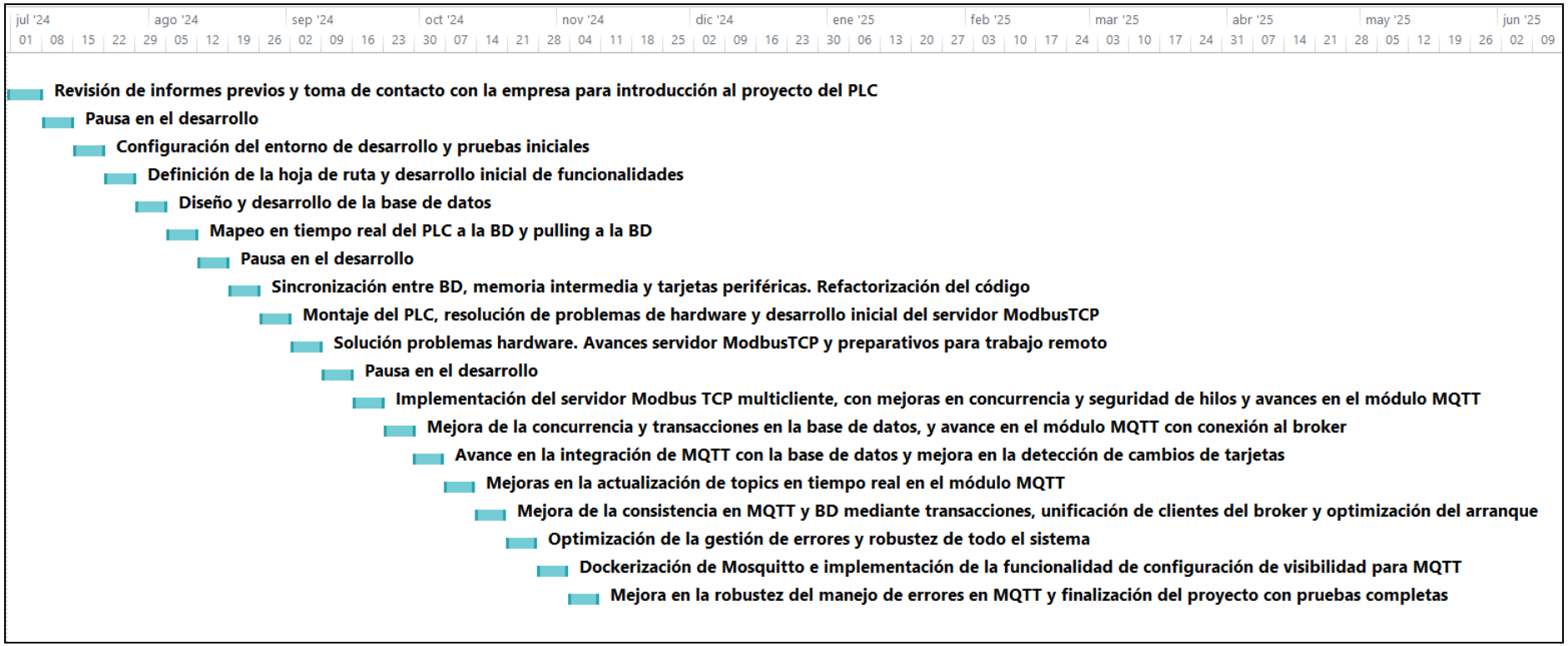

Figura 11.1: Diagrama de Gantt. Resumen de tareas realizadas semanalmente

12 Estado Actual y Trabajo a Futuro del Proyecto

El contenido presentado en este documento recoge el desarrollo del proyecto hasta mediados de noviembre de 2024. Sin embargo, **el trabajo no se ha detenido** y, a día de hoy, se continúa avanzando con diversas mejoras y nuevas funcionalidades (en la *Figura 12.1* se puede ver el estado actual del PLC).

Actualmente, se está llevando a cabo la traducción de todo el núcleo del PLC de **C a C++**. Esta decisión responde a la necesidad de mejorar la mantenibilidad y legibilidad del código, aspectos fundamentales en un sistema diseñado para escalar y evolucionar. El uso del paradigma de programación orientada a objetos (POO) permite dividir de forma clara la lógica del núcleo en distintas clases, separando, por ejemplo, el control del bus de los diferentes protocolos implementados sobre este. Además, permite el uso de una interfaz común que facilita la implementación de diferentes protocolos de manera modular y eficiente. Este cambio no solo simplifica la estructura del núcleo, sino que también mejora su extensibilidad y adaptabilidad a futuros requisitos.

Además, en colaboración con Ibercomp, la empresa encargada del diseño de las tarjetas periféricas, se están desarrollando funcionalidades críticas para garantizar la seguridad en entornos industriales. Entre ellas se incluyen:

- **Registros de estado seguro:** Estos registros permitirán almacenar valores predeterminados que serán cargados automáticamente en las tarjetas periféricas en caso de pérdida de comunicación con el núcleo del PLC. Esto asegura que los dispositivos conectados mantendrán un estado seguro hasta que se recupere la comunicación, minimizando riesgos operativos.
- **Watchdog:** Se está implementando un sistema de watchdog para supervisar continuamente el correcto funcionamiento del PLC. Este mecanismo detectará fallos críticos en el sistema y ejecutará acciones correctivas, como reiniciar el núcleo o los módulos, asegurando la continuidad operativa.

Paralelamente, se está trabajando en compilar un **kernel en tiempo real (RT)** compatible con la Orange Pi Zero 3. Este kernel proporcionará un **scheduler completamente preemptivo**, optimizado para minimizar las latencias de ejecución de tareas. Esto es esencial para sistemas de tiempo real, donde la capacidad de ejecutar tareas con latencias mínimas y predecibles es crucial para cumplir con los requisitos de control industrial, como la sincronización precisa de procesos o la gestión de eventos críticos.

Asimismo, se está abordando el problema detectado en la **prueba 9** de la *Tabla 7.1***.** Para resolverlo, se está trabajando en la implementación de **UUIDs (Identificadores Universales Únicos)**. Esto garantizará que cada tarjeta del sistema tenga un identificador inequívoco, evitando conflictos al trabajar con tarjetas del mismo tipo.

También, de cara al futuro, el proyecto contempla la incorporación de nuevos módulos para aumentar su compatibilidad con estándares ampliamente utilizados en la industria. Entre estos módulos destacan:

- **OPC UA:** Protocolo de comunicación estándar para la integración de sistemas industriales y la interoperabilidad entre dispositivos.
- **API REST:** Interfaz de programación que permitirá integrar el PLC con sistemas externos de forma flexible y escalable.

Por último, se plantea desarrollar una **capa gráfica**, que integre todo el sistema, mediante tecnologías como Node-RED[8] o similares, diseñada para facilitar la creación y programación de tareas mediante flujos visuales. Este entorno permitirá a personal no técnico y sin conocimientos avanzados de programación interactuar con el sistema de manera intuitiva, simplificando el diseño de procesos complejos. La capa gráfica se encargará de **abstraer** las complejidades del hardware y los protocolos subyacentes, proporcionando una interfaz visual amigable que integre todas las funcionalidades del sistema. Con este enfoque, se busca democratizar el uso del PLC, haciéndolo accesible para un público más amplio y fomentando la productividad en entornos industriales.

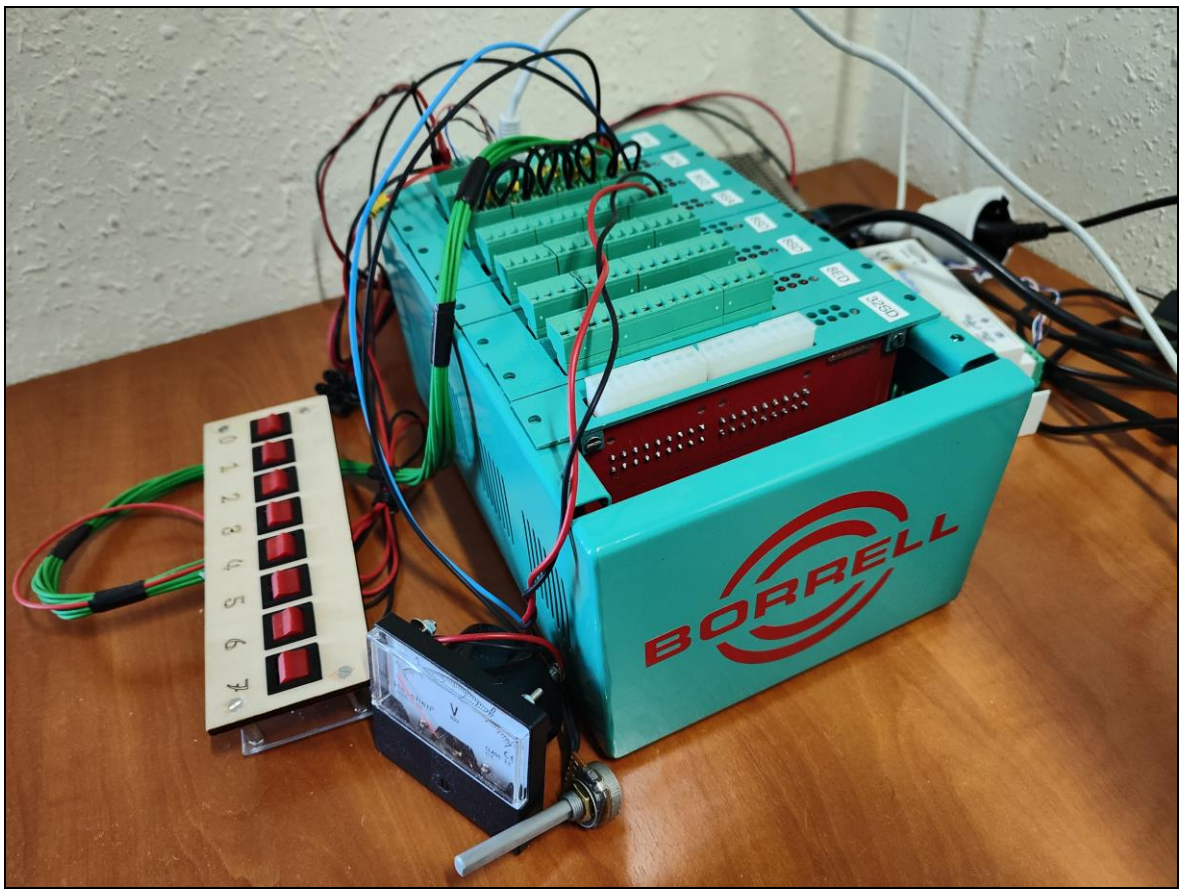

Figura 12.1: PLC en su estado actual

8 Node-RED: Entorno de desarrollo basado en flujos visuales que permite conectar dispositivos, APIs y servicios mediante nodos preconfigurados, facilitando la creación de aplicaciones y procesos de automatización sin necesidad de programación avanzada.

13 Conclusiones

En este Trabajo de Fin de Grado se ha desarrollado un controlador lógico programable modular orientado a la Industria 4.0, cumpliendo con todos los requisitos funcionales y no funcionales definidos. El sistema integra tecnologías modernas como Docker para el despliegue de servicios, protocolos de comunicación estándar como MQTT y Modbus TCP/IP, y una arquitectura modular que garantiza su adaptabilidad y escalabilidad.

El sistema, tras la realización de pruebas exhaustivas, ha quedado en una situación de alta funcionalidad y preparado para futuras expansiones. Actualmente, el proyecto está en proceso de evolución, con la traducción del núcleo a C++ para mejorar su mantenibilidad y escalabilidad, la implementación de funciones de seguridad industrial y el desarrollo de un kernel en tiempo real para garantizar la latencia mínima en sistemas críticos.

De esta manera, este proyecto no solo representa un logro significativo al cumplir con los objetivos iniciales, sino que también se posiciona como una base sólida para futuros desarrollos en el ámbito de la Industria 4.0. Con un enfoque en la innovación, la adaptabilidad y la escalabilidad, se ha sentado el fundamento para una solución tecnológica moderna y versátil, con el objetivo de evolucionar y consolidarse como un producto profesional competitivo en el mercado industrial.

14 Valoración

A lo largo de este Trabajo de Fin de Grado, se han cumplido con éxito todos los requisitos funcionales y no funcionales establecidos al inicio del proyecto. Este logro no solo representa la culminación de un esfuerzo constante, sino que también me llena de satisfacción al haber materializado los objetivos planteados desde el principio. Además, este proceso ha tenido un impacto significativo tanto en lo personal como en lo académico.

Desde un enfoque técnico, este proyecto ha sido una oportunidad excepcional para aplicar conceptos que abarcan la mayoría de las asignaturas cursadas, incluyendo campos como programación, bases de datos, sistemas operativos, redes y mucho más. Este carácter multidisciplinar me ha permitido consolidar aprendizajes previos y adquirir nuevas competencias, conectando y complementando diferentes áreas de la informática. Este proceso no solo ha fortalecido mis habilidades técnicas, sino que también me ha ayudado a entender cómo estos elementos interactúan en un sistema real.

Simultáneamente, este proyecto me ha dado la oportunidad de sumergirme en el mundo empresarial, colaborando con profesionales de la industria y enfrentándome a situaciones propias de un entorno laboral real. Gracias a esta experiencia, he podido fortalecer habilidades interpersonales clave, como la comunicación efectiva, el trabajo en equipo y la adaptabilidad. Estas llamadas *soft skills* han sido un complemento invaluable para mi desarrollo profesional, dándome una perspectiva más completa de lo que significa trabajar en proyectos reales y multidimensionales.

Por último, este proyecto ha sido un reto que me ha permitido demostrarme a mí mismo de lo que soy capaz. Ha sido un proceso en el que una idea inicial se transformó en un sistema funcional y bien estructurado, algo que me inspira confianza y satisfacción. Afrontar y superar desafíos complejos no solo ha fortalecido mis habilidades, sino que también me ha motivado profundamente a seguir creciendo y enfrentando nuevos retos cada vez más ambiciosos.

En definitiva, este Trabajo de Fin de Grado ha representado mucho más que un logro académico; ha sido un hito en mi desarrollo personal y profesional. Ha sentado las bases para una trayectoria futura sólida, siendo una experiencia que llevaré conmigo como un referente de lo que puedo lograr con esfuerzo, dedicación y pasión.

15 Recursos Utilizados

Referencias

[1] Manufactura Latam. (2024, 29 julio). Evolución de la automatización industrial. *Manufactura Latam*. https://www.manufactura-latam.com/es/noticias/evolucion-de-la-automatizacion-industrial

[2] ITI. (2023, 31 octubre). *GEMELOS DIGITALES en la transición a la Industria 4.0*. https://www.iti.es/proyectosidi/proyecto-gemelos-digitales-industria-4-0/

[3] *Serial Peripheral Interface (SPI) - SparkFun Learn*. (s. f.). https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

[4] Logicbus, & Logicbus. (2024, 30 noviembre). Protocolos de comunicación: MODBUS TCP/IP. *Blog Logicbus*. https://www.logicbus.com.mx/blog/modbus-tcp-ip/

[5] CoLtd, C. E. E. T. (2023, 23 octubre). *Explicación detallada del protocolo MQTT*. Chengdu Ebyte Electronic Technology Co.,Ltd. https://www.es-ebyte.com/news/488

[6] *BORRELL | Maquinaria para Almendras, Frutos Secos y Semillas - Home*. (s. f.). https://www.jborrell.es/

[7] IBERCOMP. (2024, 11 septiembre). *INICIO - IBERCOMP*. https://ibercomp.com/