UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Especificación de requisitos de prueba de fábrica para controladores de automatización de edificios

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero Mecatrónico

P R E S E N T A

Alexander Avilez Bahena

ASESOR DE INFORME

M.I. Ulises Martín Peñuelas Rivas

Ciudad Universitaria, Cd. Mx., 2024

Índice

1 Introducción

La Prueba Final de Fábrica (FFT) es un proceso necesario para la producción de un controlador para automatización de edificios. Esta prueba se realiza después del ensamblaje del controlador y tiene como objetivo fundamental validar tanto las funcionalidades como la fabricación del producto. Durante la prueba, se verifica que el controlador cumple con todas las especificaciones y requisitos de diseño. Estas pruebas abarcan desde la verificación de la integridad física del dispositivo hasta la evaluación de su desempeño en situaciones simuladas de operación.

Además de garantizar la calidad del producto final, la FFT también desempeña un papel importante en la identificación temprana de posibles problemas de fabricación o diseño. Esto permite tomar medidas correctivas de manera oportuna y garantizar que solo los productos de alta calidad lleguen al cliente final.

En este contexto, es importante comprender la naturaleza de los sistemas embebidos, ya que forman la base de los controladores utilizados en la automatización de edificios. Un sistema embebido es un sistema computarizado diseñado específicamente para una aplicación particular. Debido a que su misión es más limitada que la de un ordenador de propósito general, un sistema embebido tiene menos soporte para funciones no relacionadas con la ejecución de la aplicación. El hardware a menudo tiene restricciones: por ejemplo, una CPU que funciona más lentamente para ahorrar energía, un sistema que usa menos memoria para reducir costos de fabricación y procesadores que vienen solo en ciertas velocidades o que soportan un subconjunto de periféricos [1]. Estos sistemas están diseñados para realizar una tarea específica, optimizando el uso de recursos, ciclos de procesamiento o la velocidad del procesador, así como optimizando la duración de la batería (o el ahorro de energía) y uso de periféricos del procesador [2].

Comprender y verificar correctamente los sistemas embebidos es esencial porque estos sistemas están presentes en una amplia variedad de dispositivos y aplicaciones, desde electrodomésticos, dispositivos médicos hasta automóviles y equipos industriales. La calidad y fiabilidad de estos sistemas determinan el rendimiento global del producto final [3]. Además, estos deben ser diseñados y probados para asegurar que cumplen con todas las especificaciones de seguridad y eficiencia [4].

2 Objetivos

Los objetivos específicos son los siguientes:

- Describir el proceso de diseño y desarrollo de la FFT.
- Explicar la implementación de la FFT para Controlador Optimizador Unitario.
- Analizar los resultados obtenidos de la implementación de la FFT para evaluar su efectividad en la detección de errores en la producción de los Controladores.

Para realizar los objetivos anteriores, se utilizaron los conocimientos adquiridos en la formación como ingeniero mecatrónico, incluyendo programación en C/C++, ensamblador, conocimientos de sistemas operativos en tiempo real, arquitectura de microcontroladores, electrónica incluyendo circuitos, protocolos como SPI, UART, I2C, Ethernet, BLE (Bluetooth Low Energy), así como sistemas analógicos y digitales.

La FFT es una herramienta importante para garantizar la calidad del Controlador Optimizador Unitario. Los resultados de la implementación de esta prueba muestran que la prueba es efectiva para detectar errores en el hardware y el *firmware* del dispositivo.

En la sección del cuerpo del trabajo, se detallará el proceso de la FFT, así como los resultados de la implementación y las mejoras que realicé para mejorar los tiempos de la prueba.

2.1 Propósito de una prueba en fábrica

La especificación de requisitos de prueba de fábrica describe el diseño de software y hardware del banco de pruebas para la prueba final de fábrica del Controlador Unitario. Esta documentación detalla los componentes y funciones específicas que deben ser probadas, así como los procedimientos y criterios de aceptación para garantizar que el controlador cumple con los estándares de calidad y funcionamiento

requeridos. Además, incluye información sobre la configuración del entorno de prueba, los equipos necesarios y las interfaces de comunicación con el controlador.

2.2 Descripción de la empresa

2.2.1 Breve historia de la empresa

Es una empresa global de tecnología y manufactura con una amplia gama de productos y servicios que incluyen automatización, control industrial, seguridad, protección, transporte, defensa, y salud. La empresa tiene más de 100,000 empleados en todo el mundo y sus productos y servicios se utilizan en una amplia gama de industrias, desde la fabricación hasta la atención médica [5].
Se compone de varias unidades de negocio y segmentos. Algunos de las unidades de negocio clave incluyen [6]:

- *Automation and Control Solutions*
- *Buildings Automation*
- *Performance Materials and Technologies*
- *Transportation Systems and Services*
- *Aerospace*

El área en la que trabajo es en BA, donde se desarrolla, fabrica sistemas de automatización y control para edificios, que controlan el clima, la iluminación, la seguridad y otros aspectos.

Es una empresa multinacional estadounidense de tecnología con sede en Charlotte, Carolina del Norte [7]. La empresa fue fundada en 1885 por Albert Butz, quien inventó el regulador de horno y alarma. En 1893, Butz vendió su empresa a Electric Heat Regulator Co., que fue adquirida por W. R. Sweatt en 1898. Sweatt renombró la empresa como Minneapolis Heat Regulator Company y expandió su línea de productos. La empresa se expandió rápidamente en la década de 1920, y se diversificó en otros campos, como la aeroespacial, la automatización y el control, y la seguridad. En 1926, creó el primer sistema de control de temperatura para aviones. En la década de 1930, la empresa también se diversificó a la industria de la automatización y el control. En 1969, dio soporte a la misión de Apolo 11 para el sistema de Control y Estabilización. En 1999, la empresa expandió sus operaciones a diferentes industrias como la aviación, automotriz, química y más industrias [8].

Presente

Hoy es una empresa global con operaciones en varios países. La empresa ofrece una amplia gama de productos y servicios, en las áreas de negocio previamente mencionadas.

2.3 Descripción del puesto de trabajo

Mi puesto es de Desarrollador de Software de Sistemas Embebidos, el área donde trabajo está enfocada en la creación de controladores para automatizar y controlar edificios; mi trabajo principal se centra en el desarrollo de *firmware* y principalmente garantizar que estos controladores funcionen correctamente. Esto implica la creación de software y pruebas de validación, además de desarrollar código que se utiliza para obtener las certificaciones de producto necesarias, para así cumplir con los estándares requeridos para poder comercializarlos en diferentes países.

En la parte de desarrollo y pruebas de validación, soy responsable de desarrollar el *firmware* para la prueba de FFT que involucra la colaboración con el equipo de hardware, fábrica y el arquitecto del proyecto, para probar todas las funcionalidades del controlador antes de poder entregarse al cliente final. Esto incluye verificar que todos los componentes de hardware funcionen según lo previsto y que el ensamble cumplió con las especificaciones establecidas. Esta labor requiere conocimiento de la interacción entre hardware y software, así como habilidades para el diseño de pruebas efectivas, buscando el menor tiempo posible para realizarla, pero sin reducir la calidad de ésta.

Adicionalmente, en colaboración con el equipo de hardware, aseguro la compatibilidad entre el software y nuevos componentes. Esto incluye desarrollar un código específico para poder probar nuevos circuitos integrados, validar nuevos diseños de tarjetas y drivers para estos nuevos componentes.

Además, desarrollo *firmware* para realizar diferentes certificaciones de producto, como pruebas de compatibilidad electromagnética y radio frecuencia para tecnologías como BLE, asegurando que los productos cumplan con los estándares de seguridad y calidad requeridos para poder venderse alrededor del mundo.

El controlador está desarrollado con el software Niagara 4, el cual facilita la gestión eficiente de edificios al ser controlado exclusivamente mediante este programa. Admite protocolos de comunicación estándar de la industria, como BACnet IP, BACnet T1L y BACnet MS/TP, lo que permite una fácil conectividad con otros dispositivos. Además, incluye protocolos de integración como Sylk y Modbus para brindar mayor versatilidad. Asimismo, cuenta con UIO, relés de potencia y SSR para adaptarse a diversas necesidades de control en la gestión de edificios [9].

Estos controladores ofrecen una variedad de opciones de comunicación, con un enfoque principal en torno a un protocolo llamado BACnet, lo que permite la conectividad con la plataforma Niagara 4. Esta integración es fundamental, ya que los controladores manejan la complejidad de adaptarse a diferentes sensores, actuadores y sus diferentes protocolos de comunicación, mientras que Niagara 4 recibe comunicación exclusivamente a través de BACnet, con flexibilidad en IP-BACnet y MS/TP-Bacnet [9]. Esto amplía significativamente las capacidades de los controladores para adaptarse a diversas necesidades en la gestión de edificios. Dependiendo del modelo, se pueden encontrar una variedad de sensores y dispositivos conectados, cada uno utilizando su propio protocolo de comunicación específico.

En resumen, esta combinación ofrece una solución versátil y completa para la gestión de edificios, con la capacidad de integrar una amplia gama de dispositivos y sistemas gracias a la plataforma Niagara 4. La siguiente imagen muestra una arquitectura general del sistema [10]:

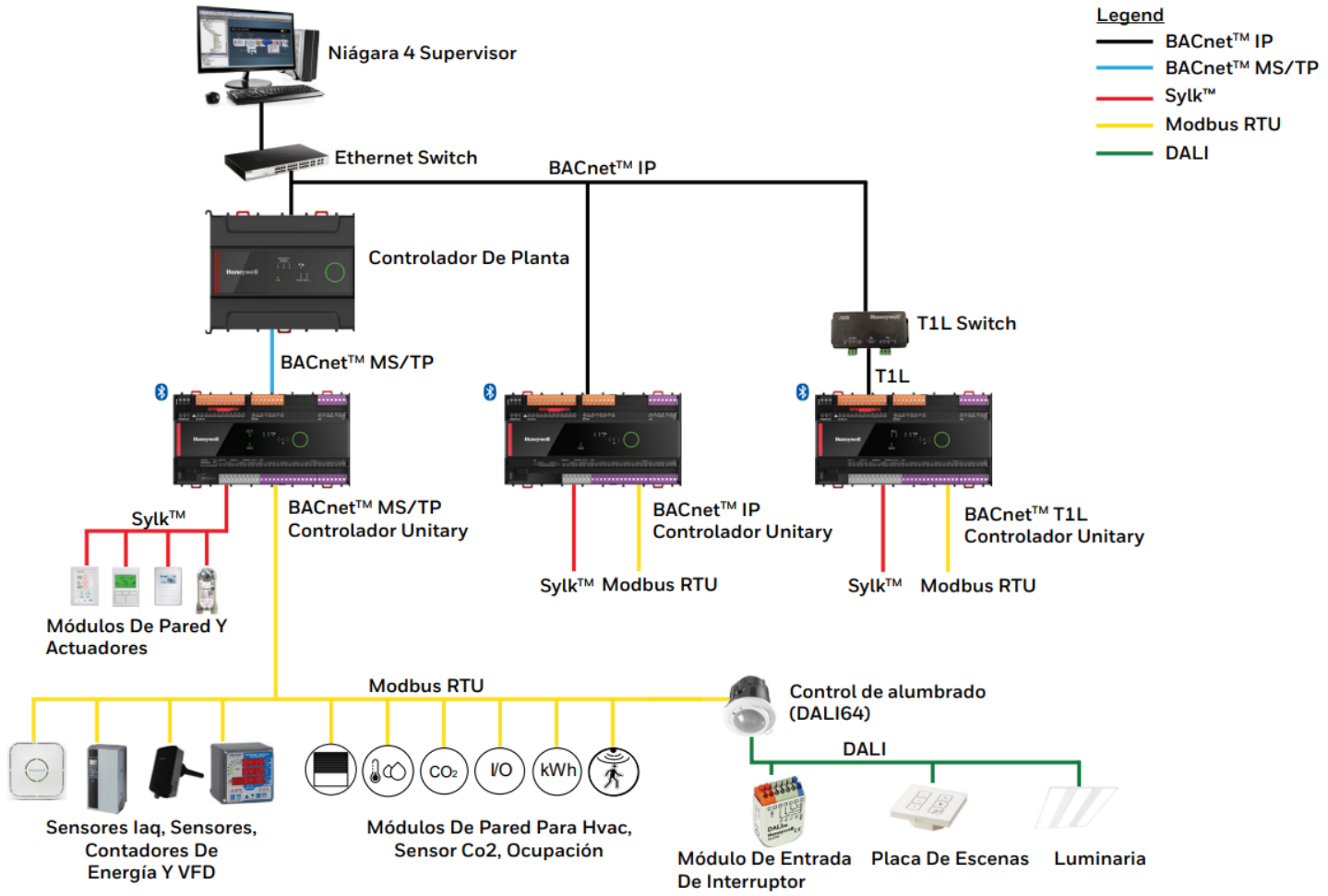

Figura 1: Arquitectura general del sistema

2.4 Descripción general del Hardware

Estos controladores, como se mencionó anteriormente, ofrecen BACnet IP, BACnet T1L o BACnetTM MS/TP como protocolos de comunicación principal, además de Sylk y MODBUS como protocolos de integración. También cuentan con UIO capaces de proporcionar valores específicos de VDC o corriente, y de leer valores analógicos como resistencias, voltajes y corrientes. Además, incluyen relés de potencia y SSR.

A continuación, se presenta una imagen que muestra una descripción general del controlador [10]:

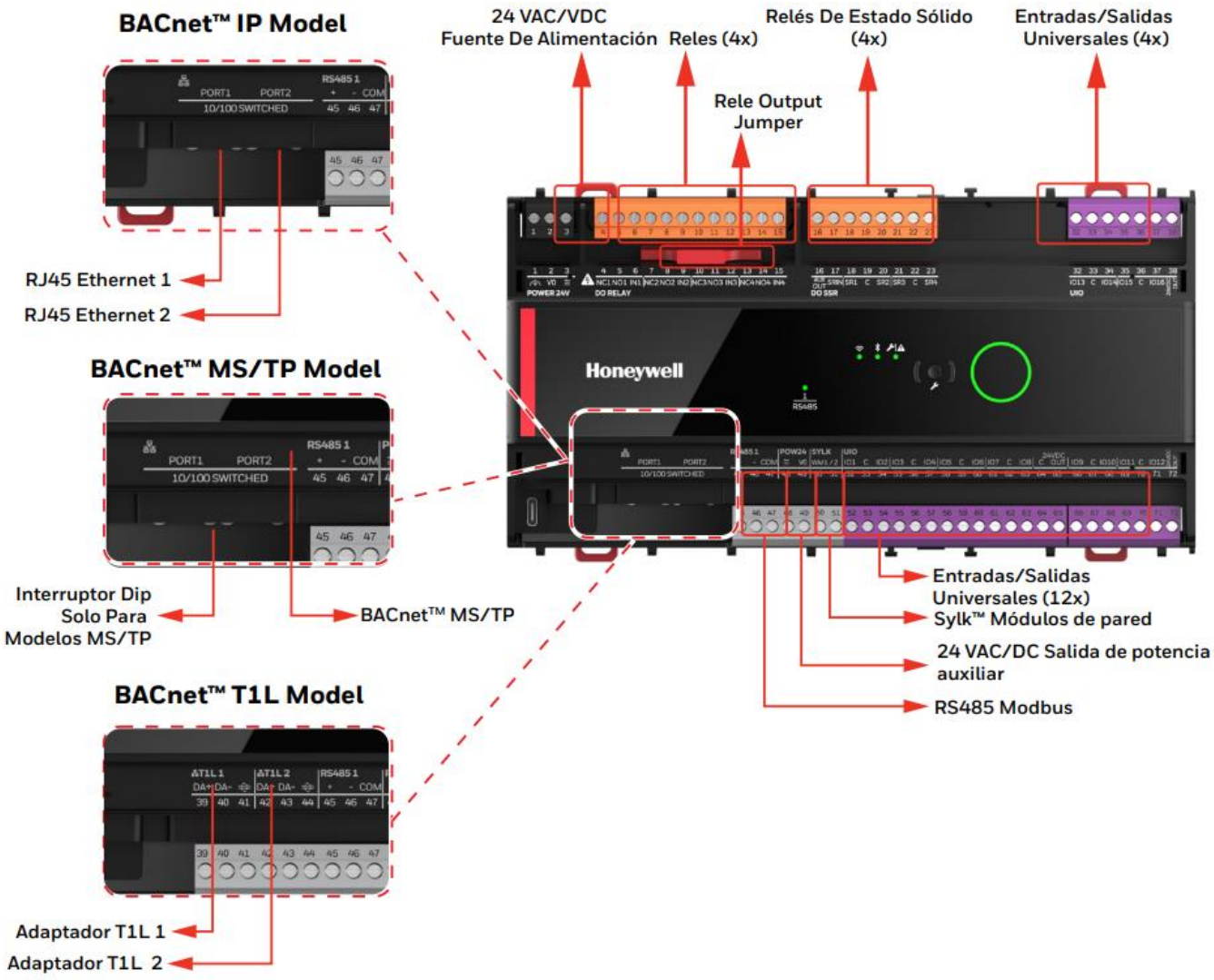

Figura 2: Descripción de puertos general del Controlador Optimizador Unitario

2.5 Consideraciones de diseño

Esta sección documenta consideraciones de diseño específicas para los elementos identificados en la Sección 2.1 - "Propósito".

2.5.1 Objetivos de diseño

El objetivo del procedimiento de Prueba de Fábrica es ejercitar todos los periféricos del DBP para probar que funcionen correctamente y dentro de los límites establecidos por el diseño de hardware y software. El enfoque se centra en las interfaces de hardware y sus parámetros, que deben estar dentro de las especificaciones.

2.5.2 Nomenclatura de Modelos

Al inicio de la Estación de Prueba Funcional (FcT), el operador debe seleccionar el número de sistema operativo. Esta elección es crucial, ya que determina la configuración y el funcionamiento del controlador. Además, es importante tener en cuenta que cada modelo de controlador puede tener diferentes puertos disponibles.

Tabla 1: Significado de las nomenclaturas de los modelos

Característica	Símbolo
Room Controller	R

Carcasa: *Small/Large*	L/S
Número de UIO	8 o 12 o 16
Número de SSR	4 u 8
Número de Relay	4 or 8
Comunicación principal	E = Ethernet
	M = BACnet MSTP
	T = BACnet IP T1L
Módulo de Pared	S = Sylk
BLE:	B = BLE
Voltaje de alimentación	24 VAC
	230 VAC
Herramienta de software	N = Niagara
	V = VisualLocalize
Modbus:	M = Modbus Máster
	S = Modbus Slave

3.2.3 Optimización del Proceso de Construcción del Proyecto

En el desarrollo del *firmware* para los distintos modelos de controladores de automatización de edificios utilizamos el compilador IAR, que se integra mediante un *plugin* al IDE Eclipse, herramientas de construcción como CMake y gestionando la compilación del proyecto a través de archivos *batch*. Cada modelo de controlador tiene características específicas que requieren un proceso de construcción del proyecto adaptado a sus funcionalidades. Estas características están detallas en la Tabla 1.

Anteriormente, este proceso se manejaba mediante múltiples archivos de construcción individuales para cada modelo, lo cual era ineficiente y complicado de mantener. Con el tiempo, la gestión de estos archivos se volvió engorrosa, especialmente al agregar nuevas variantes de hardware.

Para resolver este problema, implementé un enfoque modular y optimizado que centraliza la construcción del proyecto. Desarrollé un único archivo raíz de construcción que, a través de parámetros configurables, puede determinar dinámicamente qué librerías, módulos y funcionalidades se deben incluir en el código final. Este cambio no solo simplificó el proceso, sino que también permitió una mayor flexibilidad y capacidad de ajuste en tiempo real. A continuación, se presenta un ejemplo del *batch* que desarrollé:

Tabla 2: Parámetros de configuración para construcción del proyecto

Parámetro	Descripción
Variante	Define la comunicación principal del controlador: T1L, Ethernet o MS/TP
Voltaje	Especifica el voltaje del Controlador Optimizador Unitario: 24V o 230V, según el modelo seleccionado
Funcionalidades extra	Activas características adicionales del producto, como la funcionalidad BLE si aplica al modelo
Tamaño	Selecciona el tamaño del controlador: *Large* o *Small*

Este *script* acepta parámetros de entrada relacionados con el modelo de hardware, permitiendo que el proceso de compilación se ajuste automáticamente a las características específicas de cada variante. Ya no es necesario mantener múltiples archivos de configuración, es más entendible y ahora es fácilmente escalable si se necesitaran agregar nuevas funcionalidades.

Además, al integrar CMake en el proceso de construcción, logré que los parámetros de compilación se conviertan en macros y directivas de preprocesamiento. Esto permite que el código se adapte de manera condicional durante la etapa de preprocesamiento, lo que optimiza la inclusión de archivos de cabecera y funcionalidades según los requerimientos del hardware. El resultado es un proyecto final que solo incluye el código necesario para cada modelo y especificaciones del Controlador. Como resultado de esta optimización, logré reducir el número de archivos de construcción de manera considerable, mejorando la mantenibilidad y modularidad del proyecto. Ahora es más sencillo integrar nuevos modelos de controladores, ya que el sistema de construcción es más flexible y escalable. Este enfoque también ha permitido que el equipo realice ajustes y actualizaciones de forma más rápida, lo que ha reducido los tiempos de desarrollo y mejorado la eficiencia general del proceso.

3.2.4 Principios y Normas de Diseño

Participé grupalmente con equipos de hardware, arquitecto del Controlador, diferentes personas en fábrica para la definición del Procedimiento de la FFT, el cual ayuda a que los dispositivos cumplan con las especificaciones de calidad internas de la empresa, antes de ser enviados a los clientes. Este procedimiento se divide en dos componentes esenciales: el DBP y la Estación de Prueba de Fábrica, que trabajan de manera integrada para evaluar el rendimiento de los periféricos de hardware.

El DBP fue diseñado para entrar en un modo especial tras la programación inicial, permitiendo que las Estaciones de Prueba envíe comandos para activar y verificar el funcionamiento de los periféricos. En este modo, el DBP responde a las solicitudes enviadas por las diferentes Estaciones de Pruebas, proporcionando resultados que indican si cada periférico cumple con los criterios de *PASS*/*FAIL* establecidos en las especificaciones de FFT.

Aunque las diferentes Estaciones de Prueba de Fábrica fueron desarrolladas e implementadas en fábrica por un tercero, mi contribución se centró en la elaboración de las especificaciones necesarias para su funcionamiento. Esto incluyó la creación de un manual de FFT que utilizaron para su diseño y desarrollo, creé un programa de FFT para realizar las pruebas, los diferentes comandos y la definición de los requisitos de validación.

El flujo del proceso de prueba es el siguiente: la Estación de Prueba solicita el funcionamiento de un periférico, el DBP ejecuta la acción y confirma su operación, y la Estación de Prueba valida que los parámetros del periférico se encuentren dentro de los límites esperados. Si se detecta alguna anomalía, se procede a retirar el dispositivo para su retrabajo o descarte.

A continuación, se presenta un esquema general del flujo del proceso de prueba:

Figura 3: Flujo general del proceso de prueba

2.6 Requerimientos de Software del Producto

- El software para el DBP se cargará al controlador para que cumpla con la especificación de prueba de fábrica.
- UART (TTL, Baudios: 115200) es interfaz serial utilizada con el equipo de prueba de fábrica para establecer la comunicación.
- Los mensajes seriales estarán delimitados por <> por ejemplo, <Texto del mensaje>

El uso de UART como interfaz serial y la delimitación de los mensajes con < > permiten una comunicación clara y eficiente entre el equipo de prueba de fábrica y el Controlador. Esto es importante para asegurar que las pruebas se realicen de manera correcta y que el software cargado cumpla con las especificaciones requeridas. Estas medidas facilitan la verificación del correcto funcionamiento del dispositivo durante el proceso de prueba.

Para facilitar la gestión de los comandos y mejorar la legibilidad del código, diseñé una tabla de comandos que asocia cada uno a una función específica mediante un arreglo de estructuras. Esto permite que, al recibir un comando, el sistema identifique rápidamente la acción correspondiente y la ejecute. La tabla, implementada como un arreglo de pares clave-valor, facilita la expansión de las funcionalidades del sistema:

```
comando_entrante comandos {
    char *nombre_comando;
    void (*función_comando)();
};

static comando_entrante comandos[] = {
    {"comando_1", funcion},
};
```

Cada entrada de la tabla contiene el nombre del comando y un puntero a la función correspondiente. Al recibir un nuevo comando, el sistema lo busca y ejecuta la función asociada, simplificando la lógica de procesamiento y mejorando la modularidad del código.

Por ejemplo, el comando <fft_uio_set_mode><index><modo> recibe parámetros adicionales para su procesamiento. Este comando especifica un índice que indica el periférico objetivo y el modo que se debe aplicar. Al recibir el comando, el sistema extrae estos parámetros y los recibe la función fft_uio_set_mode, que se encarga de aplicar el modo configurado al periférico correspondiente.

Para la lógica del procesamiento de comandos, incluí la verificación de argumentos, asegurando que cada comando se ejecute con los parámetros correctos. Si los argumentos o el mismo comando no son válidos, el sistema envía mensajes de error apropiados a través de la interfaz UART, informando al operador sobre el problema con el comando.

La función principal del sistema, diseñada para escuchar continuamente los comandos entrantes, utiliza un bucle que se ejecuta en un hilo específico. Este hilo espera la recepción de nuevos comandos a través de la UART. Cuando se detecta un comando, se verifica su validez y se invoca la función correspondiente. Después de la ejecución del comando, el sistema vuelve a esperar nuevos comandos. Este enfoque secuencial garantiza un control detallado sobre cada periférico durante el proceso de prueba.

Adicionalmente, implementé un temporizador de *watchdog* en el hilo que se refresca periódicamente para garantizar el funcionamiento continuo del controlador. Este mecanismo se agregó para prevenir fallos en el sistema, ya que permite reiniciar el controlador si se detecta que no está operando correctamente o si se ha producido un *hardfault* dentro del microcontrolador.

La arquitectura la diseñe para proporcionar flexibilidad al realizar pruebas específicas en diferentes componentes. Por ejemplo, si es necesario ajustar la configuración de un sensor, se puede implementar un nuevo comando que interactúe directamente con el periférico correspondiente. Esto permite modificar parámetros de manera eficiente sin la necesidad de realizar cambios en el código base. La modularidad del sistema también simplifica la realización de pruebas unitarias. Cada comando y su correspondiente

función pueden ser probados de forma aislada, asegurando que las actualizaciones o cambios en una parte del código no afecten a las demás.

2.7 Requerimientos Mecánicos del Producto

2.7.1 Montaje y Orientación

El DBP se monta en el equipo de prueba de fábrica con el lado del logotipo hacia arriba. Esto proporcionará acceso a los TPs de UART para habilitar la comunicación, USB que es el protocolo utilizado para programar al controlador, alimentación y entradas/salidas desde la parte inferior del dispositivo

2.7.2 DBP

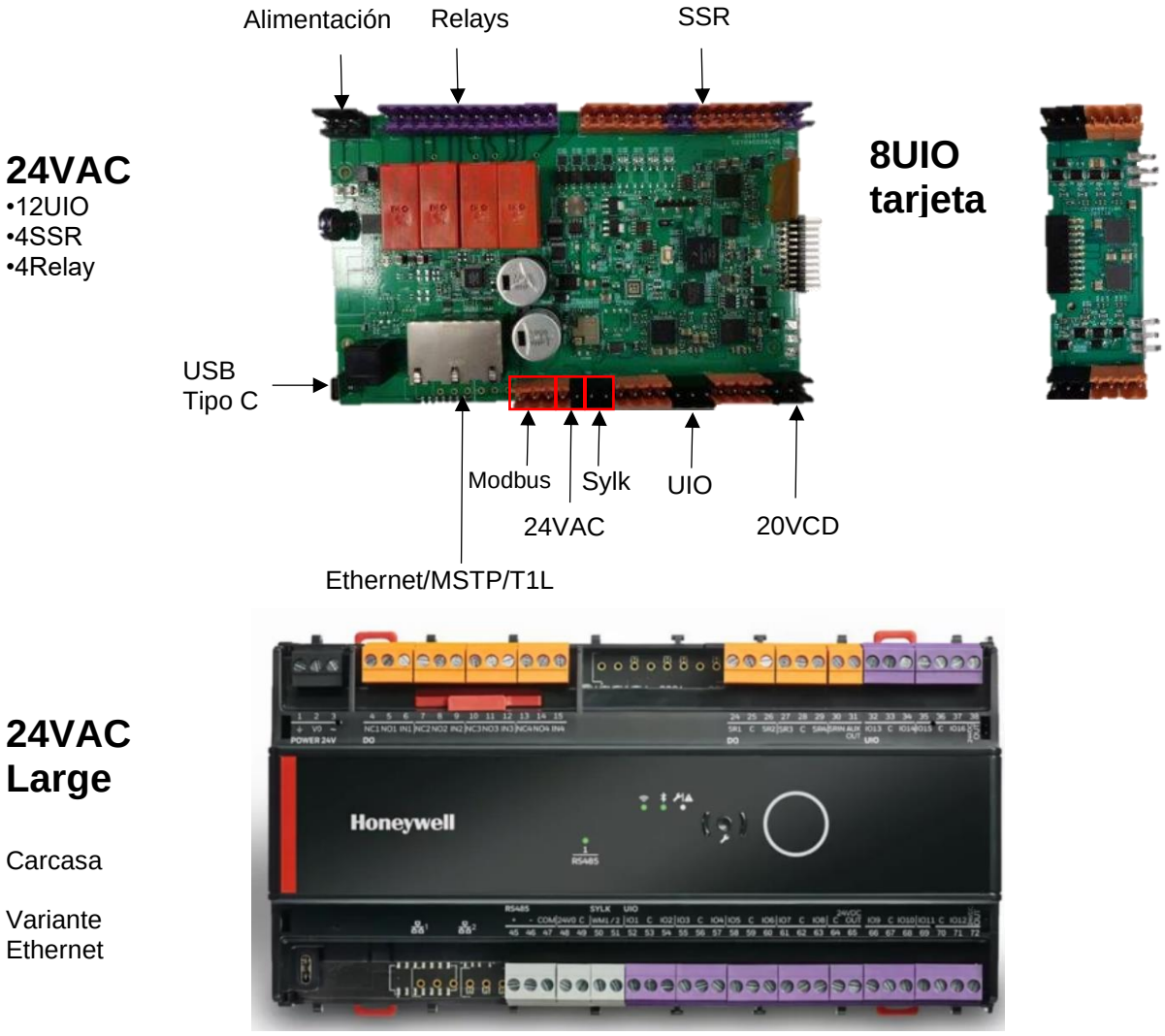

Figura 4: Imagen de carcasa del controlador Optimizador Unitario

4 Prueba de fábrica

Como se menciona en la sección *3.4 Requerimientos de Software del Producto*, la comunicación entre el *fixture* y la placa debe ser a través de UART enviando mensajes seriales delimitados por <> por ejemplo <Texto del Mensaje>, con 115200 como valor de baudios. La respuesta de los comandos será una de las siguientes:

- <OK> Después de una ejecución exitosa.
- <EINVALID> Después de cualquier formato de *string* inválido.
- <EFAIL> Después de cualquier error de ejecución.

Además, otros parámetros adicionales aparecerán en la respuesta del DBP dependiendo de la función. La siguiente conexión mostrada en la Figura 10 se utilizará en todas las estaciones, está relacionada con la fuente de alimentación y la comunicación *fixture*-tarjeta.

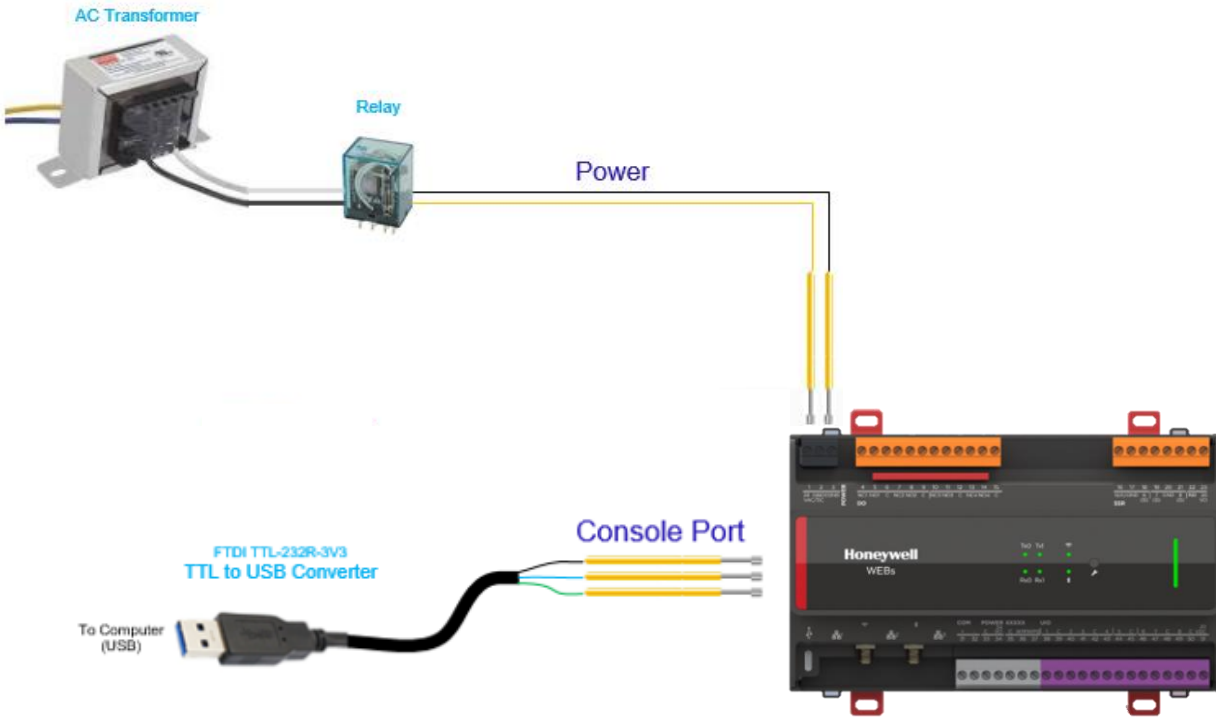

Figura 5: Conexión general del Controlador con las *fixtures* en las diferentes estaciones.

4.1 Estación de Programación

4.1.1 Descripción

La intención de esta prueba, además de programar el controlador, es preparar el dispositivo para su envío al cliente final. Para lograrlo, se reemplaza el código de FFT por el de la aplicación que utiliza el cliente. Estos dos programas son independientes y se cargan en etapas distintas.

En esta fase, llamada estación de programación, se configuran los diferentes chips con el *firmware* de FFT para realizar las pruebas, y todo esto se lleva a cabo en la estación 1.

Chips:

- Microcontrolador; NXP - i.MXRT1051CVJ5B.
- Módulo de BLE: Nordic - nRF52840.

4.1.2 Visión general

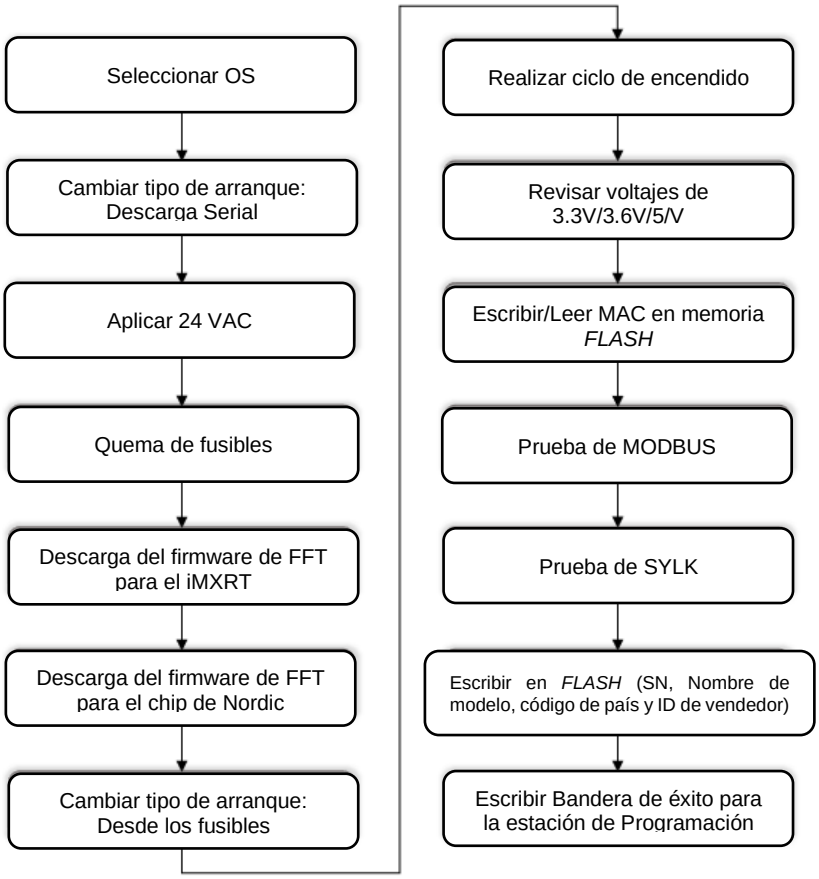

Figura 6: Flujo de estación de Programación

Seleccionar el modelo

El controlador se coloca en la fixture de programación conectándose a través de una interfaz USB-C, lo que permite interactuar con las herramientas de programación. El proceso de programación se lleva a cabo utilizando la MFGTool de NXP.

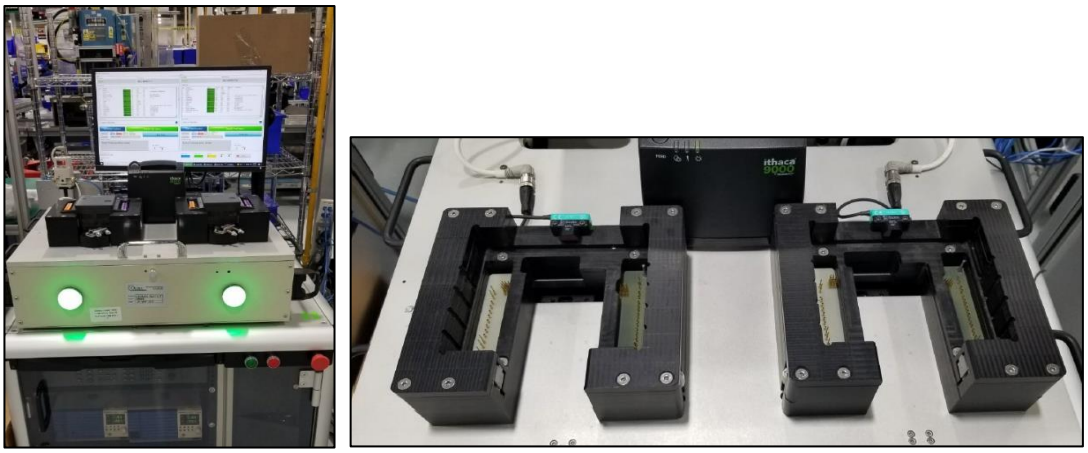

Figura 7: Fixture de Programación

El operador selecciona el modelo del controlador, lo que ajusta el flujo de programación, crea el plan de prueba junto con los diferentes comandos que se usaran durante la FFT de ese controlador. Para este proceso, proporcioné las instrucciones para configurar las herramientas necesarias en esta etapa.

Una vez seleccionado el modelo, proceden a iniciar la programación de FFT.

Aplicación de Fuente de alimentación

Se aplican 24VAC al controlador a través del DBP para proveer la energía necesaria y proceder con la quema de fusibles, escritura de datos y la programación de los diferentes programas. Programación del microcontrolador iMXRT con el firmware de FFT

La estación de programación comienza utilizando la herramienta NXP MCU BootUtility para la quema de fusibles, una etapa que depende del modelo de memoria y de las especificaciones del proyecto. Posteriormente, se ejecuta un archivo por lotes que llama al Mfgtool.exe de NXP para descargar la imagen de firmware correspondiente al microcontrolador, la cual varía según la variante utilizada.

Figura 8: Conexión para descargar firmware al MXRT

Quema de e-Fuses del microcontrolador

Como parte de mi responsabilidad en el proyecto, realicé una investigación en el manual de referencia del microcontrolador i.MXRT1051 de NXP [11] y este requiere una serie de configuraciones previas y la quema de fusibles específicos dado que por la arquitectura del Controlador se utiliza una la memoria externa *NOR Flash* y se debe habilitar esta comunicación en el microcontrolador. Utilicé la herramienta NXP MCU BootUtility para configurar la memoria y quemar los fusibles necesarios.

NXP MCU BootUtility es una herramienta desarrollada por la empresa NXP que permite configurar, inicializar y quemar fusibles en microcontroladores de la familia i.MXRT. Proporciona una interfaz que facilita la configuración de los dispositivos de memoria externos y la gestión de las opciones de arranque y seguridad del sistema.

Al iniciar el uso de este programa, se configuren ciertas características en función del modelo de memoria utilizado. En esta etapa, definí las configuraciones necesarias que deben implementarse en el programa para esta estación de programación.

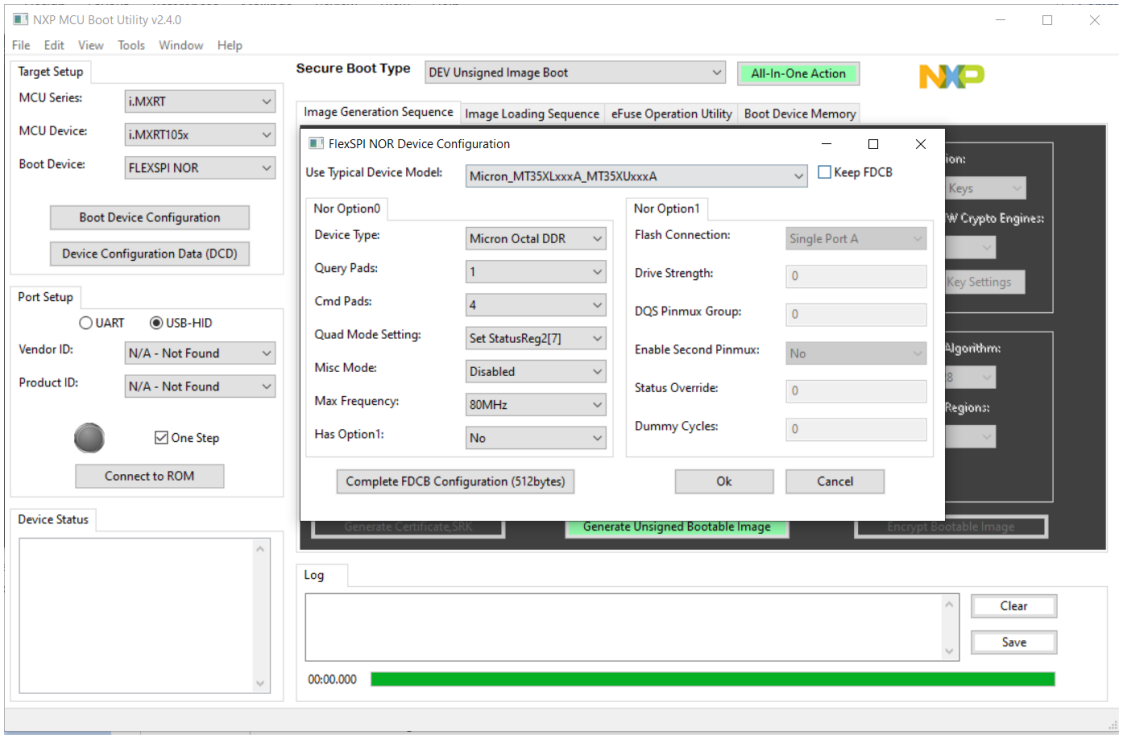

Figura 9: Interfaz de NXP MCU BootUtility

Una vez realizada la configuración de la memoria, se procede a la quema de fusibles utilizando la sección eFuse Operation Utility dentro de NXP-MCU-BootUtility. Este paso incluye la quema de fusibles específicos para:

- Habilitar la *NOR Flash* para el arranque externo.
- Deshabilitar el JTAG en las versiones de producción para proteger el firmware.

Para esto instruí a los operadores de fábrica y les compartí dos configuraciones diferentes dependiendo de la etapa en el que se encuentre el proyecto:

- Desarrollo (R&D): La *NOR Flash* está habilitada, pero el JTAG permanece activo para poder depurar en el controlador y facilitar el desarrollo de nuevas funcionalidades.
- Producción: La *NOR Flash* está habilitada y se deshabilita el JTAG para evitar el uso de funciones de depuración, ya que estas no son necesarias para el cliente y también se bloquea por razones de seguridad

Programación del Nordic con el firmware de FFT

La estación de programación ejecuta un lote de *JFlash* del programador Segger J-Link para descargar el firmware FFT de Nordic si la variante lo requiere.

Figura 10: Conexión para descarga de firmware del chip de NORDIC

Revisión de voltajes iniciales

Al terminar la programación de los chips, se realiza una revisión de ciertos voltajes importantes en la placa utilizando el *Standard Rack Instrumentation* 34980A en ciertos TPs, definidos por el equipo de Hardware. El objetivo es evitar dañar en la siguiente etapa el equipo de prueba. Algunos de los voltajes más importantes incluyen la revisión del voltaje de entrada al microcontrolador, que debe ser de 3.3 V, el voltaje del chip Nordic de 3.6 V y el voltaje de alimentación de la placa de 21 VCD, entre otros voltajes importantes.

Escritura de datos en la memoria *flash*

En este paso, la estación de programación envía los siguientes comandos para realizar una escritura de datos en la memoria *FLASH* externa:

Tabla 3: Comandos para leer y escribir datos en flash

Datos	**Comando de escritura en memoria *flash***	**Lectura de memoria *flash***
Vendor ID	<fft_set_BACnet_ID>< ID_string><chksum>	<fft_get_BACnet_ID><chksum>
Country code	<fft_set_Country_Code><country_string><chksum>	<fft_get_Country_Code><chksum>
Serial Number	< fft_set_device_sn><device_string><chksum>	< fft_get_device_sn><chksum>
Model Name	< fft_set_model_name><model_string><chksum>	<fft_get_model_name><chksum>
Vendor Name	<fft_set_vendor_name><model_string><chksum>	<fft_get_vendor_name><chksum>
RTC	<fft_rtc_set><date><time><chksum>	<fft_rtc_get><chksum>

Estos datos permiten asociar un registro e identificador a las unidades manufacturadas. Algunos, como el SN, son únicos para cada controlador y se utilizan para su identificación en la fábrica. Información como el *Vendor ID*, el *Model Name* y el SN son relevantes para el código de la aplicación. En una red BACnet con varios controladores, estos datos actúan como identificadores, facilitando la comunicación.

Para esta prueba, implementé pasos que incluyen la definición de particiones específicas en la memoria persistente, de acuerdo con los requisitos proporcionados por el arquitecto de firmware. Estas particiones están diseñadas para almacenar los datos mencionados en la Tabla 3.

Desarrollé las funciones necesarias de cada comando, para escribir cada uno de estos datos en sus ubicaciones de memoria específicas. Esto incluyó la implementación de *drivers* que gestionan la escritura en la memoria persistente del controlador, asegurando que cada dato se almacene en la partición y dirección adecuadas.

Además, incluí un manejo de errores para que el operador en fábrica pudiera validar que cada parámetro escrito cumple con el formato requerido. Si un dato no se escribía correctamente o no cumplía con los criterios predefinidos (como una longitud o valor inválido), el controlador imprime en consola un código de error y una breve descripción de este.

Esta implementación permite que, en cada estación de programación, los datos específicos de cada unidad manufacturada (como el SN y otros identificadores únicos) se almacenen de forma accesible para su uso posterior en la aplicación y en la interacción con otros dispositivos dentro de una red BACnet.

Bandera de programación

Para abordar problemas en la identificación de los controladores en la fábrica después de cada estación, implementé una solución que consiste en escribir una bandera en un espacio específico de la memoria persistente al finalizar cada estación de la FFT. Esta bandera ocupa el mismo espacio de memoria, pero su contenido cambia a medida que avanza el proceso en la FFT, reflejando el estado actual del

controlador. Antes de iniciar una nueva estación, el sistema verifica si la bandera ha sido actualizada correctamente, permitiendo que el proceso continúe solo si la escritura fue exitosa.

En la estación de programación, el comando <fft_timestamp_set><TIMESTAMP> escribe el valor del parámetro *TIMESTAMP* en la memoria *FLASH*. En esta etapa, el valor asignado a *TIMESTAMP* es <*PROG_PASSED*>, lo que indica que la programación se realizó con éxito.

4.2 FcT

4.2.3 Descripción

Para el desarrollo de esta etapa, será necesario utilizar la Estación 2, la cual se encargará de probar las funcionalidades principales del Controlador.

4.2.4 Requerimientos

Tabla 4: Requerimientos de la estación FcT

Tipo	Requerimiento	Propósito
Hardware	Instrumentación estándar de bastidor 34980A	Sistema de adquisición de datos para medir voltaje de corriente continua
Hardware	Adaptador USB-Sylk	Utilizado para la prueba de sylk test
Hardware	Resistor de potencia de 24 ohmios en serie	Usado para prueba de lectura de resistencia en UIO
Hardware	Adaptador USB-ethernet	Prueba Ethernet/T1L
Hardware	Cable Ethernet	Prueba Ethernet/T1L
Hardware	DEMO-ADIN1100-D1Z	Prueba T1L
Hardware	Cable RS485	Prueba MSTP

4.2.5 Vista general

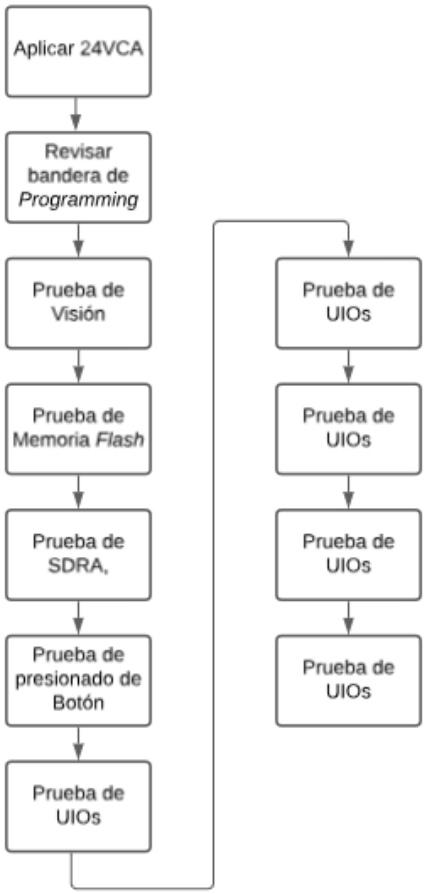

Figura 11: Diagrama de flujo general de FcT

4.2.6 Procedimiento

Colocar el controlador en la estación

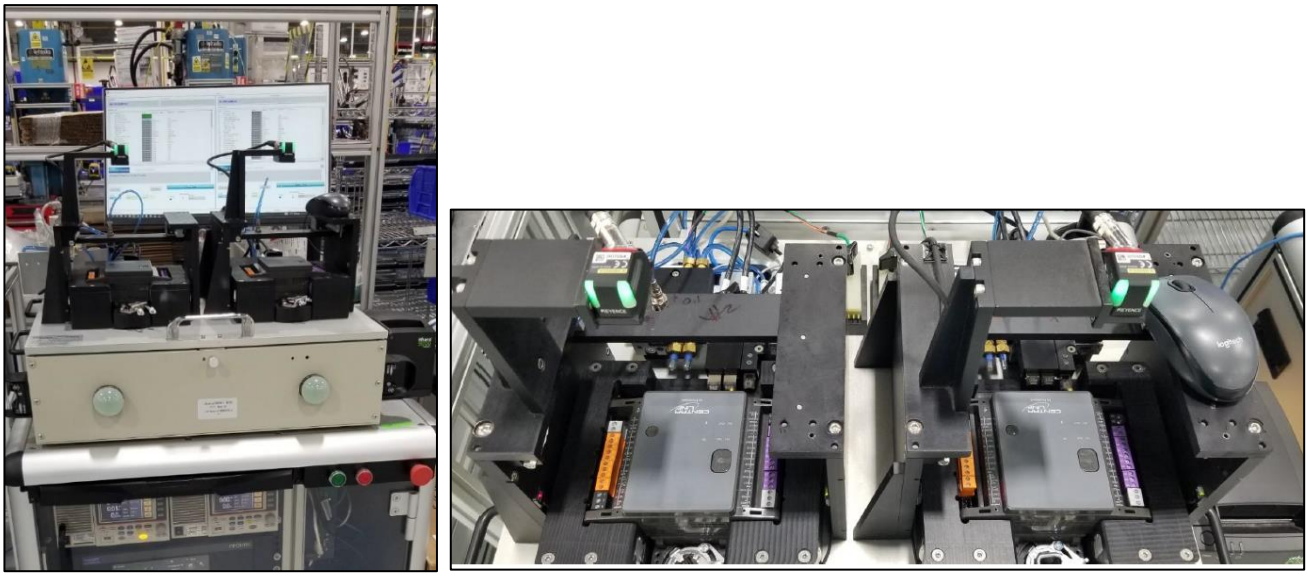

Figura 12: FcT fixture

Aplicar suministro

La FcT aplica 24VAC al DBP.

Revisar bandera de programación

La Fct envía el comando <fft_timestamp_get>. El DBP debe responder con <PROG_PASSED> para continuar con las pruebas. Esta verificación asegura que la programación se realizó correctamente antes de avanzar a las siguientes etapas de prueba.

Prueba de visión

- **Prueba de LED**

El objetivo de esta prueba es validar el funcionamiento de todos los LED en la tarjeta. Cada LED, dependiendo de la variante, indica una función específica. En el firmware de aplicación, se utilizan distintos colores para mostrar el estado de la calidad de las señales o para alertar sobre problemas en algún periférico.

En la FFT, la estación de FcT envía el comando <fft_led_set><index><COLOR> al DBP para configurar el LED en rojo, verde o amarillo.

Tabla 5: Índice y color permitido por variante

Variante	Índice	Color			
		ROJO	AMARILLO	VERDE	APAGADO
ETHERNET / T1L / MSTP	1-9	√	√	√	√
	TODOS	√	√	√	√

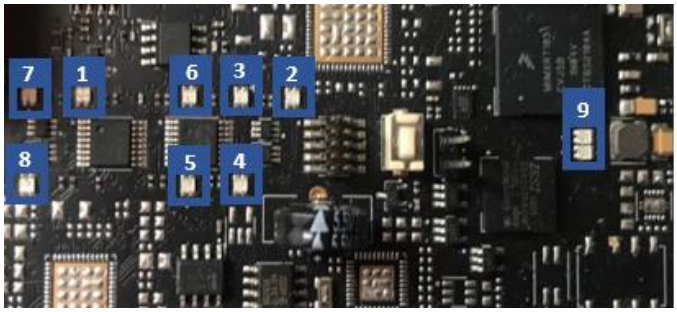

Figura 13: LED del Controlador FCU

Tabla 6: Nombre de los LED

Índice	1	2	3	4	5	6	7	8	9
LED	T1L2	Botón de servicio	BLE	T1L_0	RS485_0	Wi-Fi	T1L_1/MSTP	RS485_1	Estatus

Se realiza el encendido de cada LED para verificar su correcto funcionamiento.

Optimización del Proceso de Prueba de LED:

Durante el proceso original de la prueba, cada uno se activaba de manera individual, requiriendo el envío de comandos separados para cada LED, lo que extendía el tiempo total de prueba a 35 segundos. Para mejorar la eficiencia, implementé una solución que permite activar todos los LED mediante un único comando. Esta modificación redujo significativamente el tiempo de la prueba a solo 5 segundos.

Esta optimización disminuyó el tiempo de la prueba en la FFT, y también ayudó a reducir el tiempo total del ciclo de pruebas.

Prueba de Modbus

El FcT envía <fft_modbus><chksum> al DBP.

El DBP |RS485 envía la cadena "ABCDE" al FcT|RS485, y luego espera recibir la cadena "12345" como respuesta del FcT|RS485.

Al recibir "12345", el DBP |RS485 envía <fft_modbus><OK><12345><chksum> al FcT.

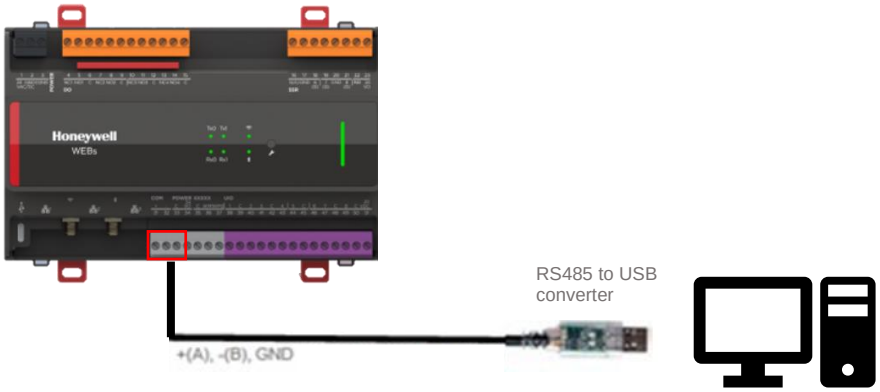

Figura 14: Conexión necesaria para la prueba de modbus

Para esta prueba, me coordiné con el arquitecto del controlador la definición de un proceso que permitiera validar el puerto Modbus de manera eficiente. Dado que Modbus se comunica a través de la interfaz UART sobre RS485, la prueba se centró en validar las capas físicas y de enlace de datos, evitando la necesidad de probar exhaustivamente el protocolo Modbus, que opera a niveles superiores del modelo OSI. La prueba consiste en el envío y recepción de cadenas a través de UART/RS485, asegurando la integridad de la transmisión y recepción de datos. Este enfoque ayudó a reducir el tiempo de validación en fábrica y garantizó que el puerto funcione correctamente, asegurando su preparación para manejar las comunicaciones Modbus a nivel de aplicación.

Prueba de Sylk

El protocolo Sylk es un estándar patentado por la empresa donde trabajo, diseñado para facilitar la comunicación con termostatos de la misma compañía. Este protocolo es muy usado en los sistemas de control HVAC. Utiliza un bus que combina la transmisión de datos y la alimentación.

El FcT envía <fft_sylk_get_value><chksum> al DBP.

El DBP |Sylk envía la cadena "ABCDE" al FcT|Sylk, y luego espera recibir la cadena "12345" como respuesta del FcT|Sylk.

Al recibir "12345", el DBP |RS485+/- envía <fft_sylk_get_value><OK><12345><chksum> al FcT.

Para esta prueba, se necesita el PCMOD W7220 como dongle USB-Sylk.

La salida Sylk de la placa del PCMOD debe estar conectada al dongle S-bus, y su puerto USB al ordenador.

Figura 15: Conexión para sylk usando el W7220 PCMOD

El objetivo de esta prueba es asegurar que la comunicación entre el FcT y el DBP funcione correctamente en campo, utilizando RS485 y UART. Aunque no se utiliza el protocolo Sylk en su totalidad, la prueba valida el funcionamiento del puerto RS485, que es necesario para la implementación del protocolo.

Durante este proceso, investigué productos dentro de la empresa que soportaran el protocolo Sylk e identifiqué al W7220-PCMOD como un reemplazo viable. Este dispositivo actúa como traductor de los mensajes enviados a través de UART/RS485, por lo que implemente el comando antes descrito, para validar la recepción/transmisión de *strings* y así asegurar la integridad de la comunicación en ambas direcciones.

Prueba de UIO

Esta prueba verifica el funcionamiento de los puertos UIO, que pueden proporcionar valores específicos de voltaje o corriente y leer valores analógicos. Asegura que cada funcionalidad de los puertos UIO se utilice correctamente.

Los modos que podemos configurar se muestran en la siguiente tabla, además de los valores a probar.

Tabla 7: modos de UIO y valores aplicados

Tipo	MODO DE ENTRADA	MODO DE SALIDA
Voltaje	VOLIN	VOLOUT

Resistencia	RESIN	
Corriente	CURIN	CUROUT

ENTRADA

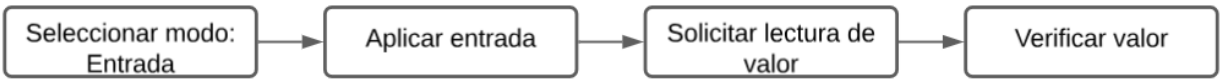

Figura 16: Flujo de prueba de entrada de UIO

Tabla 8: Prueba de entrada UIO detallada

Paso	Detalles
Enviar comando al DBP	<fft_uio_set_mode><index><modo><chksum>
Utilizando una fuente de alimentación, se aplica un valor de entrada especifico a todos los UIO (Voltaje o corriente)	Voltaje (5000 mV) Corriente (10 mA)
Enviar comando al DBP para realizar la lectura	<fft_uio_get_reading><index><chksum>
Verificar si la lectura corresponde con el valor suministrado	

Entrada de voltaje

1. El FcT envía comandos al DBP para configurar todos los UIO en modo de entrada de voltaje y luego aplica 5VDC a todos los UI/UIO.
2. El FcT envía comandos al *DBP* para leer los UIO:

   <fft_uio_get_reading><índice><chksum>

   2.1. El DBP responde con el valor.:
   <fft_uio_get_reading><OK><índice><VOLIN><**valor leído en mV**><chksum>
3. El FcT verifica si los valores leídos están dentro de las especificaciones.

Lectura de resistencias

1. El FcT envía comandos al DBP para configurar todos los UIO en modo de entrada de resistencia y luego conectar la serie de resistencias a todos los UI/UIO.
2. El FcT envía comandos al DBP para leer los UIO:

   <fft_uio_get_reading><índice><chksum>

   2.1. El DBP responde con el valor.:
   <fft_uio_get_reading><OK><índice><RESIN><**valor leído en Ohms**>
3. El FcT verifica si los valores leídos están dentro de las especificaciones.

Modo SALIDA

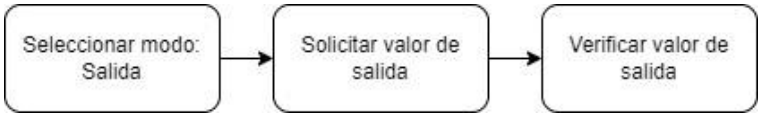

Figura 17: Flujo general de pruebas de UIO

Tabla 9: Salidas UIO detalladas

Paso	Detalles
Enviar comando al DBP para configurar el modo de salida.	<fft_uio_set_mode><índice><modo>
Enviar comando al DBP para generar la salida.	<fft_uio_set_outvalue><índice><valor>
Verificar si el valor está dentro de las especificaciones.	El FcT utiliza un multímetro para verificarlo.

Salida de corriente

1. El FcT envía comandos al DBP para configurar todos los UIO en modo de corriente*:*

   <fft_uio_set_mode><índice><CUROUT>

2. El FcT envía comandos al DBP para que emita hasta 20 mA:

   <fft_uio_set_outvalue><índice><valor>

El FcT utiliza un multímetro/amperímetro para verificar si los valores están dentro de las especificaciones.

Mejora en la Configuración de los puertos UIO.

Al estar cerca de una etapa de lanzamiento del controlador, el operador en fábrica resaltó que esta prueba tomaba alrededor de 4 minutos, por lo que los *stakeholders* del proyecto indicaron que era un tiempo muy alto para la fábrica, y tuve que trabajar en una optimización para esta prueba.

Realicé una revisión del código e identifiqué que se tenía configurado inicialmente al ADC AD74412R, que es el utilizado por el controlador, en *single mode*. Esta configuración presentó varias desventajas, junto con una incorrecta implementación que impactaron negativamente en la eficiencia y velocidad de la prueba de UIO:

1. Esperas Prolongadas: En *single mode*, tras realizar una lectura, se activa el bit ADC_DATA_RDY solo después de que la conversión del ADC se completa. Sin embargo, no se implementó la verificación de este estado, lo que resultó en que se agregó un *delay* muy grande que se utilizaba para asegurarse de que los datos estuvieran listos. Esto no solo es ineficiente, sino que también puede llevar a lecturas incorrectas si los datos aún no están listos.
2. Lecturas Erróneas: La falta de verificación del estado ADC_DATA_RDY antes de realizar la lectura resultaba en la posibilidad de obtener valores no válidos. Esto hacía que el operador en fábrica al realizar esta prueba, en ocasiones necesitaba repetir las lecturas, aumentando aún más el tiempo.
3. Ineficiencia General: La implementación de esperas prolongadas y la falta de verificación adecuada aumentaban significativamente el tiempo de prueba. En un escenario donde hay 16 puertos que leer, el tiempo de prueba se extendía innecesariamente, lo cual no era aceptable y se debía mejorar.

Por lo tanto, al revisar la hoja de datos del ADC AD74412R [12], hice el cambio en código para configurarlo en *continuous mode*. Esta configuración permite que el ADC mantenga un estado de

conversión constante, lo que optimiza el tiempo de prueba. Además, implementé un mecanismo que verificaba el bit ADC_DATA_RDY antes de cada lectura y así eliminar el uso de *delays* innecesarios para obtener, un tiempo de prueba de UIO de aproximadamente 1 minuto.

Prueba de SSR

Hay diferentes números de SSR dependiendo de la variante:

Tabla 10: Canales SSR por variante

Canales de SSR (command index)	Variante
0 (para activar todos los canales)	MSTP / IP / T1L
1 a 8	MSTP
1 a 4	IP/T1L

Inicialmente, el diseño del Controlador Optimizador Unitario contemplaba la gestión de 4 SSR. Con la introducción de una nueva variante del modelo MSTP, se habilitó la capacidad para controlar hasta 8 SSR. Mi responsabilidad consistió en realizar los ajustes necesarios en el código para permitir el control de las nuevas señales asociadas a estos 4 SSR adicionales.
Para manejar estas salidas adicionales, se incorporó un registro de desplazamiento de mayor capacidad, lo que me permitió gestionar eficazmente las señales de control mediante una interfaz SPI. Revisando el esquemático del controlador que me proporcionó el equipo de hardware, actualicé funciones específicas que habilitan y deshabilitan individualmente cada uno de los SSR. Al finalizar mi desarrollo, llevé a cabo diversas pruebas para validar la correcta implementación de esta funcionalidad.

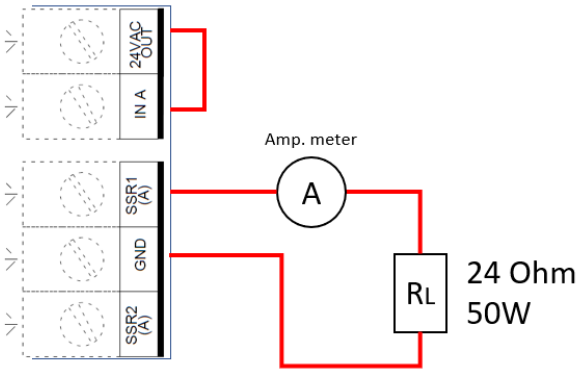

Figura 18: Conexión de prueba del SSR

Prueba:

Tabla 11: Detalle de la prueba del SSR

Pasos	Detalles
Enviar comando al DBP para encender el SSR	<fft_bo_set_outvalue><*índice*><ON>
Aplicar 24VAC al puerto BO	

Aplicar un resistor de potencia de 24 ohmios y 50 W en serie.	FcT utiliza un sistema de adquisición de datos para verificarlo.
Medir la corriente en el circuito.	La corriente debería ser de 1A.
Enviar comando al DBP para apagar el SSR.	<fft_bo_set_outvalue>< *índice* ><OFF>
Medir la corriente en el circuito.	La corriente debería ser de 0A.

Prueba de relés.

El FcT envía el siguiente comando al control del relé y realiza una prueba de continuidad entre NO y común:

<fft_relay_set_outvalue>< *índice* ><valor><chksum>

Teniendo en cuenta que el índice es un número entero decimal de los canales de interfaz de usuario: 1 ~ 4. Y para probar todos los relés al mismo tiempo, se utiliza el índice 0.

Tabla 12: Detalle de la prueba del relé.

Pasos	Detalles
Prueba de continuidad.	Continuidad entre COMÚN-NORMALMENTE ABIERTO
Enviar comando al DBP para activar el relé.	< fft_relay_set_outvalue >< *índice* ><ON>
Prueba de continuidad.	No hay continuidad entre COMÚN-NORMALMENTE ABIERTO
Enviar comando al DBP para desactivar el relé.	< fft_relay_set_outvalue >< *índice* ><*OFF*>

Por ciertos requerimientos y cambios del equipo de hardware, realizaron una nueva revisión de la PCB del controlador. Esta revisión requirió un cambio en el orden de control de los relés a través del registro de desplazamiento. Para implementar estos cambios, revisé el esquemático del controlador que me compartió el equipo de hardware y ajusté la lógica de activación de los relés. Esto incluyó la actualización de las funciones de control para reflejar la nueva disposición de las señales de control. Al final, realicé diferentes pruebas para asegurar la correcta funcionalidad de los relés tras las modificaciones.

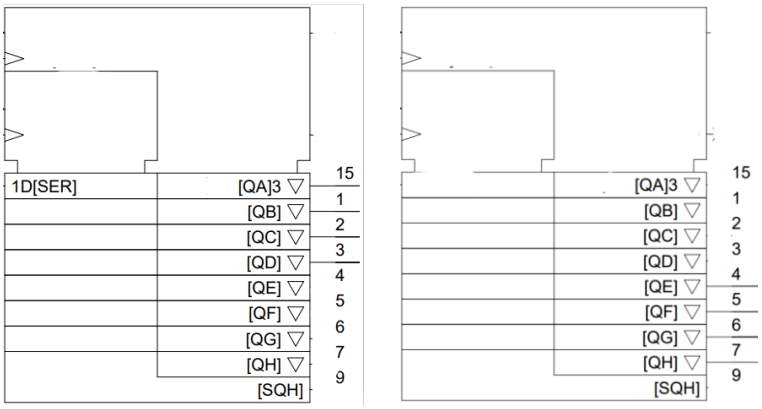

Figura 19: Cambio de conexiones en el registro de desplazamiento

Prueba ETHERNET

Esta prueba solo se aplica a las variantes de Ethernet.

Se requiere establecer una conexión LAN entre la computadora y el DBP, para obtener más detalles sobre la configuración de la PC.

El FcT envía el comando `<fft_ping><192.168.100.104> para hacer ping al DBP a través del cable Eth1. El FcT verifica si el ping fue exitoso.

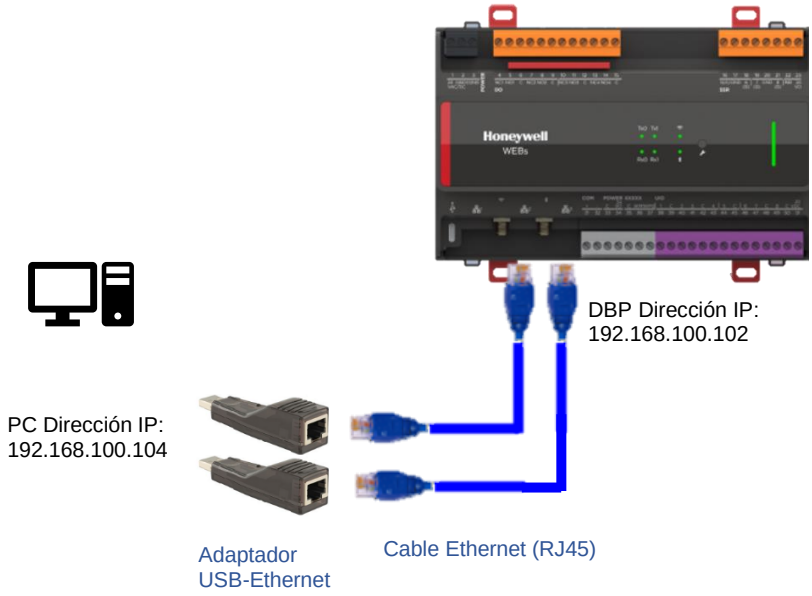

Figura 19: Conexión para la prueba de Ethernet

Con el arquitecto del controlador, trabajé para definir esta prueba y llegamos a la conclusión que el uso de pings para validar el puerto Ethernet es apropiado, ya que este protocolo opera en la capa de red (capa 3) del modelo OSI, utilizando *Internet Control Message Protocol* dentro del conjunto de protocolos TCP/IP. Aunque el controlador en su aplicación final utiliza BACnet/IP para la comunicación, la prueba de ping es suficiente para confirmar que el puerto Ethernet está funcionando correctamente, ya que asegura que la capa de red está operativa y que el hardware responde adecuadamente. Esto proporciona una verificación rápida y efectiva de la conectividad antes de proceder con pruebas más complejas y específicas.

Además, realicé una optimización de la prueba. Inicialmente, esta tomaba aproximadamente 50 segundos debido a que el chip encargado de la interfaz Ethernet se inicializaba después de ejecutar el comando de ping. Este proceso generaba un retraso significativo en la ejecución de la prueba, por lo que implementé un cambio en el flujo de ejecución. Específicamente, modifiqué la secuencia de inicialización del chip Ethernet para que se llevara a cabo de manera concurrente con otros comandos, mediante la creación de un hilo para ahí ejecutarse esta inicialización. Esta optimización redujo a 11 segundos el tiempo de prueba.

Adicionalmente, observé que, al desactivar el Wi-Fi de la computadora, la latencia en la respuesta del ping disminuía. Esta mejora puede deberse a que, al desactivar el Wi-Fi, el sistema operativo de la computadora prioriza la interfaz Ethernet, optimizando el uso del recurso de red cableada y resultando en un intercambio de paquetes más eficiente.

Prueba T1L

Esta prueba solo se aplica a las variantes con protocolo de comunicación T1L.

Se requiere establecer una conexión LAN entre la computadora y el DBP, consultar el Error! Reference source not found. para obtener más detalles sobre la configuración de la PC.

En esta prueba, el FcT envía el comando <fft_ping><192.168.100.104><índice> para hacer ping al **DBP** a través del cable Eth y verifica si el ping fue exitoso.

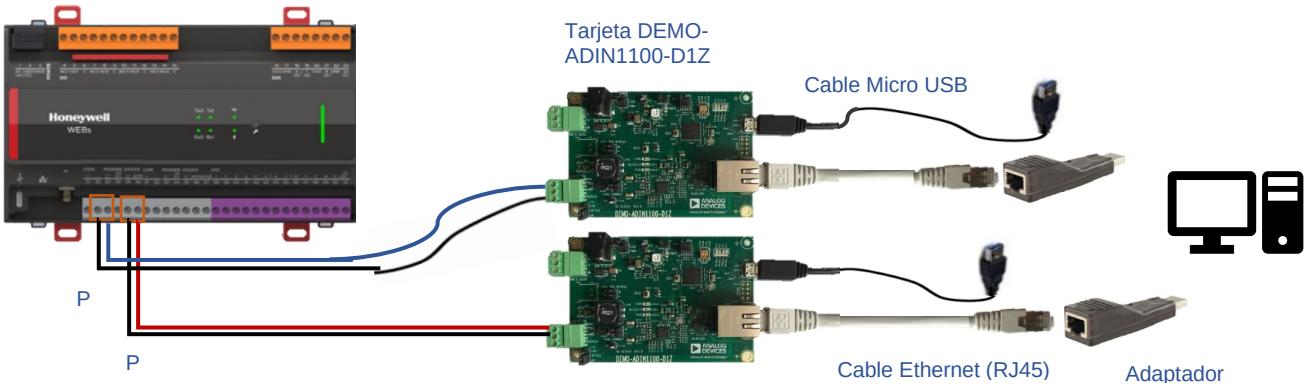

Figura 20: Conexión de la prueba T1L-1 con dos placas DEMO-ADIN1100-D1Z

Siguiendo el mismo enfoque de la definición de la prueba y la optimización que en la prueba Ethernet (ver sección 4.2.4.10), implementé la inicialización en paralelo mediante un hilo para inicializar el chip de T1L debido a que también este proceso tomaba mucho tiempo y desactivar el Wi-Fi para mejorar la latencia del ping.

Prueba MSTP

Esta prueba se aplica específicamente a las variantes que tienen el protocolo de comunicación MS/TP . El FcT envía <fft_mstp> para verificar la lectura/escritura usando este protocolo.

Tabla 13: Conexión para la prueba de MSTP

Tarjeta	RS485
P7-1	*Data* (+)
P7-2	*Data* (-)
P7-3	*GND*

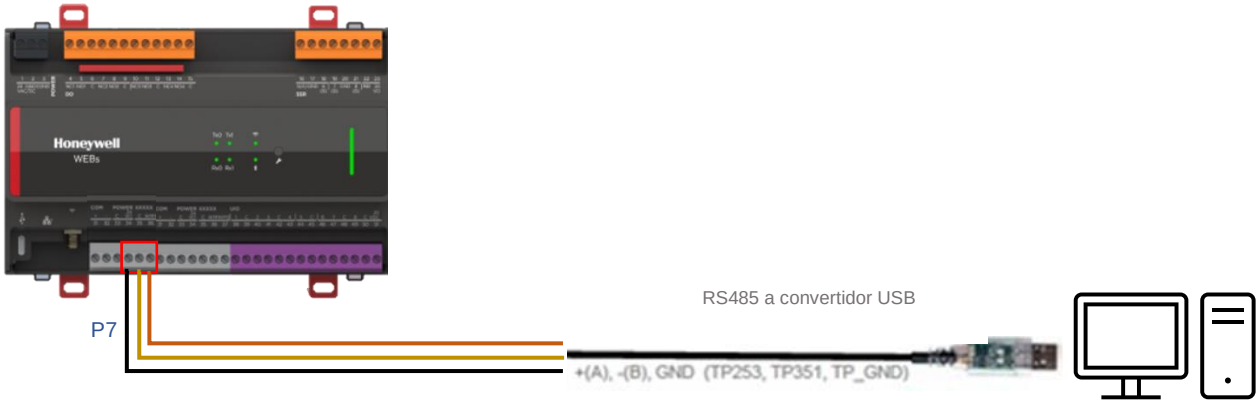

Figura 21: Conexión de la prueba MSTP

Para esta prueba, en colaboración con el arquitecto del controlador, definí un proceso similar al utilizado para la validación del puerto MODBUS, asegurando que la prueba fuese eficiente y rápida. Dado que ambos protocolos operan sobre RS485, el enfoque se centró en validar las capas físicas y de enlace de datos, evitando profundizar en las capas superiores del protocolo MS/TP, al igual que se hizo con Modbus.

La prueba consistió en el envío y recepción de cadenas de datos a través de la interfaz RS485, verificando así la integridad y fiabilidad en la transmisión y recepción de información. Esto permitió reducir el tiempo de validación en fábrica, garantizando que el puerto estuviera correctamente preparado para manejar las comunicaciones MS/TP en niveles superiores de la pila de comunicación.

Bandera de FFT

El FcT envía el comando <fft_timestamp_set><*TIMESTAMP*> para escribir en la memoria *FLASH* la cadena de texto especificada en el parámetro TIMESTAMP. En esta estación, el *TIMESTAMP* será `<*FFT_PASSED*>`, marcando así los controladores que pasaron esta etapa con éxito y permitiendo su avance a la siguiente.

4.3 Prueba de Radiofrecuencia (RFT)

4.3.1 Descripción

La estación 3 será necesaria para el desarrollo de esta etapa, donde se realizará la prueba de BLE. Si el modelo es BLE, este usará el:

- nRF52840 (BLE)

4.3.2 Requerimiento

Tabla 3: Requisitos de RFT para la estación 3.

Tipo	Requerimiento	Propósito
Hardware	Antena 2.4G	Prueba de Nordic BLE
Hardware	Analizador de espectro E6640A	Prueba BM25 y prueba Nordic BLE
Hardware	Antena plana TC-93026A	Prueba BM25 y prueba Nordic BLE

4.3.3 Vista general

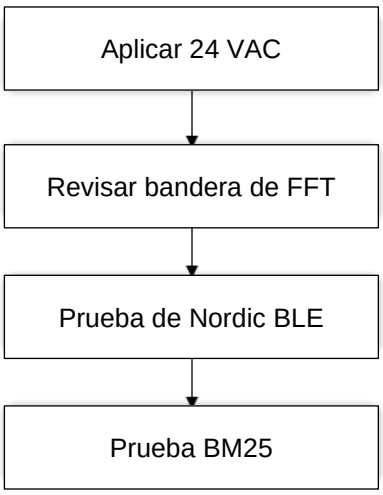

Figura 22: Descripción general del flujo de RFT

4.3.4 Procedimiento

Colocar el dispositivo en la fixture

Figura 23: Fixture para la estación 3 donde se realiza el RFT

Aplicar Voltaje

El FcT aplica 24VCA al DBP.

Revisar bandera FFT

El FcT envía el comando <fft_timestamp_get>. El DBP debe responder con <*FFT_PASSED*> para continuar con las pruebas.

Prueba Nordic BLE

Esta prueba verifica la compatibilidad y el funcionamiento de los dispositivos con el módulo BLE nRF52840. El código utilizado se basa en un ejemplo de Nordic Semiconductor denominado Radio Test [13].

Mi contribución consistió en adaptar este programa para su uso con el microcontrolador iMXRT1051, integrando una interfaz UART para la comunicación con el nRF52840.

Desde el lado del nRF52840, desarrollé un menú que permite la recepción de caracteres específicos para la configuración del módulo BLE. Este menú facilita la interacción y configuración del nRF52840, permitiendo a los usuarios establecer parámetros relevantes de manera sencilla. Adicionalmente, implementé comandos en el iMXRT1051 para enviar parámetros a través del bus UART hacia el nRF52840 y para esperar las respuestas del chip BLE. Esto asegura una comunicación bidireccional efectiva entre los dispositivos. A continuación, se presenta una imagen del menú:

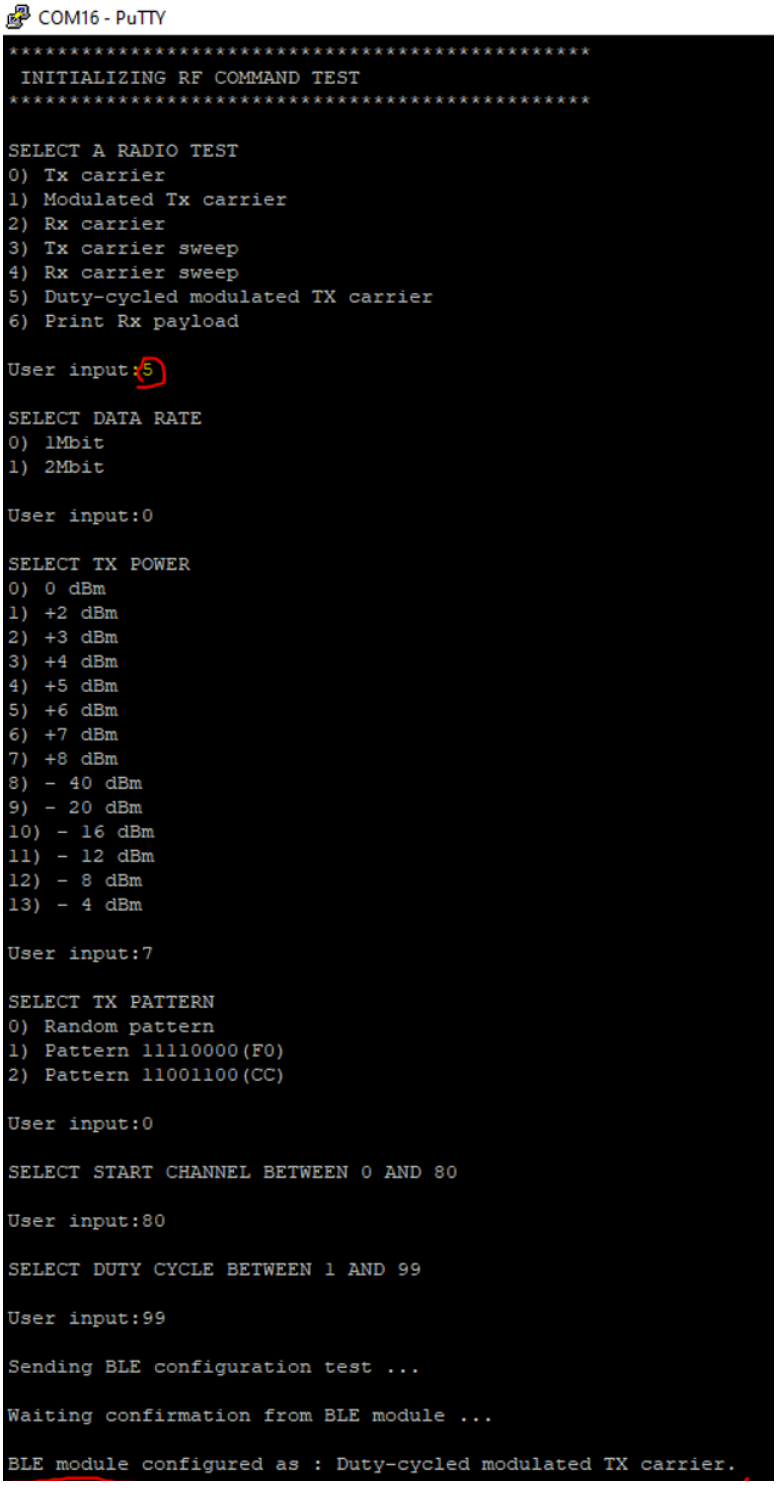

Figura 24: Ejemplo de configuración del stack del chip de BLE y el menú

Colaboré con una ingeniería de RF para definir las especificaciones de la prueba, incluyendo la selección de componentes como el analizador de espectro E6640A y los parámetros a medir en las señales de radio. Esto para establecer un *setup* de prueba adecuado que garantizara el rendimiento del dispositivo en entornos reales.

Para la prueba, se utilizó el analizador de espectro E6640A junto con antenas planas TC-93026 para realizar mediciones precisas de la señal BLE.

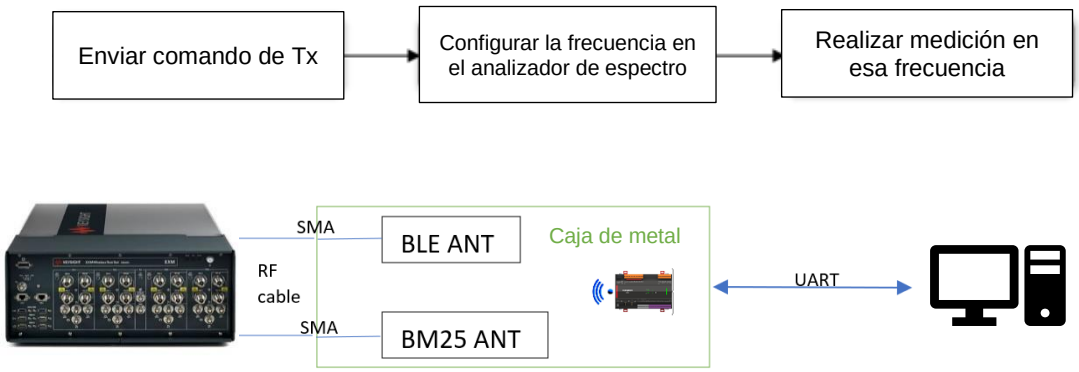

Figura 25: Prueba Nordic BLE

Prueba NORDIC BLE

El FcT envía un comando para Tx <fft_radio_nRF52840><TX><CANAL> donde CANAL es uno de los siguientes con su frecuencia correspondiente disponible:

Tabla 15: Canales y frecuencias permitidos para la prueba de NORDIC

CANAL	FRECUENCIA	POTENCIA
LOW	2.402 GHz	0 dBm
MIDDLE	2.442 GHz	0 dBm
HIGH	2.480 GHz	0 dBm

Requisitos considerados en las mediciones:

- Potencia promedio (en canales H, M y L)
- Precisión de frecuencia (ICFT) (en canales H, M y L)
- Deriva de frecuencia (en canales H, M y L)

Este procedimiento sigue una metodología similar a la utilizada en las certificaciones de RF, basada en estándares reconocidos, para garantizar la calidad y conformidad del dispositivo. Esta metodología garantiza mediciones precisas y consistentes, similar a lo que he experimentado en requerimientos para certificaciones de RF que se rigen bajo estándares de la FCC y la ETSI.

Bandera RF

El comando de envío Fct <fft_timestamp_set><*TIMESTAMP*><chksum> El comando escribe en la memoria *FLASH* la cadena del parámetro *TIMESTAMP*.

En esta estación *TIMESTAMP* será <*RF_PASSED*>.

4.4 Programación II

4.4.1 Descripción

El propósito de este proceso es programar al controlador con los firmwares de APP, tanto el microcontrolador iMXRT1051 y si es una variante con BLE, también programar el firmware de APP para el nRF52840. Esto se realiza en la estación 1, por lo que es una continuación de los bloques de pasos anteriores y regresando a esta estación.

4.4.2 Requerimiento

Tabla 16: Requisitos finales de la aplicación de criptografía y descarga para la estación 3

Tipo	Requerimiento	Propósito
Software	JFlash	Descarga de firmware para Nordic

Hardware	Programador J-Link	Descarga de firmware para Nordic
Hardware	Cable USB-USB Tipo C	Descarga de firmware para el microcontrolador MXRT

4.4.3 Vista general

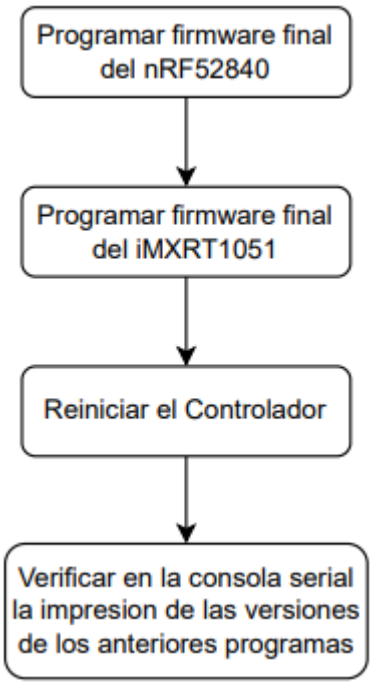

Figura 26: Flujo general de la Programación II

Escribir Sistema Operativo

El proceso de programación del controlador comienza con el apagado del DBP. Para habilitar la comunicación USB, se establece el controlador en modo serial. A continuación, se aplica 24VAC al DBP.

Se ejecuta un archivo por lotes que invoca Mfgtool.exe, herramienta del fabricante NXP, utilizada para descargar la imagen de la aplicación, de acuerdo con la variante específica. En caso de que la variante lo requiera, se ejecuta un archivo por lotes de *JFlash* del programador *Segger J-Link* para descargar el firmware de la aplicación Nordic. Luego de programar la aplicación para los chips MXRT y Nordic, es necesario realizar un ciclo de energía en el DBP. Mi contribución en esta etapa incluyó la coordinación con el arquitecto para asegurar la liberación de la última versión del código, la cual fue proporcionada al operador para su carga en los controladores.

Verificar Sistema Operativo

Al arrancar el controlador con el *firmware* de aplicación previamente cargado, el sistema utiliza la interfaz serial para mostrar comandos en la consola. En colaboración con el operador de la fábrica, identificamos que sería beneficioso verificar la versión de los programas para asegurar que se cargaron correctamente. Coordiné con el líder de desarrollo para que se imprimiera la versión de los programas del microcontrolador y, de ser pertinente, la versión de BLE. Este paso es utilizado para confirmar que el dispositivo ha sido programado adecuadamente y cumple con las especificaciones requeridas. La verificación de estos detalles contribuye a garantizar que el dispositivo esté listo para su uso y a identificar posibles problemas antes de que se considere completamente probado y listo para su envío. Este es el último paso para completar la prueba de FFT.

5 Conclusiones

El desarrollo e implementación de la FFT son fundamentales para asegurar la calidad y fiabilidad de los controladores, como el Optimizador Unitario 24V, en el proceso de manufactura. A través de las pruebas descritas en este trabajo, se validan componentes esenciales como la interfaz Ethernet, el protocolo T1L y otros sistemas de comunicación del hardware, garantizando que cumplen con los estándares técnicos necesarios para su funcionamiento en campo.

La FFT ha sido efectiva en identificar y corregir defectos de manera temprana, así como en integrar nuevos componentes de hardware mediante pruebas de *firmware* específicas que verifican su operatividad. Esto ayuda a minimizar fallos en un entorno real y a optimizar el tiempo y recursos en el proceso de manufactura, contribuyendo a mejorar la eficiencia del producto final.

En cuanto a microcontroladores, he adquirido experiencia práctica en el manejo de conceptos clave como GPIO, temporizadores, interrupciones, PWM y UART, aplicándolos en el diseño y pruebas de diferentes componentes del controlador. También he trabajado con la arquitectura ARM Cortex-M, familiarizándome con la gestión de relojes y protocolos de comunicación como I2C y SPI, lo cual fue esencial para la integración de nuevos módulos y la configuración de interfaces de comunicación en el sistema.

He utilizado estructuras de datos como buffers circulares y listas enlazadas para optimizar el manejo de datos en tiempo real durante las pruebas, asegurando la correcta captura y procesamiento de señales. Adicionalmente, poseo conocimientos básicos en sistemas operativos en tiempo real (RTOS), como la programación de tareas y la gestión de operaciones en tiempo real, los cuales apliqué para optimizar la eficiencia en las pruebas automatizadas y minimizar tiempos de respuesta críticos en el controlador.

Finalmente, trabajar en equipos globales me ha dado una visión más amplia de cómo se colabora en entornos multidisciplinarios, lo que me ha permitido integrar y coordinar diferentes enfoques para la resolución de problemas técnicos complejos. Esta experiencia me ha proporcionado un crecimiento integral, abarcando desde el diseño y validación de hardware hasta la optimización de *firmware* y la gestión de calidad, preparándome para enfrentar desafíos más avanzados en el desarrollo de sistemas embebidos.

Reflexiones Personales

Participar en el desarrollo y la implementación de las pruebas finales (FFT) me ha permitido aplicar y fortalecer una variedad de conocimientos técnicos en sistemas embebidos y electrónica. Durante el proyecto, he trabajado con circuitos eléctricos y electrónicos, lo que me ha dado la habilidad de leer esquemáticos y entender el funcionamiento de circuitos simples y complejos, garantizando la correcta integración de los componentes en el controlador Optimizador Unitario.

En el contexto del desarrollo, amplié mis conocimientos en lenguajes de programación como C, C++ y Python, profundizando en conceptos clave como el manejo de punteros y la gestión de memoria. Además, adquirí conocimientos en lenguaje ensamblador, lo cual me permitió realizar una depuración a bajo nivel y resolver problemas específicos durante el desarrollo del código.

En cuanto a microcontroladores, he adquirido una sólida base en arquitecturas ARM Cortex-M, dominando conceptos como GPIO, temporizadores, interrupciones, PWM y UART. Además, he trabajado con protocolos de comunicación como I2C y SPI para integrar diversos módulos y configurar interfaces.

He implementado estructuras de datos como buffers circulares y listas enlazadas para optimizar la recepción y procesamiento de comandos, asegurando su correcta captura y manejo en tiempo real. Además, amplié mis conocimientos en sistemas operativos en tiempo real utilizando FreeRTOS, aplicando la programación de tareas, temporizadores por software y la creación de hilos. Estas técnicas me permitieron mejorar la eficiencia del *firmware* en pruebas automatizadas y reducir los tiempos de respuesta del controlador.

Finalmente, trabajar en equipos globales me ha dado una visión más amplia de cómo se colabora en entornos multidisciplinarios, lo que me ha permitido integrar y coordinar diferentes enfoques para la resolución de diferentes problemas. Esta experiencia me ha proporcionado un crecimiento integral, abarcando desde el diseño y validación de hardware hasta la optimización de *firmware* y la gestión de

calidad, preparándome para enfrentar desafíos más avanzados en el desarrollo de sistemas embebidos en los siguientes años de mi carrera profesional.

6 Anexo

6.1 Terminología

Tabla 17: Acrónimos y abreviaciones

Abreviatura	Definición
DBP	Dispositivo bajo prueba
FcT	Estación de prueba funcional
FFT	Prueba Final de Fábrica
RFT	Prueba de Radio Frecuencia
PCB	Placa de Circuito Impreso
TP	*Test Point*
IP	*Protocolo de Internet*
T1L	*Twisted Pair LonWorks*
MS/TP	*Master-Slave/Token Passing*
SSR	*Relay de estado sólido*
NXP	*Next eXPerience*
TTL	Lógica Transistor-Transistor
I2C	*Inter-Integrated Circuit*
UART	*Universal Asynchronous Receiver-Transmitter*
BLE	Bluetooth de Baja Energía
SPI	*Serial Peripheral Interface*
T1L	*Single Pair Ethernet*
MS/TP	*Modbus Serial/Token Passing*
UIO	Entrada y Salida Universal
MODBUS	Protocolo de comunicación serial desarrollado por Schneider Electric para la comunicación en redes industriales
SYLK	Protocolo de comunicación para sistemas de control HVAC.

R&D	Investigación y desarrollo
BI	Entrada Binaria
BO	Salida Binaria
QSPI	Interfaz Periférica Serial Cuádruple
SDRAM	*Memoria Dinámica de Acceso Aleatorio Síncrono*
SN	*Número Serial*
FCC	*Federal Communications Commission*
ETSI	*European Telecommunications Standards Institute*
SDIO	*Entrada y Salida Digital Segura*
OS	*Sistema Operativo*
VOLOUT	Modo de Voltaje de Salida en UIO
CUROUT	Modo de Corriente de Salida en UIO
VOLIN	Modo de lectura de Voltaje en UIO
CURIN	Modo de lectura de Corriente en UIO
RESIN	Modo de lectura de Resistencia en UIO
APP	Firmware de aplicación.
JTAG	*Joint Test Action Group*
Fixture	Dispositivo que automatiza el envío de comandos y prueba las especificaciones requeridas de las pruebas
BA	*Building Automation*

6.2 Referencias

[1] White, E. (2011). *Making Embedded Systems: Design Patterns for Great Software*. O'Reilly Media, p. 5.

[2] White, E. (2011). *Making Embedded Systems: Design Patterns for Great Software*. O'Reilly Media, p. 7.

[3] Barr, M., & Massa, A. (2006). *Programming Embedded Systems: With C and GNU Development Tools*. O'Reilly Media.

[4] Wolf, W. (2001). *Computers as Components: Principles of Embedded Computing System Design*. Elsevier.

[5] Honeywell. *100+ Years of Transforming Industries*. Recuperado de https://www.honeywell.com/us/en/news/100-years-of-transforming-industries

[6] Honeywell.(2024). *About us*. Recuperado de https://www.honeywell.com/mx/es/company/about-us

[7] Honeywell. (2024). *Honeywell Global Locations.* Recuperado de https://sps.honeywell.com/us/en/support/global-locations

[8] Honeywell. (2024). *Our History.* Recuperado de https://www.honeywell.com/us/en/company/our-history

[9] Honeywell. (2024). *Honeywell Optimizer Unitario 24V: Hoja de datos. Recuperado de https://prod-edam.honeywell.com/content/dam/honeywell-edam/hbt/en-us/documents/literature-and-specs/datasheets/hbt-bms-Unitary-Controller-24V-31-00613-ESP-Datasheet.pdf?download=false, p. 1.

[10] Honeywell. (2024). *Honeywell Optimizer Unitario 24V: Hoja de datos. Recuperado de https://prod-edam.honeywell.com/content/dam/honeywell-edam/hbt/en-us/documents/literature-and-specs/datasheets/hbt-bms-Unitary-Controller-24V-31-00613-ESP-Datasheet.pdf?download=false, p. 4.

[11] NXP Semiconductors. (2023). i.MXRT1051 Reference Manual (IMXRT1051RM). [PDF]. Recuperado de https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/i-mx-rt-crossover-mcus/i-mx-rt1050-crossover-mcu-with-arm-cortex-m7-core:i.MX-RT1050

[12] Analog Devices. (2019). *AD74412R: Precision Analog Front End with Dual ADCs and Multiple I/O Options*. Recuperado de https://www.analog.com/media/en/technical-documentation/data-sheets/AD74412R.pdf

[13] Nordic Semiconductor. (2020, September 14). *nRF5 SDK v17.0.2: nRF Radio Test Example*. Recuperado de https://docs.nordicsemi.com/bundle/sdk_nrf5_v17.0.2/page/nrf_radio_test_example.html