UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Desarrollo Practico de Firmware y Herramientas de Control para una Tarjeta Electrónica

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Telecomunicación

AUTOR/A: Santos Martín, Alejandro

Tutor/a: Gadea Gironés, Rafael

CURSO ACADÉMICO: 2023/2024

Resumen

El presente Trabajo de Fin de Máster se centra en el desarrollo integral de un firmware destinado a una tarjeta electrónica que desempeñará un papel fundamental como componente principal en la mayoría de los equipos manufacturados por la empresa PolymerChar. Este proyecto comprende, además, la creación de una Dynamic-Link Library (DLL) que posibilitará el acceso a todas las funcionalidades desde un PC. La fase final del trabajo implica la implementación de una aplicación que permitirá a los empleados encargados del testeo o mantenimiento de los equipos tener un control exhaustivo sobre la tarjeta electrónica, accediendo a todos sus registros para evaluar el estado del equipo o llevar a cabo pruebas específicas.

La ejecución de este proyecto requiere un profundo conocimiento del componente hardware de la tarjeta. Se llevará a cabo un exhaustivo testeo de los recursos decididos para su incorporación, como pueden ser los buses de comunicación CAN, I2C, SPI, PWMs generadas desde un microcontrolador, los pines de propósito general (GPIO), el correcto funcionamiento de cada uno de los ADCs que se utilizarán para conocer el estado de cada uno de los sensores que poseerá la tarjeta...

El microcontrolador elegido para este proyecto pertenece a la familia STM32, específicamente el modelo STM32F407VG, y el desarrollo del firmware se llevará a cabo en el entorno propio de STM32 en el lenguaje C. La creación de la DLL y la aplicación de depuración se llevarán a cabo en el entorno de Visual Studio en lenguaje C#. Este enfoque integral garantizará la coherencia y eficacia en el desarrollo y funcionamiento de la tarjeta electrónica, así como en su interacción con los equipos fabricados por PolymerChar.

Palabras Clave: Firmware, Dynamic-Link Library (DLL), Microcontrolador, STM32F407VG, Visual Studio, CAN, I2C, SPI, Tarjeta Electrónica, Lenguaje C.

Abstract

The present Master's Thesis focuses on the comprehensive development of firmware intended for an electronic board that will play a crucial role as the main component in most of the equipment manufactured by the company PolymerChar. This project also involves creating a Dynamic-Link Library (DLL) to enable access to all functionalities from a PC. The final phase of the work entails implementing an application that will allow employees responsible for testing or maintaining the equipment to have exhaustive control over the electronic board. This involves accessing all its registers to assess the equipment's status or conduct specific tests.

Executing this project requires a deep understanding of the hardware component of the board. A thorough testing of the decided resources for incorporation will be carried out. This includes communication buses such as CAN, I2C, SPI, PWMs generated from a microcontroller, General Purpose Input/Output (GPIO) pins, and ensuring the proper functioning of each of the Analog-to-Digital Converters (ADCs) used to monitor the status of each sensor on the board.

The selected microcontroller for this project belongs to the STM32 family, specifically the STM32F407VG model. Firmware development will take place in the STM32 environment using the C language. The creation of the DLL and debugging application will be done in the Visual Studio environment using the C# language. This comprehensive approach ensures coherence and effectiveness in the development and operation of the electronic board, as well as its interaction with the equipment manufactured by PolymerChar.

Key Words: Firmware, Dynamic-Link Library (DLL), Microcontroller, STM32F407VG, Visual Studio, CAN, I2C, SPI, Electronic Board, C Language.

Resum

El present Treball de Fi de Màster es centra en el desenvolupament integral d'un firmware destinat a una targeta electrònica que jugarà un paper fonamental com a component principal en la majoria dels equips fabricats per l'empresa PolymerChar. A més, aquest projecte comprén la creació d'una Dynamic-Link Library (DLL) que possibilitarà l'accés a totes les funcionalitats des d'un PC. La fase final del treball implica la implementació d'una aplicació que permetrà als empleats encarregats del test o manteniment dels equips tenir un control exhaustiu sobre la targeta electrònica, accedint a tots els seus registres per avaluar l'estat de l'equip o dur a terme proves específiques. L'execució d'aquest projecte requereix un coneixement profund del component hardware de la targeta. Es durà a terme una prova exhaustiva dels recursos decidits per a la seua incorporació, com poden ser els buses de comunicació CAN, I2C, SPI, PWMs generades des d'un microcontrolador, els pins de propòsit general (GPIO), el correcte funcionament de cadascun dels ADCs que es faran servir per conèixer l'estat de cadascun dels sensors que tindrà la targeta... El microcontrolador triat per a aquest projecte pertany a la família STM32, específicament el model STM32F407VG, i el desenvolupament del firmware es realitzarà en l'entorn propi de STM32 en el llenguatge C. La creació de la DLL i l'aplicació de depuració es realitzaran en l'entorn de Visual Studio en el llenguatge C#. Aquest enfocament integral garantirà la coherència i l'eficàcia en el desenvolupament i funcionament de la targeta electrònica, així com en la seua interacció amb els equips fabricats per PolymerChar

Paraules Clau: Firmware, Dynamic-Link Library (DLL), Microcontrolador, STM32F407VG, Visual Studio, CAN, I2C, SPI, Targeta Electrònica, Llenguatge C.

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

RESUMEN EJECUTIVO

La memoria del TFM del Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación debe desarrollar en el texto los siguientes conceptos, debidamente justificados y discutidos, centrados en el ámbito de la ingeniería de telecomunicación

CONCEPT (ABET)	**CONCEPTO (traducción)**	**¿Cumple? (S/N)**	**¿Dónde? (páginas)**
1. IDENTIFY:	1. IDENTIFICAR:		
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	S	1, 6 - 7
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	S	7 - 9
1.3. Setting of goals	1.3. Establecimiento de objetivos	S	2 - 3
2. FORMULATE:	2. FORMULAR:		
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	S	19 - 30 44 - 47
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	S	10 - 17
3. SOLVE:	3. RESOLVER:		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	S	51
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	S	51 1 - 2

Escuela Técnica Superior de Ingeniería de Telecomunicación

Universitat Politècnica de València

Edificio 4D. Camino de Vera, s/n, 46022 Valencia

Tel. +34 96 387 71 90, ext. 77190

www.etsit.upv.es

Índice

Índice de Ilustraciones

Índice de Tablas

Capítulo I: Introducción

1.1 Contexto del proyecto

El **Trabajo de Fin de Máster (TFM)** se lleva a cabo en la empresa Polymer Char[1], una compañía tecnológica española dedicada a la fabricación de instrumentación analítica para polímeros. Este proyecto surge en un contexto donde la empresa ha decidido internalizar el desarrollo de la electrónica de sus equipos, un proceso que antes estaba subcontratado. Esta decisión estratégica permite a la empresa tener un mayor control sobre el diseño, evolución y personalización de sus productos, adaptándose mejor a las necesidades del mercado y asegurando la calidad y confidencialidad de su tecnología.

El proyecto se centra en el desarrollo integral de un firmware para una tarjeta electrónica denominada **New Main Board (NMB)**, que será un componente clave en los equipos de Polymer Char. Además, el proyecto incluye la creación de una **Dynamic-Link Library (DLL)** para facilitar la comunicación entre el firmware y las aplicaciones de software, y el desarrollo de una **aplicación de testeo** con una interfaz de usuario intuitiva que permita a los empleados verificar y mantener el correcto funcionamiento de la tarjeta.

1.2 Motivación y Relevancia

La motivación principal de este Trabajo de Fin de Máster (TFM) radica en la imperiosa necesidad de asegurar la calidad y la funcionalidad de los dispositivos electrónicos desde las primeras etapas de su desarrollo. Un firmware bien diseñado es fundamental para el rendimiento y la estabilidad del sistema, mientras que una DLL eficaz proporciona una interfaz de comunicación esencial entre el hardware y las aplicaciones de alto nivel. Además, una aplicación dotada de una interfaz de usuario (UI) intuitiva y funcional no solo optimiza el proceso de testeo, sino que también enriquece la experiencia del usuario, permitiendo una interacción más directa y comprensible con el dispositivo.

Este proyecto adquiere una relevancia particular en un entorno donde la velocidad de desarrollo y la fiabilidad del producto final son factores críticos. En este sentido, la capacidad de desarrollar, probar y validar estos dispositivos de manera eficiente se convierte en un objetivo estratégico de gran importancia. En línea con la ambición de la empresa de mantener una constante mejora en sus productos, actualizando la electrónica de sus equipos, este TFM también aborda la evolución interna de la compañía en cuanto al desarrollo de su tecnología. Inicialmente, la parte electrónica de los equipos era subcontratada, pero con el tiempo, la empresa ha decidido internalizar este proceso, lo que ha demostrado ser una decisión clave para su crecimiento y competitividad.

Al internalizar el desarrollo de la electrónica, la empresa ha obtenido un control total sobre el diseño y la evolución de sus productos, lo que le permite una mayor personalización para satisfacer las necesidades específicas del mercado y de sus clientes. Esta autonomía facilita la realización de ajustes y modificaciones en cualquier etapa del proyecto sin depender de los plazos o procesos de un proveedor externo, lo que acelera el desarrollo y reduce significativamente el tiempo de lanzamiento al mercado. Además, al mantener el desarrollo dentro de la empresa, se protege de manera más eficaz la propiedad intelectual, minimizando el riesgo de filtraciones de información sensible que podrían ocurrir con la subcontratación. Este control directo también simplifica la gestión legal y administrativa, al eliminar la necesidad de negociar complejos acuerdos sobre derechos de propiedad.

A largo plazo, aunque la inversión inicial para internalizar el desarrollo puede ser considerable, los costos recurrentes se reducen significativamente al no tener que pagar los márgenes de beneficio de un tercero, lo que se traduce en ahorros importantes, especialmente cuando se optimizan y reutilizan diseños y componentes. La internalización también fortalece la colaboración y la comunicación entre el equipo de desarrollo de electrónica y otros departamentos de la empresa, favoreciendo una integración más eficiente de los diferentes componentes del producto. Esta sinergia no solo mejora la calidad del producto final al permitir un control más estricto sobre cada etapa del proceso, sino que también fomenta la innovación continua, ya que el equipo dedicado tiene un conocimiento profundo del producto y puede identificar oportunidades de mejora o desarrollo de nuevas funcionalidades. Finalmente, gestionar la cadena de suministro de manera directa permite a la empresa seleccionar proveedores de manera más estratégica y controlar mejor los plazos de entrega, lo que reduce el riesgo de retrasos y garantiza un flujo de producción más predecible y confiable. Este enfoque integral y controlado posiciona a la empresa en una situación ventajosa, permitiéndole responder con agilidad a las demandas del mercado y asegurar la calidad de sus productos.

1.3 Objetivos del proyecto

El principal objetivo de este TFM es diseñar y desarrollar un sistema completo que permita el desarrollo y testeo de una tarjeta electrónica. Para lograr esto, se han establecido los siguientes objetivos específicos:

1 ***Desarrollo Firmware:***

- Diseñar y programar un firmware que controle las funcionalidades básicas y avanzadas de la tarjeta electrónica.
- Asegurar que el firmware sea modular y fácilmente actualizable.

2 ***Creación de la DLL:***

- Desarrollar una DLL que permita la comunicación entre el firmware de la tarjeta electrónica y las aplicaciones de software de alto nivel.
- Garantizar que la DLL sea eficiente, segura y fácil de integrar con otras aplicaciones.

3 ***Diseño de la Aplicación con UI:***

- Desarrollar una aplicación con una interfaz de usuario intuitiva que permita el testeo y la validación de la tarjeta electrónica.
- Incluir funcionalidades que faciliten el diagnóstico de problemas y la evaluación del rendimiento del dispositivo.

4 ***Integración y Testeo Completo del Sistema:***

- Integrar el firmware, la DLL y la aplicación en un entorno cohesivo y funcional.
- Realizar pruebas exhaustivas para validar la funcionalidad y la estabilidad del sistema completo.

1.4 Objetivos de desarrollo sostenible

El objetivo principal de los Objetivos de Desarrollo Sostenible (ODS)[2] es proporcionar un marco global para abordar los desafíos más urgentes que enfrenta la humanidad, con el fin de lograr un desarrollo sostenible en todas sus dimensiones: económica, social y ambiental. Adoptados en 2015 por todos los Estados miembros de la Organización de las Naciones Unidas (ONU), los ODS forman parte de la Agenda 2030 para el Desarrollo Sostenible y buscan erradicar

la pobreza, proteger el planeta y garantizar que todas las personas puedan disfrutar de paz y prosperidad.

Para ello se propuso una lista de 17 objetivos para la Agenda 2030 como se ha comentado, de los cuales en este trabajo de fin de máster se ha hecho hincapié en los siguientes:

- ***Ods9: Industria, Innovación e infraestructura.***
  Un firmware bien diseñado en estos equipos de instrumentación analítica es esencial para optimizar su rendimiento y garantizar su fiabilidad, lo que contribuye a la creación de infraestructuras tecnológicas avanzadas y eficientes. Además, al impulsar la innovación en el diseño y desarrollo de firmware, se promueve una industrialización más sostenible, mejorando la precisión y eficiencia en procesos analíticos críticos que apoyan la investigación y el desarrollo.

- ***Ods12: Producción y Consumo Responsable.***
  Este objetivo se enfoca en garantizar modalidades de consumo y producción sostenibles. Mediante un firmware eficiente se puede optimizar el uso de recursos en estos equipos mejorando el uso de rendimiento energético y reduciendo el desperdicio de materiales. Esto no solo prolonga la vida útil de equipo, sino que también minimiza el impacto ambiental asociado con su operación y mantenimiento.

- ***Ods8: Trabajo decente y crecimiento económico.***
  El motivo de este objetivo es promover el crecimiento económico sostenido, el empleo pleno y productivo y el trabajo decente para todos, con relación a este TFM si el firmware está bien diseñado mejora la eficiencia y precisión del equipo, lo que puede aumentar la productividad. Esto, a su vez, impulsa el crecimiento económico al facilitar procesos más efectivos y de alta calidad

Capítulo II: Diagrama Temporal

En el trabajo de fin de máster, se incluye un diagrama temporal para proporcionar una visión estructurada y organizada del proceso de desarrollo del proyecto. Este diagrama no solo facilita la planificación y la gestión del tiempo, sino que también asegura que todas las fases del proyecto, desde la definición inicial hasta la entrega final, se ejecuten de manera secuencial y coordinada.

Las tareas marcadas como puntos de fases del proyecto son las siguientes:

1 Definición del Proyecto y Planificación

En esta fase inicial, se definen claramente los objetivos y el alcance del proyecto, así como se identifican los requisitos específicos tanto para el firmware como para la aplicación de testeo. Esto incluye realizar una investigación preliminar sobre el hardware y las herramientas disponibles, y documentar las especificaciones técnicas del sistema. Además, se crea un cronograma detallado con hitos importantes y fechas de entrega, para asegurar una planificación efectiva del tiempo y los recursos.

2 Revisión del Diseño

Durante la fase de revisión del diseño, se analizan y ajustan los planes del firmware y la aplicación de testeo para garantizar que cumplen con las especificaciones y requisitos del proyecto. Esto implica revisar la arquitectura propuesta, los diagramas de flujo, y los detalles de la implementación para identificar y corregir posibles inconsistencias o problemas antes de iniciar el desarrollo.

3 Verificación del Hardware

La verificación del hardware es una fase crítica en la que se realizan pruebas iniciales para asegurarse de que la tarjeta electrónica y sus componentes funcionan correctamente. Esto incluye la comprobación de la conectividad, la funcionalidad básica de los componentes, y la confirmación de que el hardware está en condiciones de soportar el desarrollo del firmware. Cualquier problema detectado en esta etapa debe ser solucionado antes de comenzar el desarrollo del firmware.

4 Desarrollo del Firmware

En la fase de desarrollo del firmware, se procede con el diseño y la codificación lógica que controlará el hardware. Esto incluye la implementación de las funciones básicas, la realización de pruebas unitarias para cada módulo, y la integración de los diferentes componentes del firmware. La optimización del código y los ajustes basados en las pruebas también son parte de esta fase. El objetivo es garantizar que el firmware funcione de manera eficiente y confiable con el hardware.

5 Desarrollo de la Aplicación de Testeo

Durante el desarrollo de la aplicación de testeo, se diseñará y codificará el software necesario para probar y validar el funcionamiento del firmware y del hardware. Esta fase incluye la creación de la interfaz de usuario, la implementación de las pruebas específicas y la integración de la aplicación con el firmware. Es importante realizar pruebas de funcionalidad exhaustivas para asegurarse de que la aplicación cumple con los requisitos y puede detectar correctamente cualquier problema.

6 Integración y Pruebas Conjuntas

La fase de integración y pruebas conjuntas se centra en combinar el firmware y la aplicación de testeo para verificar que funcionan de manera conjunta sin problemas. Se realizarán pruebas de sistema completas para evaluar la interacción entre el firmware y la aplicación, así como la respuesta del hardware. Esta etapa también incluye la depuración de cualquier problema encontrado durante las pruebas y la realización de ajustes necesarios para asegurar un funcionamiento fluido y correcto.

7 Pruebas y Validación Final

En esta fase final de pruebas y validación, se llevan a cabo pruebas exhaustivas del sistema completo para asegurar que todos los componentes funcionen de acuerdo con las especificaciones del proyecto. Esto incluye la validación del rendimiento, la estabilidad y la fiabilidad del sistema en su conjunto. Cualquier ajuste necesario se realiza para resolver problemas identificados durante estas pruebas finales, garantizando que el sistema esté listo para la entrega.

8 Documentación

La documentación y preparación de la presentación es el proceso de crear y organizar toda la información técnica y de proyecto necesaria para la entrega final. Esto incluye la redacción del informe técnico detallado que cubre el diseño, la implementación y las pruebas realizadas, así como la preparación de una presentación visual que resuma los aspectos clave del proyecto.

9 Revisión y Ajustes Finales

En esta etapa, se revisa y ajusta el informe final y la documentación técnica basándose en cualquier feedback recibido de asesores o compañeros. Se realizan las correcciones necesarias y se asegura que todo el material esté completo y en el formato adecuado para la entrega final. También se realizan preparativos adicionales para la presentación, garantizando que el proyecto esté listo para su evaluación final.

10 Buffer

La fase de buffer proporciona un período adicional para abordar cualquier imprevisto o problema que pueda surgir antes de la entrega final. Este tiempo adicional permite hacer ajustes finales y resolver cualquier asunto pendiente, asegurando que el proyecto esté en su mejor forma antes de la entrega y presentación final. Este margen de tiempo es crucial para manejar cualquier contingencia de última hora.

A continuación, se presenta Diagrama de Gantt que ofrece una representación visual del desarrollo de las tareas descritas a lo largo del tiempo. Este diagrama ilustra claramente la secuencia y la duración de cada fase del proyecto, que ha abarcado aproximadamente seis meses de trabajo. La extensión temporal y el esfuerzo invertido aseguran que se cumplan los requisitos necesarios para

la realización del Trabajo de Fin de Máster (TFM), garantizando así una planificación y ejecución adecuadas del proyecto.

TAREAS A REALIZAR	FEBRERO	MARZO	ABRIL	MAYO	JUNIO	JULIO	AGOSTO	SEPTIEMBRE
Definicion del proyecto y planificacion								
Revision del diseño								
Verificacion del diseño Hardware								
Desarrollo del firmware								
Desarrollo de la aplicación de testeo								
Integracion y pruebas conjuntas								
Pruebas y validacion final								
Documentacion								
Revision y ajustes finales								
Buffer								

1. Diagrama de Gantt

Capítulo III: Marco Teórico

En este capítulo se analizará el estado actual de las tecnologías utilizadas tanto en la empresa donde se ha desarrollado este proyecto de fin de máster como en el mercado general.
Los equipos de instrumentación analítica están equipados con una amplia gama de sensores y detectores, diseñados para capturar la mayor cantidad de información posible sobre los compuestos a analizar. Este enfoque permite obtener resultados con un alto grado de precisión y exactitud, lo cual es esencial para garantizar la fiabilidad y calidad en los procesos de análisis.

1. GPC-IR Polymer Char

Como ejemplo de instrumentación analítica aplicada en Polymer Char, se ha seleccionado el equipo denominado GPC-IR[3] (1. GPC-IR Polymer Charmás atrás), un instrumento GPC multidetector especializado en el análisis de poliolefinas. En estos equipos se encuentran diferentes tipos de detectores, entre los que se incluyen:

Detector de infrarrojos[4], utilizado para determinar la concentración y composición de un polímero mediante la medición de la absorbancia a diferentes longitudes de onda.

Detector de viscosidad, este detector mide la viscosidad de los polímeros al hacerlos pasar por tubos capilares, donde se registran valores de presión. Estos datos, junto con otros parámetros como las dimensiones de los tubos, permiten calcular la viscosidad tras un proceso de post-procesado de los datos.

Detector de dispersión de luz, este detector emplea una técnica que mide el peso molecular absoluto, basado en la relación entre la intensidad de la luz dispersada por una molécula y su peso y tamaño molecular.

Para que todos estos detectores funcionen en conjunto, es necesaria una electrónica que indique el inicio y el fin de cada tipo de inyección, lo que permite obtener datos con mayor precisión y exactitud. La precisión y exactitud son fundamentales en la instrumentación analítica.

Además, la sincronización de los datos de todos estos detectores facilita a los usuarios o técnicos del equipo la detección de posibles anomalías.

Este tipo de equipos están equipados con tecnologías avanzadas, incluyendo diversos sensores que garantizan que las condiciones de trabajo se mantengan estables o, al menos, bajo control.

3.1 Estado Actual

Actualmente, los equipos de la compañía están equipados con un conjunto de tarjetas electrónicas cuyo desarrollo de hardware y firmware ha sido históricamente subcontratado. Estas tarjetas son responsables de controlar la sincronización de todos los detectores, recopilar información de los sensores integrados y mantener la comunicación entre los diferentes módulos electrónicos y el PC conectado al equipo.

3.2 Tecnología Empleada

3.2.1 STMicroelectronics

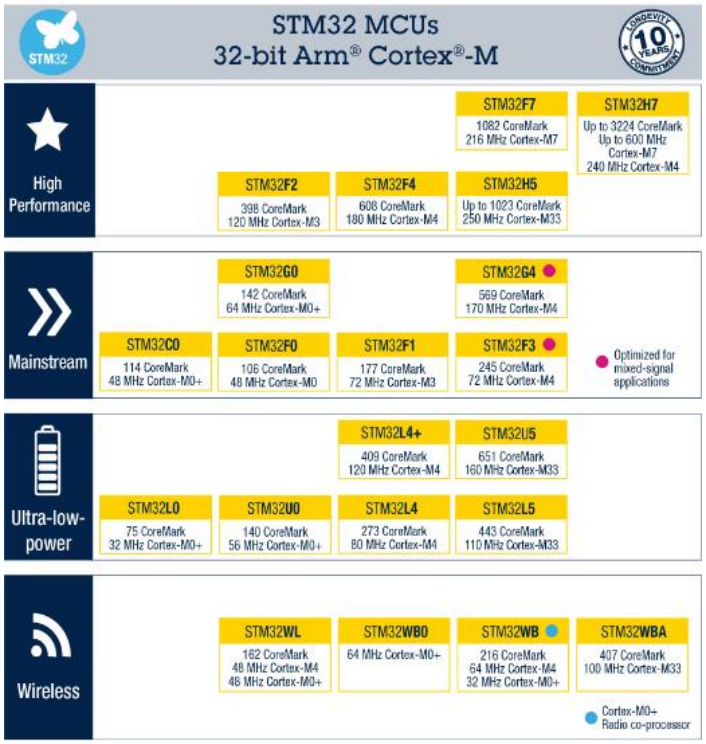
Para la parte de desarrollo firmware se ha escogido trabajar con un microcontrolador de la compañía STMicroelectronics[5] por diversas razones claves.

En primer lugar, STMicroelectronics ofrece una amplia gama de microcontroladores, particularmente en la familia STM32, que abarca desde dispositivos de bajo costo y bajo consumo hasta microcontroladores de alto rendimiento. Esta variedad permite seleccionar un microcontrolador que se ajuste de manera precisa a los requerimientos específicos del proyecto.

Además, STMicroelectronics proporciona un ecosistema de desarrollo completo y robusto, con herramientas como el entorno de desarrollo integrado STM32CubeIDE y la biblioteca STM32CubeMX, que facilitan considerablemente el desarrollo, depuración y optimización del firmware. La disponibilidad de librerías y middleware predefinidos también permite una integración rápida de funciones avanzadas como USB, Ethernet y control de motores, lo que acelera el desarrollo del proyecto.

En cuanto al rendimiento, los microcontroladores STM32, basados en núcleos ARM Cortex-M, se destacan por su equilibrio entre potencia de procesamiento y eficiencia energética, características esenciales en aplicaciones embebidas que requieren alto rendimiento con consumo controlado. Los modos de bajo consumo de estos microcontroladores son especialmente útiles en aplicaciones donde la eficiencia energética es una prioridad.

Otro aspecto relevante es la integración de periféricos y soporte para comunicaciones avanzadas. Los microcontroladores STM32 incluyen una rica variedad de periféricos integrados, como ADCs, DACs, UART, SPI e I2C, lo que proporciona flexibilidad en el diseño del sistema embebido. Además, el soporte para tecnologías de comunicación avanzadas como CAN, USB, Ethernet y BLE facilita la integración en redes y sistemas más complejos.



2. Familia de STMicroelectronics

Para esta electrónica se ha escogido la familia STM32F4 de STMicroelectronics ya que es ideal para proyectos que requieren un equilibrio entre alto rendimiento y costo. Basada en núcleos ARM Cortex-M4 con capacidades de procesamiento en punto flotante y una velocidad de reloj de hasta 180 MHz, esta serie es adecuada para aplicaciones exigentes en tiempo real. Además, ofrece una amplia gama de periféricos avanzados, gran capacidad de memoria, y excelente conectividad, lo que la hace más versátil que otras series más básicas. Su popularidad en la comunidad garantiza acceso a recursos y soporte, facilitando el desarrollo y reduciendo tiempos de implementación.

3.2.2 Visual Studio

Para el desarrollo de las herramientas de control se optado por utilizar la aplicación de Visual Studio[6], este software proporciona un entorno de desarrollo integrado (IDE) robusto y versátil, que soporta una amplia gama de lenguajes de programación, incluyendo C#, que es el lenguaje seleccionado para desarrollar todo el código de esta aplicación. Su potente editor de código, con características como el autocompletado, la navegación eficiente del código y la refactorización facilita la escritura y el mantenimiento del código.

Además, Visual Studio integra herramientas avanzadas de depuración y análisis, que permiten identificar y corregir errores de manera más eficiente. Gracias a la visualización detallada de datos en tiempo real, mejora significativamente la calidad del software y acelera el proceso de desarrollo.

Por otro lado, una cualidad que posee esta aplicación es la compatibilidad con Git, este es un sistema de control de versiones distribuido que permite gestionar y realizar un seguimiento de los cambios en el código fuente de un proyecto, además de facilitar la colaboración entre múltiples desarrolladores al permitir que cada uno trabaje en sus propias ramas del proyecto de manera

independiente, fusionando sus cambios de forma controlada. Git registra el historial de modificaciones, lo que permite revertir a versiones anteriores si es necesario, y asegura que todas las contribuciones se integren de manera eficiente y sin conflictos.

3.2.3 WPF vs Windows Forms

En la compañía históricamente en el departamento de electrónica, para desarrollar las aplicaciones de depuración de las electrónicas se ha utilizado la herramienta de Visual Studio, Windows Forms, pero para esta nueva electrónica se optado por escoger WPF, ya que esta proporciona una mayor flexibilidad y potencia en la creación de interfaces de usuario ricas y modernas. Utiliza XAML (eXtensible Application Markup Language) para definir interfaces, lo que permite separar claramente la lógica de la presentación y facilita el diseño de interfaces más dinámicas y estilizadas. Esta capacidad de personalización es mucho más limitada en Windows Forms, que se basa en controles más rígidos y menos personalizables.

Además, WPF soporta gráficos vectoriales, lo que permite que las interfaces se escalen sin pérdida de calidad, haciendo que las aplicaciones luzcan nítidas en pantallas de alta resolución. También ofrece un mejor rendimiento en la renderización de gráficos complejos y soporta animaciones y efectos visuales avanzados, algo que es más complicado de implementar en Windows Forms.

WPF también facilita el desarrollo de aplicaciones con patrones de diseño modernos, como MVVM (Model-View-ViewModel), que mejoran la mantenibilidad y escalabilidad del código. En comparación, Windows Forms es más adecuado para aplicaciones más simples y rápidas de implementar, pero carece de la capacidad de adaptarse a las necesidades de interfaces más complejas y avanzadas que WPF maneja con facilidad

WPF	Windows Forms
• Es más nuevo y, por lo tanto, está más en consonancia con los estándares actuales	• Tiene mas tiempo por lo que ha sido mas utilizado y probado
• Es más flexible, por lo que puede hacer más cosas sin tener que escribir o comprar nuevos controles	• Ya hay muchos controles de empresas de terceros que puedes comprar o conseguir gratis
• Cuando necesite utilizar controles de terceros, es probable que los desarrolladores de estos controles se enfoquen más en WPF porque es más reciente	• El diseñador en Visual Studio sigue siendo, en cuanto a la escritura, mejor para WinForms que para WPF, donde tendrá que hacer más trabajo usted mismo con WPF
• XAML hace que sea fácil crear y editar su GUI, y permite que el trabajo se divida entre un diseñador (XAML) y un programador (C#, VB.net etc.)	
• Enlace de datos (Databinding), le permite obtener una separación más limpia de los datos y el diseño	
• Utiliza la aceleración de hardware para dibujar la GUI, para un mejor rendimiento	

2.WPF vs Windows Forms [7]

Capitulo IV: Desarrollo del proyecto

El desarrollo de esta nueva electrónica surge con el objetivo de crear una solución propia dentro de la empresa, destinada a sustituir el conjunto de tarjetas electrónicas subcontratadas, tal como se mencionó anteriormente. El primer paso en este proceso consistió en listar exhaustivamente todas las funciones que realizaban las tarjetas electrónicas existentes y redistribuir esas funciones de manera óptima en las nuevas tarjetas diseñadas por la empresa. Este proyecto se centra en el estudio y desarrollo de una de estas nuevas tarjetas, específicamente la que ha sido denominada NMB (New Main Board). Este nombre, que significa "Nueva Tarjeta Principal," se eligió debido a que esta tarjeta será la unidad central en la mayoría de los equipos que la empresa actualmente ofrece a sus clientes. La NMB está destinada a ser la pieza clave que integrará y coordinará las funciones esenciales de los sistemas, posicionándose como un componente crucial en la evolución tecnológica de los productos de la empresa.

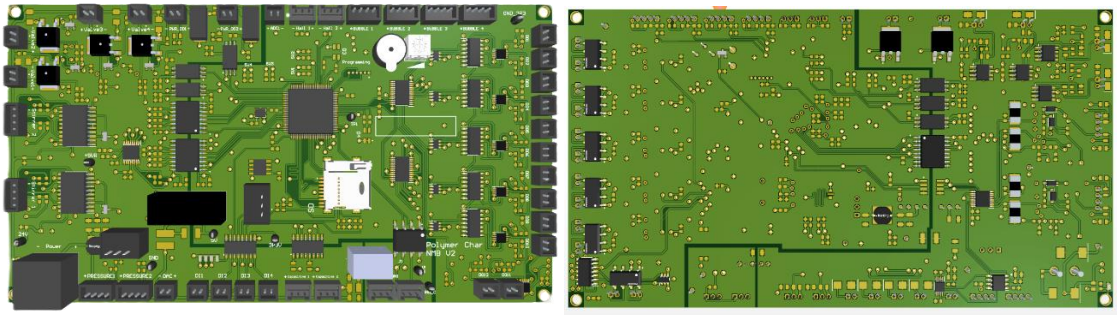
A partir del listado de funciones, esta electrónica se encargará de mantener sincronizados tanto los detectores del equipo como los subsistemas de terceros añadidos a este. Esta función es crucial, ya que garantiza una mayor exactitud en los resultados obtenidos. Además, la electrónica se encargará de recopilar y almacenar toda la información proveniente de los sensores conectados que puedan influir en los resultados, actuando de manera proactiva cuando sea necesario para mantener el equipo en condiciones óptimas de funcionamiento.

Por lo tanto, se procederá a describir el hardware incorporado en esta electrónica, así como los distintos dispositivos y sensores que pueden conectarse a ella. A continuación, se detallará el tipo de firmware desarrollado, proporcionando una explicación exhaustiva tanto del propio firmware como de la BIOS[8] diseñada para esta tarjeta. Finalmente, se explicará la integración de esta electrónica en la DLL, destacando las funciones creadas para acceder a la información proporcionada por el sistema. Además, se presentará el funcionamiento de la interfaz de usuario, diseñada específicamente para este proyecto.

4.1 Hardware

Este proyecto no incluye el diseño del hardware, ya que este fue desarrollado por un compañero de la empresa. Sin embargo, es fundamental comprender con precisión su funcionamiento, dado que la verificación del hardware sí formó parte del trabajo. Además, para desarrollar un firmware preciso y funcional, es esencial tener un conocimiento detallado del hardware al que se destina la programación.

Por ello en la siguiente imagen podemos apreciar el diseño tanto de la capa TOP (Vista Superior), como de la capa BOTTOM (Vista Inferior).



3. Vista Superior e inferior NMB

A nivel de hardware, la tarjeta está diseñada para manejar los siguientes componentes:

- 4 válvulas solenoides.
- 2 agitadores.
- 2 salidas de potencia.
- 1 ADC (Convertidor Analógico a Digital) externo.
- 2 sensores Hall.
- 4 sensores de burbujas.
- 1 sensor de temperatura y humedad
- 12 salidas digitales.
- Comunicación CAN.
- 2 sensores capacitivos.
- 4 entradas digitales.
- 1 DAC (Convertidor Digital a Analógico).
- 2 sensores de presión.
- 1 buzzer.
- 1 tarjeta MicroSD.
- 1 microcontrolador STM32F407

Esta diversidad de recursos destaca la capacidad y versatilidad de la electrónica desarrollada. A continuación, se ofrece una breve explicación de cada uno de estos componentes.

4.1.1 Válvula Solenoide

Una válvula solenoide, también conocida como válvula de funcionamiento eléctrico, funciona mediante la fuerza electromagnética. Cuando se aplica una corriente eléctrica a la bobina del solenoide, se genera un campo magnético que desplaza una varilla de metal ferroso, lo que permite el accionamiento de la válvula.

Estas válvulas requieren una alimentación inicial para activarse, la cual generalmente está especificada por el fabricante y suele estar acompañada de un tiempo determinado, em caso de exceder los tiempos marcados puedes llegar a quemarlas. En el caso de la empresa, se utilizan válvulas que se alimentan inicialmente a 24 V. Además, las válvulas tienen una alimentación de trabajo, también conocida como "fallback". Por otro lado, a nivel de conexión con el microcontrolador, se conectan a un recurso PWM del cual en el apartado del firmware se hará más hincapié, lo que permite un control más eficiente de las tensiones de trabajo.

4. Válvulas Bürkert y Parker

Estas válvulas desempeñan un papel fundamental en los equipos, y prácticamente todos los sistemas cuentan con alguna de ellas. Su función principal es permitir el paso de los disolventes, con los polímeros disueltos en ellos, de una estancia a otra a través de tubos específicos.

4.1.2 Agitadores y Sensores HALL

Los agitadores también conocidos como "Stirrer" en ingles son muy populares en el sector químico ya que estos ayudan a mantener las disoluciones siempre en el punto más optimo y ayudan a acelerarlas lo cual hacen que los procesos sean más rápidos.

En este caso utilizamos un sistema de agitación basado en un agitador de inducción típico, se coloca un recipiente de material no magnético sobre una base que contiene una bobina electromagnética. Al pasar corriente eléctrica a través de la bobina, se genera un campo magnético alterno. Este campo magnético induce corrientes de Foucault en un objeto metálico, como una barra magnética, que se encuentra dentro del recipiente como se puede apreciar en la ilustración 5.

La barra magnética, debido a las corrientes inducidas, empieza a girar, generando el movimiento necesario para agitar el líquido en el recipiente. El giro de la barra magnética se puede controlar ajustando la frecuencia y la intensidad del campo magnético, lo que permite variar la velocidad de agitación según las necesidades del proceso, esto último será controlado por el firmware mediante una máquina de estados.

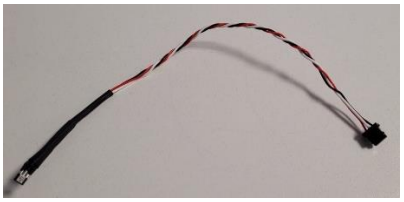
5. Agitador con recipiente e imán

Por otro lado, se encuentran los sensores Hall[9], un sensor Hall se utiliza para medir las vueltas de un imán detectando los cambios en el campo magnético a medida que el imán gira. Cuando un imán está unido a un eje giratorio y pasa cerca del sensor Hall, el sensor detecta la presencia del campo magnético generado por el imán. Cada vez que el imán completa una vuelta, el sensor registra un cambio en el campo magnético.

Este cambio genera una señal de salida en el sensor Hall, que puede ser un pulso digital (en el caso de un sensor Hall digital) o una variación continua en la salida (en un sensor Hall analógico).

Al contar el número de pulsos o medir las variaciones en la señal, es posible determinar el número de vueltas que ha dado el imán.

En este caso se utiliza un sensor hall del tipo pulso digital.



6. Sensor Hall

4.1.3 Salidas de Potencia

Las salidas de potencia en esta tarjeta son dos, ambas con el mismo propósito: suministrar una alimentación de 24V a otro subsistema mediante la activación de un pin del microcontrolador. Esto permite controlar el encendido y apagado de componentes externos directamente desde esta electrónica. Además, para una mayor protección, la rama de 24V cuenta con un fusible que protege el resto del sistema electrónico en caso de cualquier problema.

4.1.4 ADC Externo

El ADC[10] externo es un recurso que permite la introducción de señales analógicas a través de un conector Hirose. Estas señales se dirigen a un pin específico del microcontrolador, que está dedicado a la conversión analógico-digital (ADC). A través de este proceso, la señal analógica se convierte en una señal digital, lo que permite su monitorización y procesamiento dentro del sistema.

4.1.5 Sensores de burbujas

Los sensores de burbujas son fundamentales en estos equipos, ya que permiten detectar la presencia de bolsas de aire en los tubos por donde circula el disolvente. La detección de estas burbujas es crucial, ya que, si llegan a partes delicadas del sistema, pueden causar daños significativos, comprometiendo la funcionalidad del equipo. Además, estas piezas suelen ser costosas, por lo que identificar y actuar ante la presencia de burbujas es de vital importancia para prevenir daños y garantizar el correcto funcionamiento del equipo.

Mas tarde en el capítulo del firmware se comentará como hacer que funcionen y que acciones se han de tomar en caso de detección.

4.1.6 Sensores Capacitivos y Entradas Digitales

En este proyecto, tanto los sensores capacitivos como las entradas analógicas a nivel de microcontrolador cumplen la misma función: monitorizar o registrar señales del tipo ON/OFF. Por lo tanto, el microcontrolador recibe únicamente señales de nivel alto o bajo. Para garantizar una mayor seguridad y un funcionamiento óptimo, todas estas señales están optoacopladas.

4.1.8 DAC

Un Convertidor Digital a Analógico (DAC) es un dispositivo que convierte señales digitales en señales analógicas. En este proyecto, el DAC se comunica a través de un bus I2C, mediante el cual el microcontrolador envía un valor digital que representa el nivel de tensión deseado para la salida. El DAC se encarga de convertir este valor digital en una señal de tensión analógica.

Históricamente, esta señal analógica se ha utilizado en la empresa para proporcionar la concentración de disoluciones, un parámetro crítico en sistemas como el viscosímetro. La viscosidad de una disolución está directamente relacionada con su concentración, por lo que conocer ambas variables es esencial para un análisis preciso.

4.1.9 Sensor de Presión

Para la medición de presión en el sistema, la señal analógica generada por el sensor de presión debe ser procesada antes de llegar al microcontrolador. Este procesamiento se realiza a través de un integrado que actúa como un Convertidor Analógico-Digital (ADC). El ADC convierte la señal analógica en un valor digital, que es posteriormente transmitido al microcontrolador a través de un bus I2C.

Una vez que el microcontrolador recibe la señal digital, se emplean algoritmos específicos para reconstruir el valor analógico de la señal original. Con este valor y aplicando un factor de conversión proporcionado por el fabricante del sensor, se puede calcular con precisión la presión en unidades de psi (libras por pulgada cuadrada).

En este proyecto, se ha seleccionado un sensor de presión fabricado por Gefran[11], conocido por su alta precisión y fiabilidad. Este sensor es utilizado para medir la presión en componentes críticos del sistema, como es el caso de una jeringa, garantizando un control exacto de la presión aplicada en las aplicaciones desarrolladas.

7. Sensor de Presión Gefran

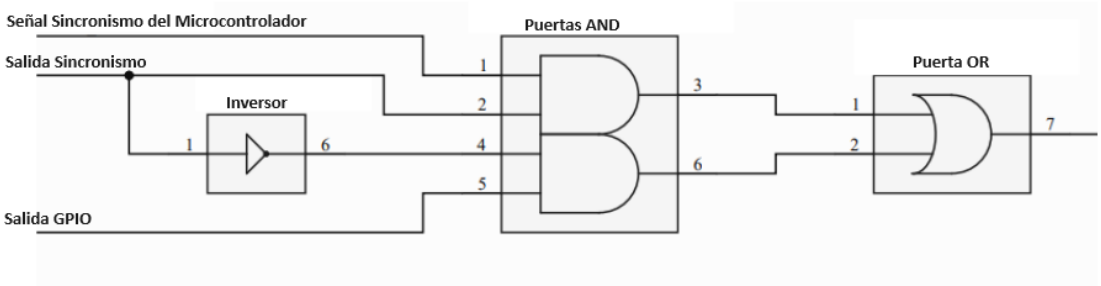
4.1.10 Salidas Digitales

Las salidas digitales en este proyecto tienen varias utilidades. Una es su uso como salidas de propósito general, donde el microcontrolador controla el encendido y apagado de dispositivos externos. Sin embargo, su objetivo principal es sincronizar todos los detectores o componentes que lo requieran.

Cabe señalar que estas salidas no pueden realizar múltiples funciones simultáneamente. Es decir, pueden configurarse como salidas de propósito general o de sincronización, pero no ambas a la vez. Esto se debe a que el hardware incluye puertas lógicas que previenen su configuración dual.

Además, para activar simultáneamente las salidas destinadas a la sincronización, se envía una señal desde el microcontrolador a todas las salidas digitales. Esta señal, tratada como un pulso, activa todas las salidas de sincronismo al mismo tiempo.

En la siguiente ilustración 8 se muestra el diagrama del hardware de las salidas digitales:



8. Diagrama de Salidas Digitales

A partir de este diagrama, se obtiene la siguiente tabla de verdad:

Señal Sync	Salida Sync	Salida GPIO	Resultado
0	0	0	Bajo
X	0	1	Alto
0	1	0	Bajo
0	1	1	Bajo
1	0	0	Bajo
1	1	0	Alto
1	1	1	Alto

3. Tabla de verdad Salidas Digitales

4.1.11 Sensor Temperatura y Humedad

El sensor de temperatura y humedad empleado en este proyecto es un dispositivo integrado proveniente de la compañía Honeywell[12]. Su función es medir la temperatura de la PCB y la humedad ambiental. Aunque su importancia no es crítica, contar con información sobre las condiciones del entorno siempre es útil para el funcionamiento general del sistema.

Este sensor se comunica a través del bus I2C, y mediante fórmulas específicas es posible convertir los valores proporcionados por el dispositivo en temperatura (grados Celsius) y en un valor porcentual de humedad relativa.

9. Sensor de Temperatura y Humedad

4.1.12 Zumbador

El zumbador es un componente sonoro utilizado en la electrónica para alertar al usuario o técnico sobre la aparición o existencia de un problema en el equipo.

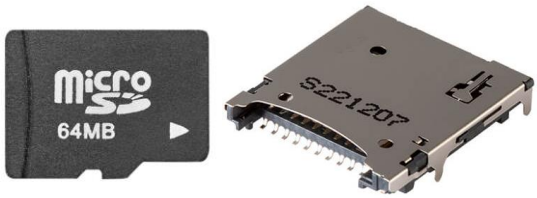
En este proyecto, se han desarrollado diferentes tipos de alarmas variando el ciclo de trabajo de una señal PWM, ya que el zumbador está conectado directamente a esta señal. Así, la velocidad del pitido permite identificar el tipo de alarma que se ha activado. Además, el zumbador cuenta con un potenciómetro manual que permite ajustar la intensidad del sonido del pitido. En futuras versiones del proyecto, este potenciómetro podría ser reemplazado por una versión digital que se comunique a través de un bus I2C, lo que permitiría la implementación de diferentes tonalidades en las alarmas.

10. Zumbador

4.1.13 Tarjeta MicroSD

La tarjeta microSD se implementará por primera vez en la electrónica de la empresa, y su principal funcionalidad será almacenar información sobre todas las alarmas que hayan surgido durante la vida útil del equipo. Esta capacidad de almacenamiento facilitará el mantenimiento de los equipos en los clientes, proporcionando una visión detallada del estado del dispositivo a lo largo del tiempo.

La comunicación con el microcontrolador se realizará a través de un bus SPI, que ofrece una alta velocidad de transferencia de datos. Además, la tarjeta microSD estará conectada a un pin del microcontrolador que indicará si la tarjeta está o no insertada en la ranura.



11. Tarjeta y Carcasa SD

4.1.14 Microcontrolador

El microcontrolador desempeña un papel fundamental como el componente central en la PCB, ya que controla, coordina y comunica todas las funciones del circuito, garantizando la ejecución precisa y eficiente de las tareas para las que se diseñó el sistema. En este proyecto, el microcontrolador es el eje principal, ya que aloja la información del firmware programado.

La selección de este componente es crítica debido a la amplia variedad disponible en el mercado, tanto entre diferentes fabricantes como dentro de una misma compañía, con múltiples modelos diseñados para adaptarse a diversas funciones específicas. En este caso, se eligió la familia STM32 debido a que los últimos firmwares desarrollados en la empresa se han realizado con componentes de esta marca. Además, STM32 ofrece un entorno de desarrollo intuitivo y eficiente para la programación y depuración del firmware.

El microcontrolador seleccionado para este proyecto es el modelo STM32F407VG[13], que es un dispositivo de 32 bits basado en la arquitectura ARM Cortex con una frecuencia de reloj de CPU de 168 MHz.

Caracteristica	Descripción
Nucleo	ARM Cortex-M4
Frecuencia Maxima de Funcionamiento	168 MHz con FPU
Rendimiento	210 DIMPS y 566 CoreMark
Serie	STM32F4 Series
Linea de Producto de Serie	STM32F407VG
Numero de Pines	100
Tamaño de la memoria Flash	1 Mbyte
Tamaño de la memoria RAM	192 Kbytes

4. Características STM32F407VG

Por último, se listarán todos los recursos de los que dispone este microcontrolador:

1. **2x USB OTG:** Uno de los puertos USB OTG (On-The-Go) con soporte para High-Speed (HS)
2. **Audio:** PLL dedicado para audio y 2 interfaces I²S de duplex completo.

3. **Hasta 15 Interfaces de comunicación:** Incluye 6 USARTs que operan hasta 11.25 Mbit/s, 3 SPI que operan hasta 45 Mbit/s, 3 I²C, 2 CAN y SDIO
4. **Analógico:** Dos convertidores digital-analógico (DAC) de 12 bits, tres convertidores analógico-digital (ADC) de 12 bits con una velocidad de hasta 2.4 MSPS o 7.2 MSPS en modo intercalado.
5. **Hasta 17 Timers:** Timers de 16 y 32 bits que operan hasta 168 MHz.
6. **Memoria Extensible:** Rango de memoria fácilmente ampliable utilizando el controlador de memoria estática flexible que soporta Compact Flash, SRAM, PSRAM, NOR y NAND.
7. **Generador de números aleatorios verdaderos:** Generador de números aleatorios analógicos
8. **Procesador Criptografico/Hasher:** Integra un procesador de cifrado/hash que proporciona aceleración por hardware para AES (128, 192, 256 bits), Triple DES y hash (MD5, SHA-1).

12. Microcontrolador STM32F407VG

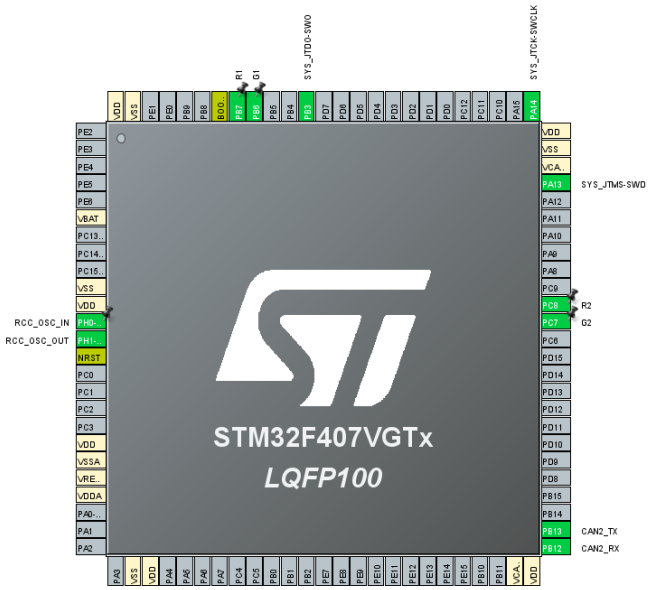
4.2 Configuración Microcontrolador

Para la configuración del microcontrolador empleado en este proyecto, se decidió utilizar el software proporcionado por la empresa fabricante de los microcontroladores, STMicroelectronics. Este software, conocido como STM32CubeMX, fue seleccionado por diversas razones que lo hacen especialmente adecuado para el desarrollo y configuración de sistemas embebidos basados en la familia de microcontroladores STM32.

Una de las principales razones para optar por STM32CubeMX es su carácter altamente intuitivo, que permite a los desarrolladores configurar de manera eficiente los distintos periféricos y características del microcontrolador sin necesidad de interactuar directamente con el código de bajo nivel. Este entorno gráfico facilita la selección y configuración de pines, la activación de periféricos como ADC, DAC, timers, y comunicaciones, así como la gestión del reloj del sistema. De esta forma, se minimizan los errores humanos durante la configuración y se asegura una mayor precisión en los ajustes iniciales del hardware.

Además, STM32CubeMX se integra perfectamente con otros entornos de desarrollo, como STM32CubeIDE, lo que permite una transición fluida desde la configuración del hardware hasta la programación del firmware. Esta integración contribuye a una mejora significativa en la eficiencia del flujo de trabajo, ya que reduce la necesidad de herramientas adicionales y simplifica el proceso de desarrollo.

Otra ventaja importante de STM32CubeMX es su capacidad para generar código automáticamente en base a las configuraciones realizadas. Esto no solo acelera el desarrollo del proyecto al reducir el tiempo necesario para escribir y depurar código manualmente, sino que también asegura que el código generado sea optimizado y esté alineado con las mejores prácticas recomendadas por STMicroelectronics. Este enfoque estandarizado facilita además el mantenimiento y la escalabilidad del proyecto, ya que permite realizar cambios en la configuración de manera rápida y sin afectar negativamente otras partes del sistema.



13. Configuración Microcontrolador BIOS

4.2.1 Configuración BIOS

La BIOS desarrollada para este proyecto se diseñó con una arquitectura basada en un hilo principal, que es responsable de ejecutar las tareas críticas necesarias para la inicialización y el correcto funcionamiento del bucle principal del sistema. Este hilo principal maneja las rutinas esenciales, como la inicialización de los periféricos, la verificación de los componentes clave a través de pruebas de diagnóstico, y la preparación del sistema para cargar el firmware o sistema operativo.

Además del hilo principal, se ha implementado un temporizador (timer) que opera en segundo plano y está configurado para generar interrupciones a intervalos regulares.

La combinación del hilo principal y la interrupción del timer permite que la BIOS gestione de manera eficiente tanto las tareas continuas como las tareas temporizadas, asegurando que el sistema mantenga un rendimiento estable y confiable desde el momento en que se enciende.

4.2.1.1 Configuración Timer

En la configuración del temporizador, se pueden modificar varios parámetros para lograr el objetivo de activar su interrupción cada 250 milisegundos. Para ello, se utiliza la siguiente fórmula:

$$Actualizacion\ Evento = \frac{TIM_{CLK}}{(PSC+1)*(ARR+1)*(RCR+1)} \qquad (1)$$

Donde:

- TIM_CLK es la frecuencia de reloj que afecta a los temporizadores y tiene un valor de 84 MHz.
- Los valores de PSC (prescaler), ARR (Counter Period) y RCR pueden variar, siempre que el resultado final sea el valor deseado.

Para optimizar el funcionamiento del microcontrolador el valor del Counter Period es mejor que sea mayor que el valor del prescaler, por ello se eligen los siguientes valores:

- PSC=999
- ARR=20999
- RCR=0

Con estos valores, se obtiene una actualización de evento deseada de 250 milisegundos.

4.2.1.2 Configuración CAN

El recurso CAN es necesario para poder realizar una comunicación efectiva con el PC, y así poder proporcionar la información que este requiera, además es necesaria para poder realizar una reprogramación remota, ya que, es en el estado de BIOS cuando se ha de realizar este proceso.

Por motivos de Hardware y rutado en la PCB los Pines designados como recepción de datos y transmisión son los siguientes:

- PB13: para la transmisión

- PB12: para la recepción

4.2.1.3 Configuración Salidas y Entradas Digitales

Para la implementación de la BIOS no ha sido necesario realizar muchas conexiones de propósito general, limitándose únicamente a la inclusión de los LED's presentes en la tarjeta. Al analizar el esquemático, se observa que la tarjeta cuenta con dos LED's configurables en color rojo o verde. Estos LED's están conectados de tal manera que se encienden cuando el microcontrolador (MCU) envía una señal de nivel bajo, y permanecen apagados cuando se envía una señal de nivel alto.

En la se muestra cómo se han configurado los pines del MCU asignados a las respectivas conexiones de los LED's. Como parte de la configuración inicial, los LED's se mantendrán apagados hasta que se active la interrupción que controla su funcionamiento. Esta configuración permite un control eficiente del estado de los LED's, asegurando que solo se activen en momentos específicos definidos por la lógica de la BIOS.

Pin Name	Signal on ...	GPIO outp...	GPIO mode	GPIO Pull-up/Pull-down	Maximum o...	User Label	Modified
PB6	n/a	High	Output Push Pull	No pull-up and no pull-down	Low	G1	☑
PB7	n/a	High	Output Push Pull	No pull-up and no pull-down	Low	R1	☑
PC7	n/a	High	Output Push Pull	No pull-up and no pull-down	Low	G2	☑
PC8	n/a	High	Output Push Pull	No pull-up and no pull-down	Low	R2	☑

14. Configuración GPIO BIOS

4.2.2 Configuración Firmware

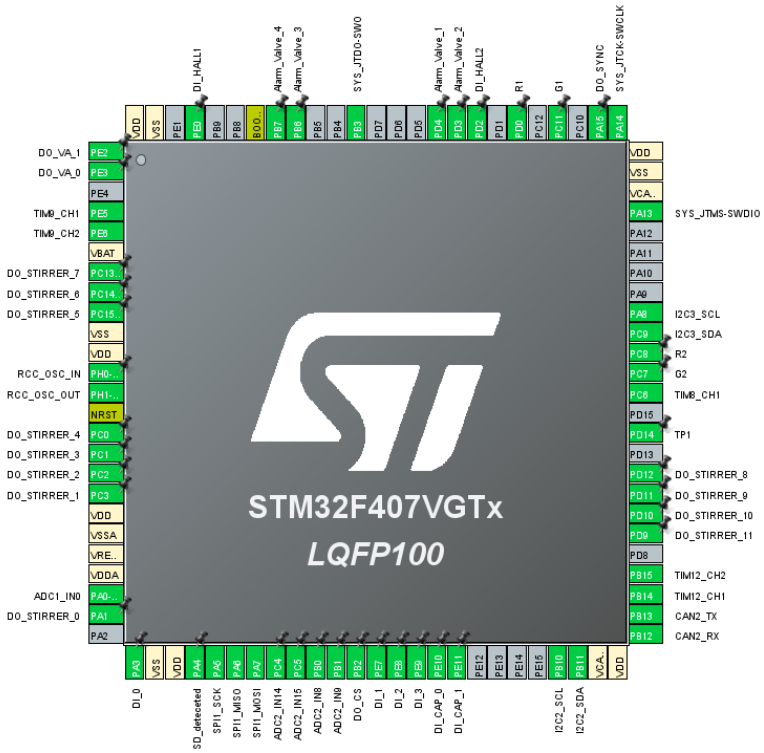
Un firmware es un tipo de software especializado que se encarga de controlar y gestionar el hardware de un dispositivo electrónico. A diferencia del software de aplicación, que es fácilmente modificable por el usuario, el firmware se almacena en una memoria de solo lectura o en una memoria flash dentro del dispositivo, proporcionando las instrucciones fundamentales que permiten que el hardware funcione de manera correcta y eficiente. Este firmware actúa como una capa intermedia entre el hardware y el software de alto nivel, interpretando las órdenes de este último y traduciendo esas instrucciones en acciones concretas que el hardware puede ejecutar.

En este proyecto, se ha aprovechado extensamente una variedad de recursos del microcontrolador. A continuación, se describen con mayor detalle los elementos utilizados.

Para la comunicación, se ha implementado el protocolo CAN para la interacción con el PC, mientras que el bus I2C se ha empleado para la conexión con los distintos integrados. Por otro lado, la comunicación con la tarjeta SD se ha gestionado a través del bus SPI.

Además de las interfaces de comunicación, se han utilizado diversos recursos internos del microcontrolador. Entre ellos, destacan dos ADCS, ocho temporizadores para la gestión de interrupciones, dos temporizadores con disparo externo y tres temporizadores adicionales que actúan como generadores de señales PWM. También se han configurado varios pines como salidas y entradas digitales para interactuar con otros componentes del sistema.

Todo esto se ha desarrollado desde la aplicación STM32CubeMX, al igual que se hizo con la BIOS, en la siguiente imagen se puede apreciar los pines del microcontrolador de los que se ha hecho uso.



15. Configuración Microcontrolador FW

4.2.2.1 Configuración Timers por Interrupción

Como se ha mencionado anteriormente, el código se basa en una estructura con un bucle principal en formato secuencial. Sin embargo, también incluye una serie de interrupciones que son controladas a intervalos regulares gracias a los temporizadores (Timers). Además, el microcontrolador STM32 cuenta con un sistema denominado *Nested Vectored Interrupt Controller* (NVIC), que permite asignar prioridades y subprioridades a las interrupciones, otorgando la capacidad de gestionarlas de manera jerárquica y excluyente cuando sea necesario.

Para calcular los tiempos de actualización de eventos, se utiliza la misma fórmula que se empleó previamente para el cálculo de los temporizadores en la BIOS:

$$Actualizacion\ Evento = \frac{TIM_{CLK}}{(PSC + 1) * (ARR + 1) * (RCR + 1)}$$

Los Timers que se emplean para dar uso de interrupciones se muestran en la siguiente imagen junto a su configuración con los valores empleados:

TIMER	6	14	10	11	5	13	2	7
PSC	1	399	999	999	199	39	3999	39
ARR	4199	20999	1049	1049	20999	20999	20999	2099
RCR	0	0	0	0	0	0	0	0
Tiempo Actualización (ms)	0,1	100	12,5	12,5	50	10	1000	1

5. Configuración Timers por interrupción

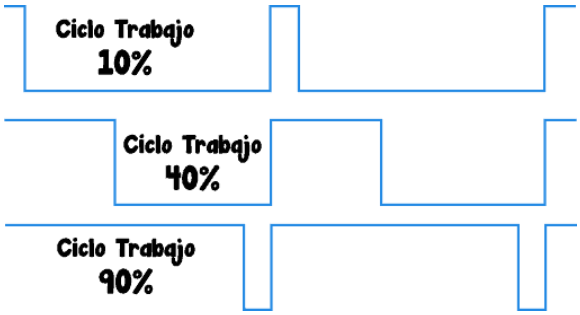
Además, a cada uno de estos Timers en función de la importancia de que poseen se les ha asignado los niveles de prioridad para poder asignar las capacidades de exclusión e importancia. En la siguiente tabla se puede apreciar los niveles de estos:

TIMER	Preemption	Subpriority
6	0	0
14	1	0
10	2	0
11	2	1
5	3	0
13	4	0
2	5	0
7	6	0

6. Prioridades Interrupciones Firmware

4.2.2.2 Configuración Timer por PWM

Un timer PWM[14] (Pulse Width Modulation) es un recurso clave en microcontroladores y sistemas embebidos que permite generar señales digitales con una frecuencia y un ciclo de trabajo específicos. El ciclo de trabajo, expresado en porcentaje, indica la proporción del tiempo que la señal permanece en estado alto (encendida) durante un ciclo completo de la señal.



16. PWM

Los Timers que aprovechan esta característica son los Timers 9, 12 y 8, los cuales, aunque comparten propiedades similares, tienen diferentes aplicaciones. Los Timers 9 y 12 se utilizan para controlar las válvulas solenoide, mientras que el Timer 8 se emplea para gestionar el zumbador.

En cuanto a sus configuraciones comunes, todos operan en modo de contador ascendente y utilizan el reloj sin división. Para optimizar su rendimiento, los parámetros ajustables en estos Timers son el Prescaler y el período, lo que permite adaptar su funcionamiento a las necesidades específicas del proyecto.

4.2.2.3 Configuración Timer por External Trigger

Un Timer por Trigger es un temporizador que se activa o sincroniza mediante una señal externa conocida como "trigger" o disparo.

Este tipo de temporizador se emplea en el proyecto para capturar la señal generada por el sensor Hall, lo que permite contabilizar el número de pasos o transiciones del polo sur del imán frente al sensor. Los Timers que aprovechan esta funcionalidad son el Timer 3 y el Timer 4. Su configuración específica es la siguiente:

TIMER	3	4
Clock Source	Ext. Trigger	Ext. Trigger
Polarity	Non Inv.	Non Inv.
Prescaler	Div 1	Div 1
Filter	0	0

7. Configuración Timer 3 y 4

4.2.2.4 Configuración ADCs

La configuración de los ADCs en este proyecto involucra el uso de dos convertidores analógico-digitales (ADCs), cada uno con un propósito y configuración distintos.

El ADC1 está destinado a convertir la señal proveniente del ADC externo, y en este caso, solo se empleará un canal para dicha conversión.

En cambio, el ADC2 se asigna para obtener los valores digitales de los sensores de burbujas. Dado que se utilizan cuatro sensores, es necesario emplear cuatro canales de este ADC.

ADC	1	2
Prescaler	Div4	Div4
Resolution	12 bits	12 bits
Scan Conv. Mode	Disable	Enable
Cont. Conv. Mode	Disable	Enable

8. Configuración ADCs

El parámetro de configuración "Scan Conversion Mode" debe activarse cuando se necesiten realizar múltiples conversiones en un solo ADC, es decir, cuando se utilicen varios canales dentro del mismo convertidor.

4.2.2.5 Configuración GPIO

En esta sección se detallará la configuración de los distintos grupos de periféricos. Primero, se presenta la asignación de pines correspondientes a las señales digitales de entrada y salida. Adicionalmente, para las salidas digitales, también se muestra su valor inicial, es decir, si el microcontrolador establece un nivel alto o bajo para estas señales desde el inicio.

Pin Name	GPIO Pull-up/Pull-down	User Label
PA3	No pull-up and no pull-down	DI 0
PA4	No pull-up and no pull-down	SD Detected
PD2	No pull-up and no pull-down	DI HALL 2
PE0	No pull-up and no pull-down	DI HALL 1
PE7	No pull-up and no pull-down	DI 1
PE8	No pull-up and no pull-down	DI 2
PE9	No pull-up and no pull-down	DI 3
PE10	No pull-up and no pull-down	DI CAP 0
PE11	No pull-up and no pull-down	DI CAP 1

9. Digital Inputs

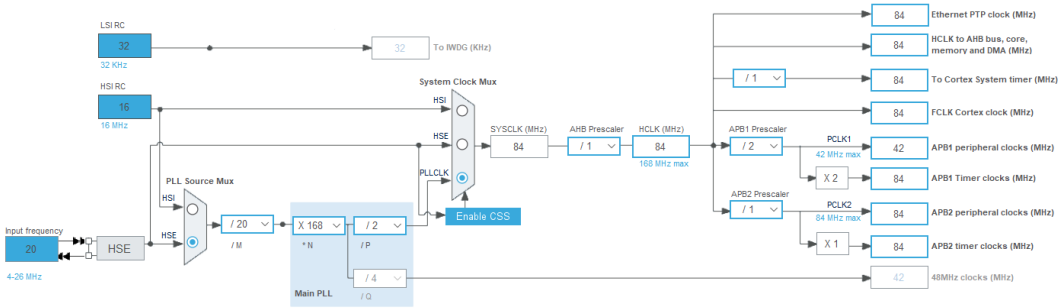
Pin Name	GPIO output	GPIO Mode	GPIO Pull-up/Pull-down	User Label
PB2	High	Output Open Drain	No pull-up and no pull-down	DO_CS
PA1	Low	Output Push Pull	No pull-up and no pull-down	DO_STIRRER_0
PA15	Low	Output Push Pull	No pull-up and no pull-down	DO_SYNC
PB6	Low	Output Push Pull	No pull-up and no pull-down	Alarm_Valve_3
PB7	High	Output Push Pull	No pull-up and no pull-down	Alarm_Valve_4
PC0	Low	Output Push Pull	No pull-up and no pull-down	DO_STIRRER_4
PC1	Low	Output Push Pull	No pull-up and no pull-down	DO_STIRRER_3
PC2	Low	Output Push Pull	No pull-up and no pull-down	DO_STIRRER_2
PC3	Low	Output Push Pull	No pull-up and no pull-down	DO_STIRRER_1
PC7	High	Output Push Pull	No pull-up and no pull-down	G2
PC8	High	Output Push Pull	No pull-up and no pull-down	R2
PC11	High	Output Push Pull	No pull-up and no pull-down	G1
PC13	Low	Output Push Pull	No pull-up and no pull-down	DO_STIRRER_7
PC14	Low	Output Push Pull	No pull-up and no pull-down	DO_STIRRER_6
PC15	Low	Output Push Pull	No pull-up and no pull-down	DO_STIRRER_5
PD0	High	Output Push Pull	No pull-up and no pull-down	R1
PD3	Low	Output Push Pull	No pull-up and no pull-down	Alarm_Valve_2
PD4	Low	Output Push Pull	No pull-up and no pull-down	Alarm_Valve_1
PD9	Low	Output Push Pull	No pull-up and no pull-down	DO_STIRRER_11
PD10	Low	Output Push Pull	No pull-up and no pull-down	DO_STIRRER_10
PD11	Low	Output Push Pull	No pull-up and no pull-down	DO_STIRRER_9
PD12	Low	Output Push Pull	No pull-up and no pull-down	DO_STIRRER_8
PD14	Low	Output Push Pull	No pull-up and no pull-down	TP1
PE2	Low	Output Push Pull	No pull-up and no pull-down	DO_VA_1
PE3	Low	Output Push Pull	No pull-up and no pull-down	DO_VA_0

10. Digital Outputs

4.2.2.6 Configuración del reloj

Las configuraciones del reloj que se han empleado han sido útiles tanto para la BIOS como para el Firmware. Estas características se pueden observar en la Figura 15, donde se muestra que la frecuencia de operación del microcontrolador se ha establecido en 84 MHz. Los pines configurados para el RCC (Reset and Clock Controller) y el SYS (System) son los siguientes:

- RCC → Pin PH0: *RCC-OSC-IN*, Pin PH1: *RCC-OSC-OUT*
- SYS → Pin PB3: *SYS_JTDO-SWO*, Pin PA14: *SYS_JTCK-SWCLK*, Pin PA13: *SYS_JTMS-SWDIO*



17. Configuración del Reloj

4.3 BIOS

La BIOS (Sistema Básico de Entrada/Salida) es un firmware esencial presente en computadoras y dispositivos electrónicos. Su principal objetivo es iniciar y verificar los componentes fundamentales del hardware, como la CPU, la memoria RAM, el disco duro y otros periféricos, cuando se enciende el dispositivo. La BIOS se ejecuta desde una memoria ROM o flash y su primera tarea es realizar el POST (Power-On Self Test), que asegura el correcto funcionamiento de los componentes esenciales.

Una vez que la BIOS confirma el estado adecuado del hardware, procede a cargar el sistema operativo desde el dispositivo de almacenamiento, cediéndole el control del sistema. Además de estas funciones de arranque, la BIOS actúa como una interfaz de bajo nivel entre el hardware y el software, permitiendo que el sistema operativo y otros programas se comuniquen con los componentes de hardware. En resumen, la BIOS es un software crítico que permite que un dispositivo arranque y funcione correctamente desde el encendido.

En cuanto a su ubicación en la memoria, la BIOS se distingue del firmware, siendo ambas almacenadas en posiciones diferentes dentro de la memoria del microcontrolador. En este proyecto, la BIOS se coloca en la dirección 0x80000000, mientras que el firmware, tras calcular el espacio ocupado por la BIOS y los parámetros a escribir en la memoria flash, se ubica en la dirección 0x804000000. Así, la estructura de direccionamiento del microcontrolador se organiza de la siguiente manera:

	Address	Size
BIOS	0x80000000	128Kbytes
Parametros	0x80200000	128Kbytes
Firmware	0x80400000	128Kbytes

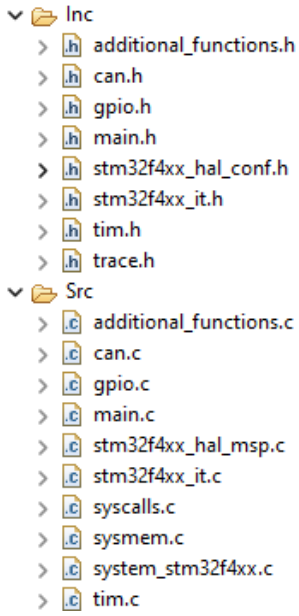
11. Direcciones Microcontrolador

Los parámetros son la información que se ha decidido almacenar en la memoria flash debido a su importancia para el funcionamiento del dispositivo. Este tipo de datos, que se recuperan durante el arranque, incluye parámetros de configuración de hardware, ajustes de calibración, estados de operación, entre otros. Al mantener estos parámetros en la memoria flash, se asegura que el sistema pueda iniciar con las configuraciones y estados necesarios para operar correctamente, incluso después de un ciclo de apagado y encendido.

Para iniciar la programación del código, se utilizó el software STM32CubeMX, el cual facilita la configuración de los recursos del microcontrolador y genera automáticamente el código fuente, organizándolo en una serie de directorios asociados al proyecto.

La estructura de programación adoptada a lo largo del desarrollo del proyecto se ha basado en la incorporación de los #DEFINE, #INCLUDE, funciones y estructuras en todos los archivos .h ubicados en la carpeta "Inc". De esta manera, en los archivos .c se limitó a la creación de variables globales y/o estáticas, así como a la implementación de las funciones lógicas que el programa debe ejecutar.

En la Figura 16 se muestra la organización de los ficheros generados por la aplicación CubeMX dentro del directorio CORE. Adicionalmente, se incluyeron archivos como trace.h, destinados a la depuración, lo que permitió visualizar mensajes y facilitar el proceso de desarrollo y prueba del software.



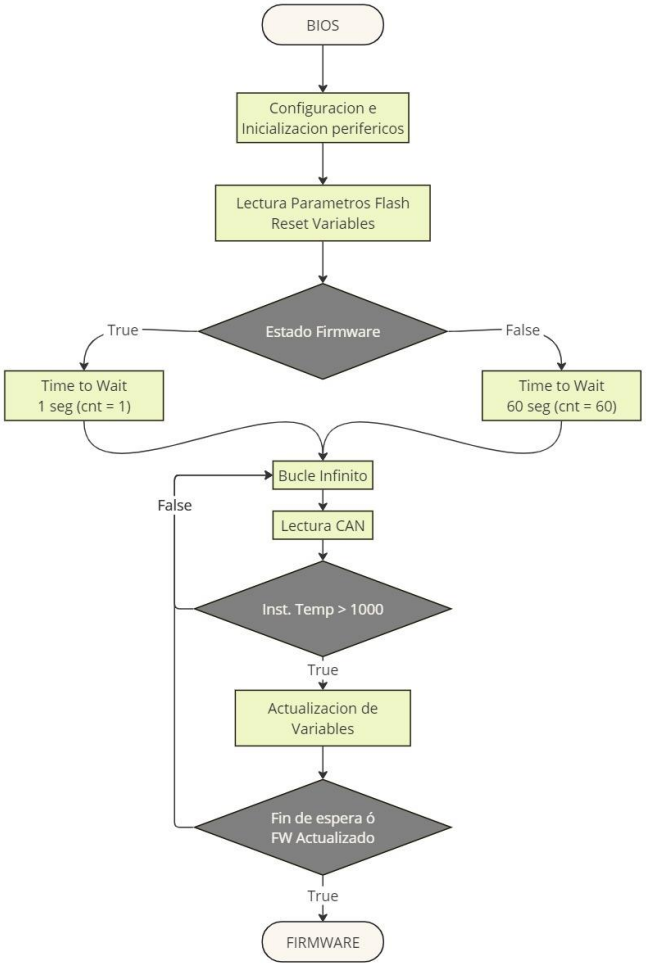
18. Arquitectura de Archivos BIOS

4.3.1 Bucle Principal (main.c)

En el bucle principal del programa se llevan a cabo todas las configuraciones e inicializaciones necesarias de los recursos requeridos para el correcto funcionamiento de la BIOS. Además, se procede a inicializar y resetear los valores de las variables globales. Una vez completadas estas tareas, se realiza una lectura de la memoria flash para obtener la información esencial, como la verificación de si ya existe un firmware cargado o si se debe esperar la posibilidad de cargar uno.

En caso de que se detecte la presencia de un firmware, se ajusta el tiempo de espera a 1 segundo. Si, por el contrario, no se encuentra ningún firmware cargado, se establece un tiempo de espera de 60 segundos. Si al final de este periodo no se ha cargado ningún firmware, la tarjeta se reiniciará automáticamente.

En el siguiente diagrama se ilustra de manera más clara el proceso descrito:



19. Diagrama main BIOS

4.3.2 Interrupción Timer (stm32f4xx_it.c)

La finalidad de esta interrupción es controlar el tipo de parpadeo de los LEDs principales que se encuentran en la tarjeta. El objetivo de este parpadeo es proporcionar información visual al usuario. Los diferentes tipos de parpadeo se pueden apreciar en las siguientes: tablas:

BIOS Standby		
Inst. Temporal	LED 1	LED 2
1		
2		
3		
4		

BIOS FW Update		
Inst. Temporal	LED 1	LED 2
1		
2		
3		
4		

12. Juego de Luces

En las tablas anteriores, se muestra la correlación entre el color de los LEDs, el instante temporal y el modo en el que se encuentra la BIOS.

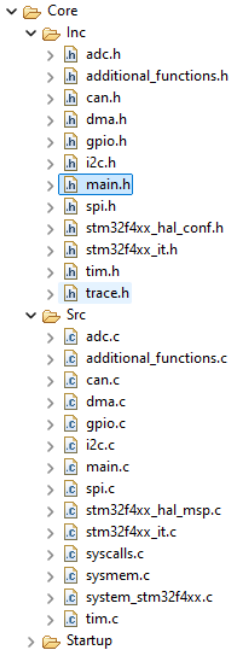
En el modo *Standby*, la BIOS espera un tiempo determinado antes de saltar al firmware.

Por otro lado, el acceso al modo de *Firmware Update* se realiza únicamente si el usuario envía el comando correspondiente desde un PC. En este caso, la BIOS inicia la rutina de actualización remota del firmware, un proceso que permite la descarga e instalación de nuevas versiones del software sin necesidad de intervención física directa en el equipo. Este enfoque no solo facilita la implementación de mejoras y correcciones de errores en el sistema, sino que también resulta extremadamente eficiente desde el punto de vista operativo y logístico. Al poder actualizar el firmware de forma remota, se eliminan o reducen significativamente los costos y el tiempo asociados con el envío de personal técnico a las instalaciones del cliente para realizar actualizaciones, optimizando así el mantenimiento y prolongando la vida útil de los equipos. Esta capacidad de actualización remota es particularmente valiosa en contextos donde los equipos están desplegados en ubicaciones remotas o de difícil acceso, proporcionando una solución práctica y efectiva para mantener los sistemas al día con las últimas mejoras tecnológicas.

4.4 Firmware

Para iniciar el desarrollo del Firmware, se utiliza la aplicación CubeMX, la cual genera el código base en el que se integrará toda la programación necesaria para el proyecto. Este proceso crea los archivos que se muestran en la imagen 20, ubicados en el directorio Core. Sin embargo, en la ilustración 20 no solo se observan los archivos generados por CubeMX; también se añadió el archivo Trace.h, al igual que en la BIOS, para facilitar la depuración del programa y mostrar mensajes durante estos procesos.

Además del archivo Trace.h, se incorporaron documentos adicionales para ayudar a la organización del proyecto con el fichero de aditional_funcitons en el cual se encuentran funciones de conversión de datos fundamentalmente. Al igual que en la BIOS, se emplearán diagramas ASM para facilitar la explicación del proceso.



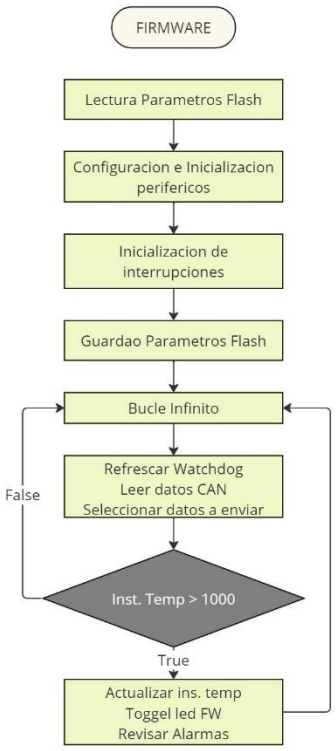
20. Estructura de archivos del firmware

4.4.1 Bucle principal (main.c)

Antes de ingresar al bucle principal, de manera secuencial, se realiza una lectura de la memoria flash para determinar el estado de configuración y las condiciones operativas en las que el equipo se encontraba antes de ser apagado. A continuación, se llevan a cabo todas las configuraciones e inicializaciones de los recursos utilizados en el proyecto. Seguidamente, se activan las interrupciones necesarias para el funcionamiento del sistema y, finalmente, antes de entrar en el bucle principal, se guarda el estado en la memoria flash.

Una vez completados todos estos ajustes, comienza el bucle principal, del cual el sistema no saldrá hasta que se apague nuevamente. Durante este bucle, se monitorea constantemente si se ha recibido un nuevo mensaje a través del protocolo de comunicación CAN. En caso afirmativo, se ejecuta la acción correspondiente a dicho mensaje y, si el usuario lo solicita, se envían los datos registrados.

Además, dentro del bucle, cada segundo se hace parpadear un LED para indicar que la tarjeta está operando en el firmware, informando así al usuario sobre el estado del dispositivo. Finalmente, se revisa si se ha detectado alguna alarma; en caso de que así sea, se enciende un LED rojo que indica la presencia de una alarma en la electrónica.



21. Diagrama ASM main

4.4.2 Interrupciones (stm32f4xx_it.c)

Las interrupciones son un componente crucial en el desarrollo de firmware, ya que permiten que el microcontrolador responda de manera inmediata y eficiente a eventos externos o internos, sin la necesidad de mantener un monitoreo constante mediante el bucle principal. Este mecanismo

asegura que el sistema pueda gestionar múltiples tareas en paralelo, optimizando así el rendimiento y garantizando una respuesta rápida a situaciones críticas, como cambios en señales de entrada, la recepción de datos, o la gestión de temporizadores.

4.4.2.1 Control de Agitadores (Timer 10 & 11)

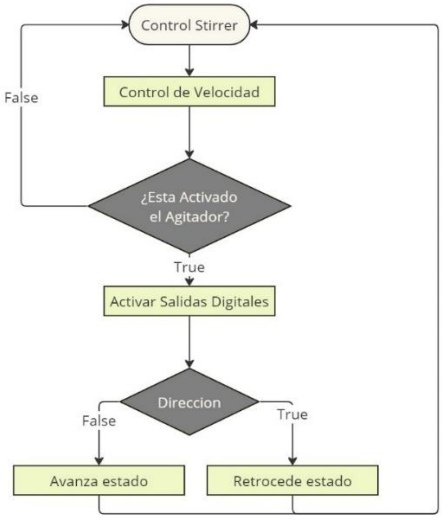
En esta interrupción se gestiona el control de los agitadores, los cuales pueden ser conectados a la tarjeta electrónica a través de dos conectores, cada uno manejado por una interrupción distinta. Sin embargo, a nivel de código, ambas interrupciones son prácticamente idénticas, ya que realizan las mismas funciones para cada agitador.

Al inicio de la interrupción, se lleva a cabo un control de la velocidad de los agitadores. Este ajuste es crucial para evitar cambios bruscos de velocidad que podrían provocar que el imán del agitador pierda sincronización y deje de funcionar. Para prevenir esto, la velocidad se incrementa de manera gradual, en pasos de 10 RPM, hasta alcanzar el valor deseado por el usuario.

Una vez completado el control de velocidad, se procede con la secuencia de funcionamiento del agitador, la cual está implementada mediante una máquina de estados. Dependiendo del estado en el que se encuentre, se activan las salidas digitales necesarias para alimentar la bobina correspondiente. La bobina alimentada determinará la dirección de giro del imán, permitiendo que el agitador gire en una dirección u otra.

Para cambiar la dirección de giro del agitador, basta con incrementar o decrementar el estado de la máquina de estados. Asimismo, el cambio de velocidad se logra ajustando el periodo de la interrupción, de modo que se ejecute con una frecuencia mayor o menor, en función del nuevo periodo establecido.

Finalmente, para asegurar que no haya problemas con el agitador, en dos de los estados de la máquina se activa un flag que permite medir la corriente a través de otra interrupción. Esto es fundamental para detectar cualquier falla, como cuando se intenta activar el agitador y no se registra corriente, lo que indicaría un problema. La medición de corriente se realiza en una interrupción separada debido a que el sensor utilizado opera mediante comunicación I2C, y para mantener un control eficiente de este bus, todas las operaciones relacionadas se manejan de manera secuencial dentro de la misma interrupción.



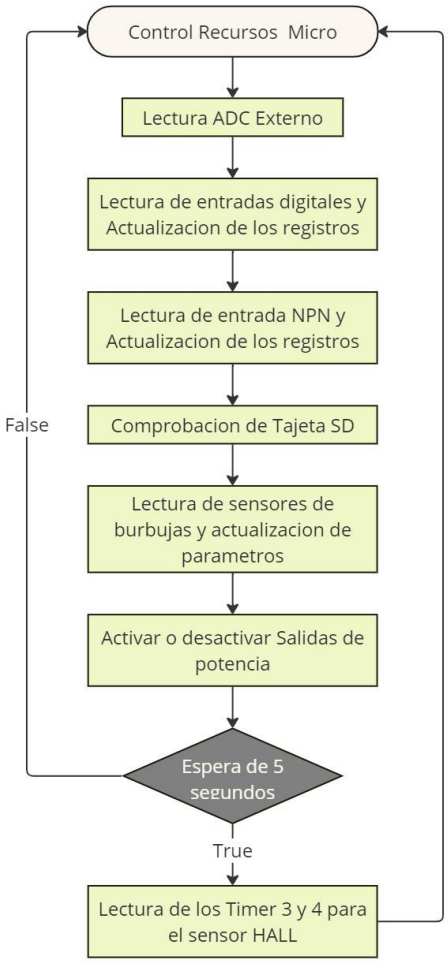
22Diagrama ASM Control de Stirrer

4.4.2.2 Control de Recursos del Microcontrolador (Timer 2)

Para gestionar de manera eficiente los recursos asociados a los sensores conectados, ya sea a través de ADCs o mediante entradas y salidas digitales, se ha decidido agrupar todos estos recursos en una única interrupción. Esta interrupción se ejecuta cada segundo, dado que no es necesario actualizar la información con mayor frecuencia, ya que los datos proporcionados por estos sensores no requieren una actualización más rápida.

Dentro de esta interrupción, se procesa la información recibida del ADC externo, actualizando su valor en el registro correspondiente. Además, se gestionan los ADCs encargados de medir los sensores de burbujas. Si alguno de estos sensores supera los umbrales establecidos por el usuario, se genera una alarma. Adicionalmente, esta alarma tiene la capacidad de activar una salida digital configurada para notificar a sistemas externos sobre el problema detectado, permitiendo detener el proceso en curso si es necesario.

En esta misma interrupción, se llevan a cabo todas las acciones relacionadas con las entradas y salidas digitales. Asimismo, se verifica la presencia de la tarjeta SD y se almacena toda la información recopilada en la memoria, asegurando un control adecuado de los recursos del sistema.



23. Diagrama ASM Recursos de Microcontrolador

4.4.2.3 Control de Sensores comunicados por I2C (Timer 13)

Esta interrupción está dedicada exclusivamente a los dispositivos que se comunican a través del bus I2C. La decisión de agrupar estas comunicaciones en una sola interrupción se tomó debido a la criticidad del bus I2C, que es propenso a problemas si se intenta comunicar múltiples dispositivos simultáneamente. Para evitar que el bus entre en un estado no óptimo, todas las comunicaciones en este bus se realizan de manera secuencial.

La frecuencia de ejecución de esta interrupción es de 10 milisegundos, mientras que su tiempo de ejecución es de 2,4 milisegundos. Es crucial que el tiempo de ejecución nunca supere el intervalo de refresco de la interrupción, para asegurar un funcionamiento fluido del sistema.

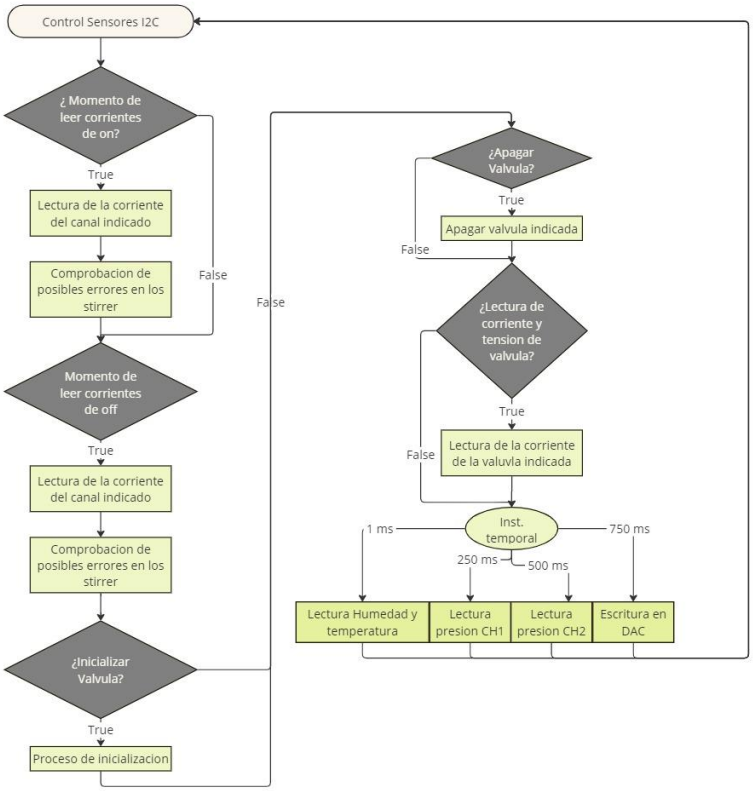
Dentro de esta interrupción, se distinguen dos tipos de comunicaciones: aquellas que se realizan bajo comando o mediante flags, y aquellas que se ejecutan de manera constante. Las lecturas bajo comando o por flag incluyen:

- Lectura de corrientes de las válvulas
- Lectura de tensión de las válvulas
- Lectura de corriente de los agitadores
- Inicialización de las válvulas
- Apagado de válvulas

Las comunicaciones que se ejecutan de manera continua están distribuidas en intervalos de 250 milisegundos para evitar la saturación del bus. Estas incluyen:

- Lectura de temperatura y humedad
- Lectura presión canal 1
- Lectura presión canal 2
- Ajustes del DAC

Esta estructuración permite un manejo eficiente del bus I2C, minimizando riesgos y asegurando la correcta operación de todos los dispositivos conectados.



24. Diagrama ASM sensores I2C

4.4.2.4 Gestión de flags para inyección y señal de sincronismo (Timer 14)

En este proyecto, se ha implementado un temporizador específico que genera una variable utilizada como controlador temporal para las inyecciones. Este mecanismo permite un control más preciso y detallado sobre el proceso de inyección.

```
typedef struct Injection{
    volatile uint8_t Status;
    volatile uint32_t Time[MAX_N_TIME];
    volatile uint8_t Mode[MAX_N_TIME];
    volatile uint8_t N_Time;
    volatile uint8_t Ptr_time;
}Injection;
```

25. Estructura Inyección

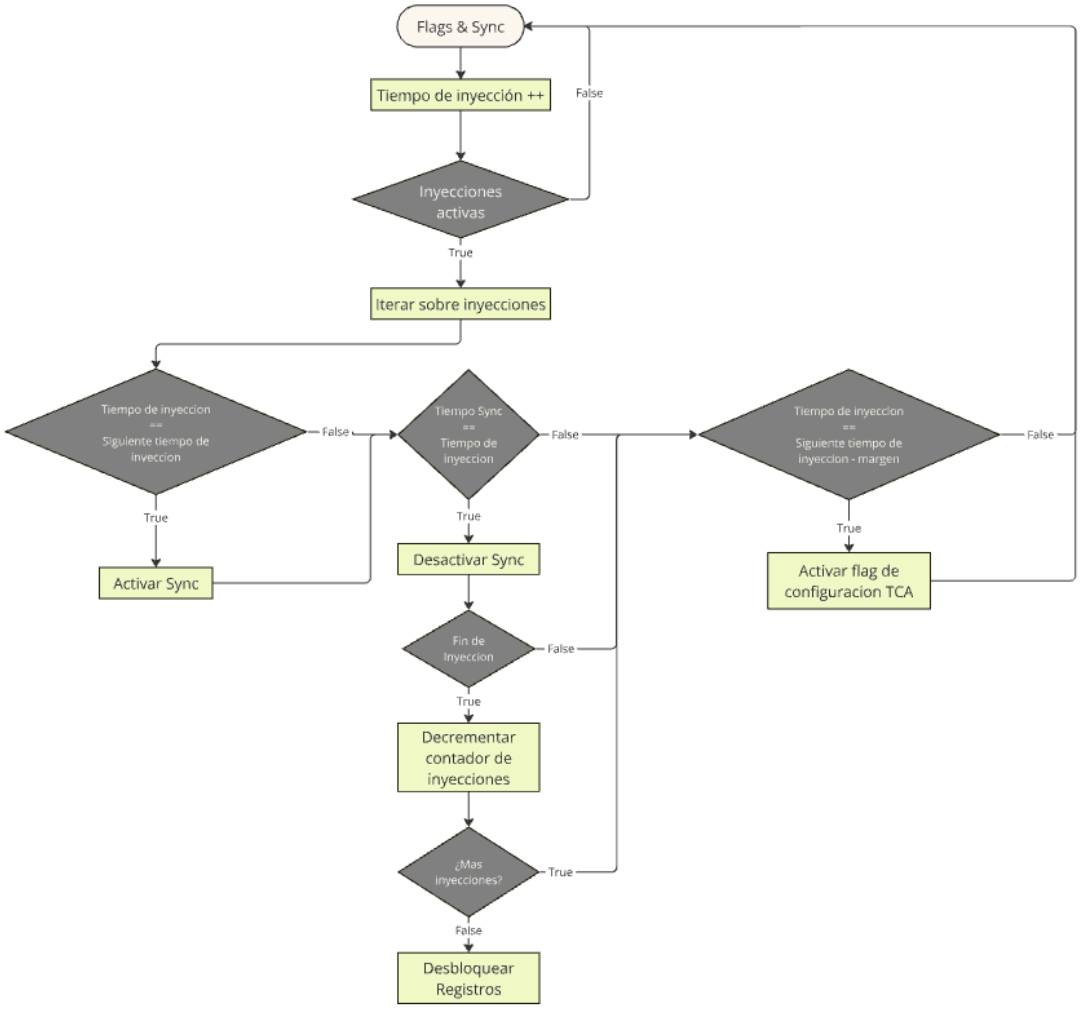
Para realizar un control exhaustivo de las inyecciones, se ha diseñado una variable del tipo estructura que encapsula toda la información relevante. Esta estructura contiene las siguientes variables:

- **Status**: Indica si la inyección ya se ha realizado o si está pendiente. Este campo es crucial para evitar repeticiones o saltos en el proceso de inyección.
- **Time**: Es un array que almacena los instantes de tiempo en los que deben ejecutarse las inyecciones. Este array permite programar múltiples inyecciones en diferentes momentos.
- **Mode**: Contiene la configuración necesaria para las salidas digitales durante la inyección. Esto incluye qué salidas deben activarse.

- **N_time**: Define el número de instantes de tiempo registrados en el array Time.
- **Ptr_time**: Es un puntero que indica la posición actual dentro del array Time y Mode. Este puntero es esencial para gestionar inyecciones que podrían solaparse, permitiendo al sistema saber en qué punto del ciclo se encuentra cada inyección.

El temporizador que controla esta estructura tiene un período de interrupción de cien milisegundos, y se le ha asignado la máxima prioridad en el sistema. Esto se debe a la importancia crítica de esta fase en el proyecto, ya que cualquier retraso o error en la ejecución de las inyecciones podría afectar significativamente en los resultados finales obtenidos.

La función principal de esta interrupción es gestionar las configuraciones de las salidas digitales y la activación de la señal de sincronismo en el momento adecuado. Este control preciso garantiza que las inyecciones se realicen de manera sincronizada y eficiente.



26. Diagrama ASM Flags & Synch

4.4.2.5 Configuración salidas digitales (Timer 5)

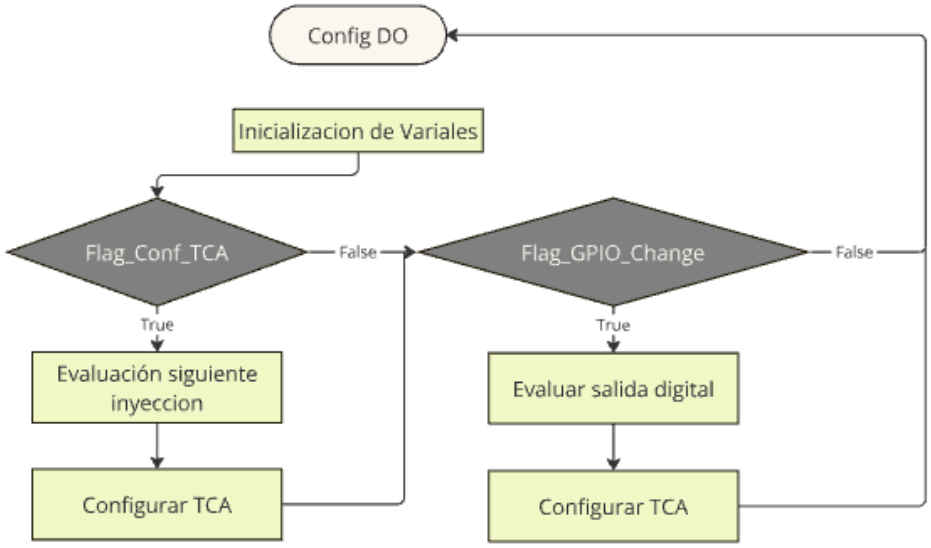
La función de esta interrupción es controlar la configuración de las salidas digitales (GPIO) para asegurarse de que están correctamente configuradas según los requisitos de cada inyección. La interrupción se ejecuta cada 50 milisegundos, un intervalo de tiempo lo suficientemente largo como para evitar solapamientos entre el tiempo de interrupción y su período. Este tiempo es crucial, ya que el controlador de las salidas digitales tarda aproximadamente 2,5 milisegundos en configurarse, garantizando que no se produzcan conflictos de tiempo.

El funcionamiento de esta interrupción se basa en flags (banderas) que se activan en otros momentos del código. Estas flags permiten que la interrupción se encargue tanto de las salidas digitales sincronizadas como de las salidas de propósito general.

A continuación, se describen las etapas clave del proceso que se ejecuta en esta interrupción:

1. **Inicialización de Variables**:
   - Se inicializan las variables StatusDO_SYNC, aux_StatusDO_SYNC y StatusDO_GPIO para manejar los estados de las salidas digitales.
   - Se define un array (DO_ARRAY) que contiene las configuraciones posibles de las 12 salidas digitales.
2. **Configuración de Salidas Sincronizadas (SYNC)**:
   - Si la flag Flag_Conf_TCA está activa (indicando que se debe reconfigurar el controlador TCA), se evalúa la configuración de la próxima inyección (Next_Inj_Config) para determinar qué salidas sincronizadas deben activarse.
   - Dependiendo del tipo de inyección (inicio, parada, modos A2D o B2D), se selecciona la configuración adecuada y se almacena en aux_StatusDO_SYNC.
   - A continuación, se actualiza StatusDO_SYNC para reflejar qué salidas digitales deben estar activas, y se registra esta información para ser enviada al controlador TCA.
   - Si ocurre algún error durante la configuración del TCA, se incrementa el flag de error flag_error_TCA.
3. **Configuración de Salidas de Propósito General (GPIO)**:
   - Si la flag flag_GPIO_change está activa, se reconfiguran las salidas de propósito general basadas en el estado actual de la tabla CAN_lookup_table_0x03.
   - Se actualiza StatusDO_GPIO con las salidas que deben activarse, y esta información se envía al controlador TCA. Si se detecta un error durante esta operación, también se incrementa el flag de error flag_error_TCA.

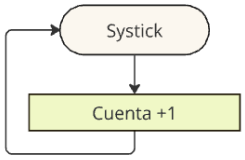
El siguiente diagrama ASM representa visualmente el proceso descrito en la interrupción:



27. Diagrama ASM Config DO

4.4.2.6 Generador de tiempos (Timer 6)

Este es uno de los temporizadores más sencillos ya que únicamente realiza una cuenta para así poder realizar un mejor control del tiempo y obtener la posibilidad de realizar delays con una resolución de cien microsegundos ya que el periodo del timer es de este tiempo.



28. Diagrama ASM Systick

4.4.2.7 Registrador (Timer 7)

La interrupción del registrador es diferente al resto ya que esta posee un periodo dinámico es decir puede cambiar en función de las necesidades del usuario, bastaría con cambiar el registro de SEND_TIME.

La principal función del registrador es almacenar la información de las señales que se consideran más críticas o las cuales son necesarias de monitorizar, estas son:

- Sequence Timer: El tiempo que lleva funcionamiento el firmware.
- Injection Count: El número de inyecciones activas.
- Valve Status: El estado de las válvulas, si están encendidas o apagadas.
- Stirrer Status: El estado de los agitadores, si están encendidos o apagados.
- Digital Inputs: Estado de las entradas digitales.

- Digital Inputs NPN: Estado de las entradas digitales dedicadas a NPN
- Bubble Status: Estado de los sensores de burbujas
- ADC Externo: El valor del ADC externo
- Temperature: La temperatura de la PCB
- Humidity: La humedad del ambiente
- Pressure Ch1: Presión leída por el sensor del canal 1
- Pressure Ch2: Presión leída por el sensor del canal 2
- Hall Ch1: Numero de vueltas detectadas por el sensor hall del canal 1
- Hall Ch2: Numero de vueltas detectadas por el sensor hall del canal 2
- Alarms Status: El estado de las alarmas a tiempo real.

La manera de almacenar toda esta información es mediante un array delimitando por 60 instantes temporales y en el cual se almacena la información de la dirección de memoria del registro, la información que este posee y el número de bytes que le atañen.

4.4.3 Protocolo CAN

El protocolo CAN (Controller Area Network) ha sido elegido como la interfaz de comunicación para la transmisión y recepción de tramas en este proyecto. Esta decisión se tomó en gran parte debido a la experiencia previa de la empresa en el desarrollo de otros tipos de tarjetas que también emplean este protocolo. La implementación de CAN en la tarjeta AIS facilitaría su integración con las demás tarjetas ya existentes, lo que simplifica la interoperabilidad entre diferentes dispositivos dentro del mismo ecosistema de la empresa.

Además de la compatibilidad con otros proyectos, el protocolo CAN ofrece una serie de ventajas técnicas significativas que lo hacen ideal para este tipo de aplicaciones:

- **Velocidad**: CAN es capaz de operar a una alta velocidad de transmisión, alcanzando hasta 1 megabit por segundo. Esta velocidad es crucial en sistemas donde el tiempo de respuesta es crítico.

- **Seguridad**: La robustez del protocolo CAN se manifiesta en su mecanismo de verificación de errores, mediante un CRC (Cyclic Redundancy Check) extenso en comparación con los datos transmitidos. Esto le proporciona una alta distancia de Hamming (valor de 6), lo que convierte al bus CAN en una de las redes de comunicación más seguras disponibles en el mercado.

- **Inteligencia**: Una de las principales fortalezas de CAN es su capacidad de arbitraje inteligente, que permite que cualquier estación conectada pueda transmitir en cualquier momento, sin riesgo de colisiones o pérdida de datos. Este mecanismo prioriza los datos más importantes y permite que todos los nodos en la red reciban y procesen los datos según sea necesario. Cada dato transmitido tiene un Identificador único, que no solo distingue los datos, sino que también establece su prioridad dentro del sistema.

Para gestionar la configuración de las direcciones de memoria, las tramas de datos y los controles necesarios, se creó un archivo Excel con una serie de tablas que resumen las características de cada registro utilizado en el proyecto. Este Excel se encuentra añadido en los anexos de la memoria del proyecto.

Para gestionar la información de los registros en el sistema, se han creado cuatro variables en formato de array bidimensional. Estas variables se dividen en dos categorías principales: dos de ellas están dedicadas a la gestión de comandos generales y las otras dos a comandos específicos.

Variables de Información Predefinida

Una de las variables de cada categoría almacena información predefinida, es decir, valores que no pueden ser modificados durante la ejecución del programa. Estas variables estáticas están denominadas como ALL_CAN_0X, donde X toma el valor de 2 para los comandos genéricos y 3 para los comandos específicos. En estos arrays se almacena la dirección de cada registro, los datos asociados y el DLC (Data Length Code), que indica el número de bytes que contiene cada registro. Debido a su naturaleza estática, los valores en estas variables permanecen constantes a lo largo de la ejecución del programa, proporcionando una referencia fija para el manejo de los registros.

Variables de Información Dinámica

Por otro lado, se encuentran las variables dinámicas denominadas CAN_lookup_table_0x. Estas variables son del tipo volatile, lo que permite que su contenido se actualice continuamente en función de los comandos recibidos o los procesos en curso dentro del programa. Al igual que las variables estáticas, X toma el valor de 2 para los comandos genéricos y 3 para los comandos específicos. Estas tablas de búsqueda son esenciales para la gestión dinámica del sistema, ya que reflejan el estado actual de los registros y permiten que el programa reaccione y se ajuste en tiempo real a las operaciones que se están ejecutando.

En resumen, mientras que las variables estáticas (ALL_CAN_0X) proporcionan una referencia constante para los registros, las variables dinámicas (CAN_lookup_table_0x) permiten una gestión flexible y adaptable de la información en función del comportamiento del sistema.

Capitulo V: DLL y Programa de Aplicación

5.2 DLL

5.2.1 ¿Qué es una DLL?

Una Dynamic Link Library (DLL)[15] es un archivo que contiene código y datos que pueden ser utilizados simultáneamente por múltiples programas en un sistema operativo Windows. Las DLLs permiten a los desarrolladores encapsular funciones comunes, como rutinas matemáticas, controladores de dispositivos o componentes de la interfaz de usuario, en un solo lugar. Esto facilita la reutilización del código y reduce la redundancia, ya que varias aplicaciones pueden compartir la misma biblioteca de funciones sin necesidad de incluir el código directamente en sus ejecutables.

En el contexto de este proyecto, las DLLs se utilizan para facilitar la interacción entre el software desarrollado y el hardware específico, permitiendo una modularidad y una actualización más sencilla de componentes. Por ejemplo, se pueden cargar dinámicamente bibliotecas que contienen rutinas de control para diferentes periféricos del microcontrolador o para manejar protocolos de comunicación como CAN. Esto es especialmente útil en sistemas embebidos, donde se requiere una gestión eficiente de los recursos y la capacidad de actualizar o modificar funcionalidades sin necesidad de recompilar todo el software.

El uso de DLLs en este proyecto no solo mejora la eficiencia del desarrollo al permitir la reutilización de código preexistente, sino que también garantiza que las actualizaciones de ciertos módulos o funciones puedan realizarse de manera ágil, sin impactar negativamente en el resto del sistema. Este enfoque modular y dinámico es fundamental para mantener la flexibilidad y escalabilidad del proyecto, permitiendo adaptaciones futuras con un mínimo esfuerzo.

5.2.2 PCharNetComm

En el proyecto, la DLL ha sido denominada *PCharNetComm*. Esta biblioteca dinámica encapsula diferentes clases, cada una diseñada para facilitar la comunicación con las diversas tarjetas electrónicas que han sido desarrolladas o son controladas por la empresa PolymerChar. Las clases dentro de PCharNetComm están organizadas jerárquicamente, lo que significa que algunas clases derivan sus propiedades y métodos de otras clases, siguiendo un modelo de herencia. Esta estructura permite que las funciones y atributos comunes se gestionen de manera centralizada, mejorando la eficiencia y la coherencia del código.

Dentro de esta DLL, se ha diseñado y creado una clase particular dedicada a la NMB. Esta clase incluye todas las funciones específicas que son esenciales para el testeo y la comunicación de la NMB. Al centralizar estas funciones dentro de una única clase en la DLL, se facilita el mantenimiento y la extensión del código, permitiendo que nuevas funcionalidades se añadan de manera más ordenada y con menor riesgo de generar conflictos o redundancias en el sistema.

La creación de PCharNetComm no solo estandariza el proceso de comunicación entre el software y el hardware desarrollado por PolymerChar, sino que también proporciona una base sólida y flexible para futuras expansiones del sistema, asegurando que cualquier nuevo dispositivo o tarjeta que se desarrolle en el futuro pueda integrarse sin problemas en la infraestructura existente.

En los sistemas desarrollados para los equipos de PolymerChar, las electrónicas pueden establecer comunicación con un PC mediante dos métodos principales: a través de un puerto serie directamente conectado al PC o mediante una combinación de interfaces, donde la

comunicación inicial se realiza a través de un puerto serie o un bus CAN, que luego se dirige a una tarjeta electrónica intermedia, esta electrónica intermedia encargada de gestionar esta comunicación se denomina NTC y ha sido diseñada y desarrollada por la empresa. Esta tarjeta intermedia actúa como un controlador de comunicaciones, enlazando las electrónicas del sistema con el PC a través de una conexión Ethernet.

El primer método, que involucra una conexión directa por puerto serie al PC. En este escenario, cada electrónica se conecta individualmente al PC, y es útil cuando se quiere testear o comprobar el funcionamiento de una electrónica aislándola del resto.

El segundo método introduce un mayor nivel de complejidad y flexibilidad. Este enfoque es especialmente útil en sistemas donde se requieren conexiones con múltiples dispositivos o donde es necesario gestionar un volumen elevado de datos o dispositivos distribuidos. La tarjeta de control centraliza las comunicaciones, reduciendo la carga sobre el PC y facilitando la integración de varios sistemas electrónicos bajo un único canal de comunicación.

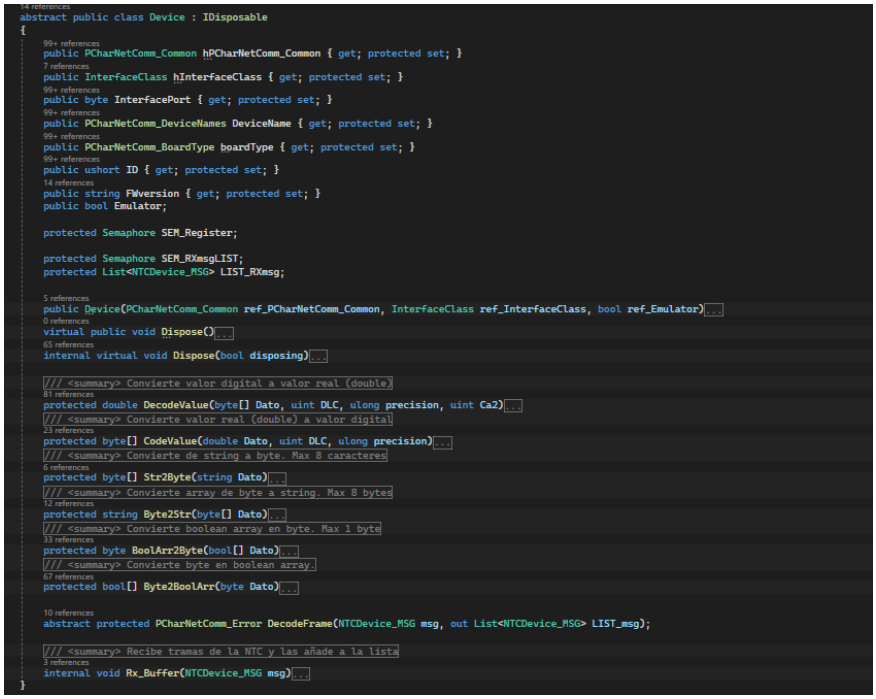
El uso de Ethernet como medio de comunicación entre la tarjeta de control y el PC también aporta ventajas significativas, como la posibilidad de operar en redes locales o remotas, una mayor velocidad de transferencia de datos, y la capacidad de gestionar comunicaciones sobre distancias más largas que las posibles con un puerto serie tradicional.

En el proyecto, la NMB utiliza el segundo método de comunicación mencionado, que opera a través de un bus CAN.

Para gestionar estas comunicaciones en la DLL desarrollada, se ha estructurado el código en diferentes clases basadas en el tipo de comunicación y el tipo de dispositivo. En primer lugar, se crearon dos clases específicas para manejar los distintos tipos de comunicación: la clase *NTC_Interface*, que gestiona la comunicación a través del bus CAN mediante la NTC, y la clase *Computer_Interface*, que se encarga de la comunicación directa con el PC.

En cuanto a la organización por tipo de dispositivo, se implementó una jerarquía de clases en la que todos los dispositivos heredan de una clase base llamada *Device*. Sin embargo, para las electrónicas desarrolladas específicamente por la empresa, se creó una clase derivada llamada *PCharDevice*, que hereda de *Device*. Esta estructura jerárquica permite una gestión más eficiente y organizada de las funciones y propiedades inherentes a cada tipo de dispositivo, garantizando que las especificidades de las electrónicas de PolymerChar sean adecuadamente manejadas en el entorno de la DLL.

La clase jerárquica que sigue la DLL hasta llegar a la Clase NMB que es la realizada para el desarrollo de la aplicación posterior es la siguiente:

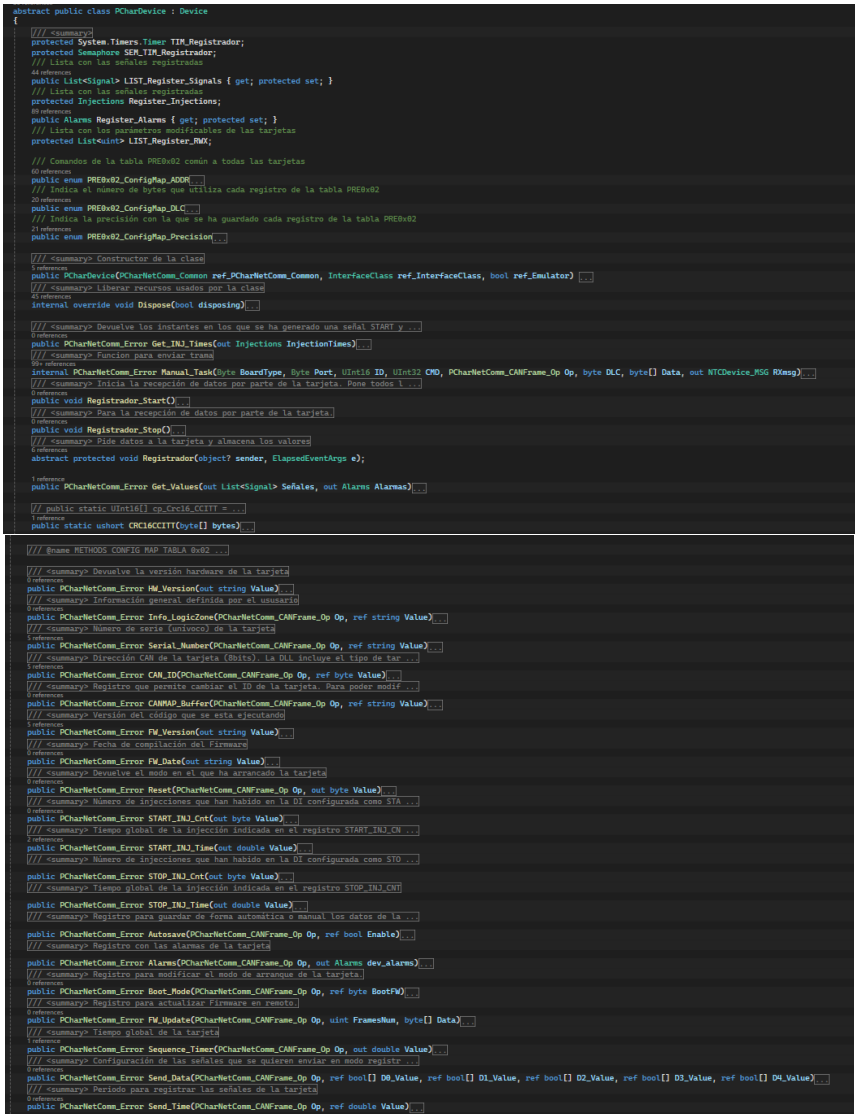


```
14 references
abstract public class Device : IDisposable
{
    99+ references
    public PCharNetComm_Common hPCharNetComm_Common { get; protected set; }
    7 references
    public InterfaceClass hInterfaceClass { get; protected set; }
    99+ references
    public byte InterfacePort { get; protected set; }
    99+ references
    public PCharNetComm_DeviceNames DeviceName { get; protected set; }
    99+ references
    public PCharNetComm_BoardType boardType { get; protected set; }
    99+ references
    public ushort ID { get; protected set; }
    14 references
    public string FWversion { get; protected set; }
    public bool Emulator;

    protected Semaphore SEM_Register;

    protected Semaphore SEM_RXmsgLIST;
    protected List<NTCDevice_MSG> LIST_RXmsg;

    5 references
    public Device(PCharNetComm_Common ref_PCharNetComm_Common, InterfaceClass ref_InterfaceClass, bool ref_Emulator)...
    0 references
    virtual public void Dispose()...
    65 references
    internal virtual void Dispose(bool disposing)...

    /// <summary> Convierte valor digital a valor real (double)
    81 references
    protected double DecodeValue(byte[] Dato, uint DLC, ulong precision, uint Ca2)...
    /// <summary> Convierte valor real (double) a valor digital
    23 references
    protected byte[] CodeValue(double Dato, uint DLC, ulong precision)...
    /// <summary> Convierte de string a byte. Max 8 caracteres
    6 references
    protected byte[] Str2Byte(string Dato)...
    /// <summary> Convierte array de byte a string. Max 8 bytes
    12 references
    protected string Byte2Str(byte[] Dato)...
    /// <summary> Convierte boolean array en byte. Max 1 byte
    33 references
    protected byte BoolArr2Byte(bool[] Dato)...
    /// <summary> Convierte byte en boolean array.
    67 references
    protected bool[] Byte2BoolArr(byte Dato)...

    10 references
    abstract protected PCharNetComm_Error DecodeFrame(NTCDevice_MSG msg, out List<NTCDevice_MSG> LIST_msg);

    /// <summary> Recibe tramas de la NTC y las añade a la lista
    3 references
    internal void Rx_Buffer(NTCDevice_MSG msg)...
}
```

29. Clase Device

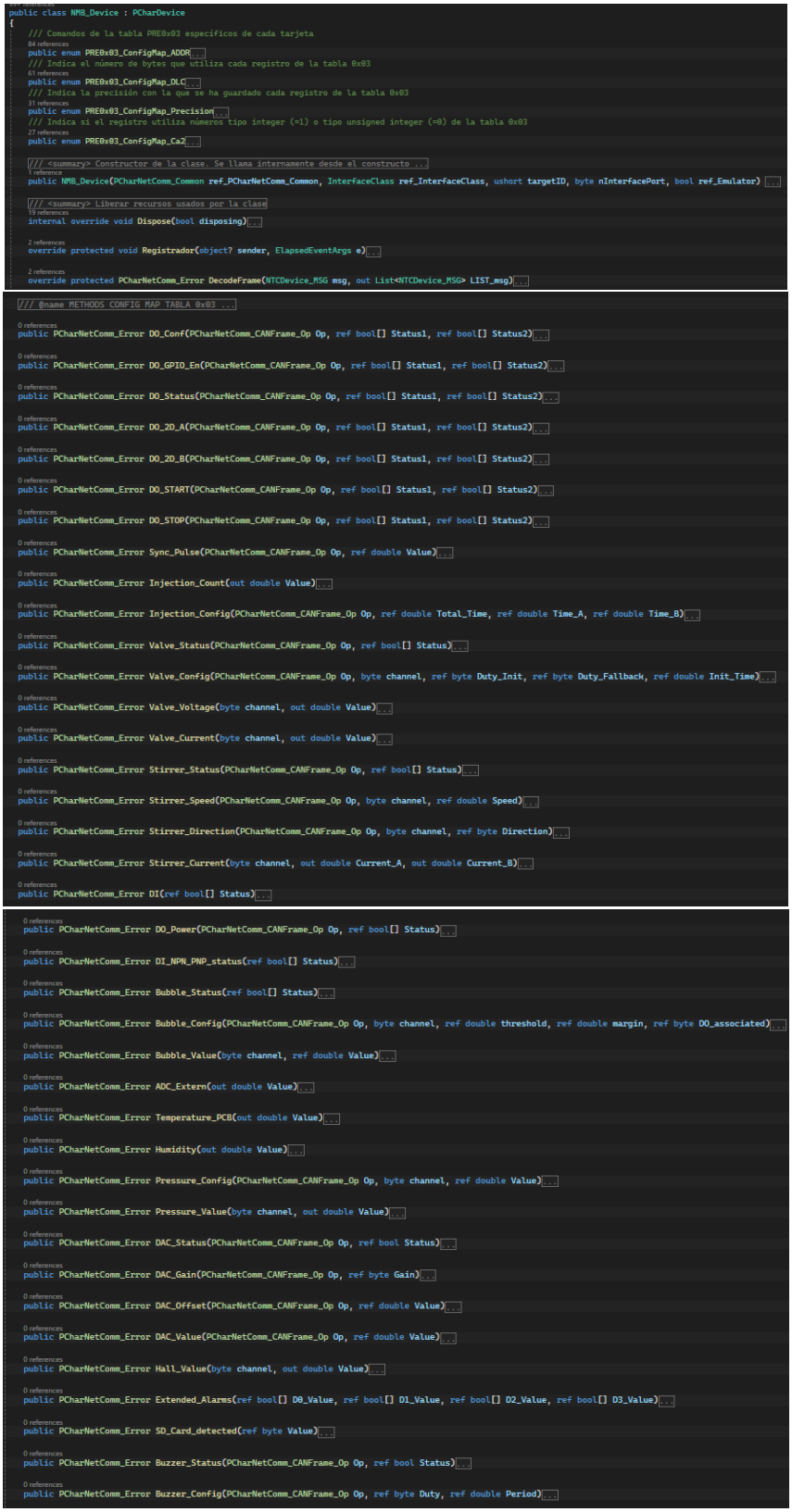
En la parte más alta de la jerarquía se encuentra la clase *Device*, que actúa como una referencia general para todos los dispositivos con los que la empresa ha logrado establecer comunicación. Esta clase es fundamental, ya que en ella se definen los semáforos necesarios para controlar el acceso y manejo de datos, lo cual es crucial para evitar conflictos durante la comunicación entre dispositivos. Además, *Device* incluye listas para almacenar datos, así como diversas funciones destinadas a su procesamiento.

Las funciones dentro de la clase *Device* están organizadas de acuerdo con su propósito y nivel de accesibilidad. Algunas funciones son privadas, lo que significa que solo pueden ser utilizadas dentro de la misma clase; otras son protegidas, permitiendo su acceso desde clases derivadas; y finalmente, existen funciones públicas que pueden ser utilizadas desde cualquier lugar en el que se instancie la clase o sus derivadas.



30. Clase PCharDevice

Por otro lado, se encuentra la clase *PCharDevice*, que hereda todas las propiedades y características de la clase *Device*. Esta clase específica se encarga de implementar y gestionar todas las funciones comunes a las electrónicas diseñadas por la empresa. Es decir, en *PCharDevice* se centralizan todas las funcionalidades relacionadas con los comandos generales, tal como se ha detallado en la sección del protocolo CAN en el capítulo dedicado al desarrollo del firmware.



31. Clase NMB

Finalmente, en la jerarquía de clases se encuentra la clase *NMB*, que es la más específica y está diseñada exclusivamente para el control y gestión de la tarjeta electrónica *NMB* desarrollada en

este proyecto. Esta clase hereda todas las propiedades y métodos de la clase *PCharDevice*, la cual, a su vez, deriva de la clase base *Device*. Esta cadena de herencia garantiza que la clase *NMB* no solo tenga acceso a las funcionalidades generales definidas para todos los dispositivos en *Device*, sino también a las operaciones específicas que son comunes a todos los dispositivos diseñados por la empresa, proporcionadas por *PCharDevice*.

La clase *NMB* incorpora todas las funciones necesarias para gestionar los comandos específicos de la tarjeta *NMB*. Estos comandos son esenciales para el correcto funcionamiento de la tarjeta electrónica y para facilitar la comunicación entre la tarjeta y la aplicación principal. Al centralizar estas funciones en la clase *NMB*, se logra simplificar enormemente el desarrollo de la aplicación, permitiendo que las operaciones complejas se manejen de manera sencilla y eficiente.

Además, cada función dentro de la clase *NMB* está diseñada con bloques try-catch, lo que añade un nivel significativo de robustez al sistema. Estos bloques de manejo de excepciones permiten que la aplicación pueda gestionar errores de manera controlada, evitando que el programa se congele o deje de funcionar ante problemas inesperados. Esto no solo mejora la estabilidad de la aplicación, sino que también facilita la depuración y el mantenimiento del software, ya que los errores pueden ser capturados y gestionados sin interrumpir el flujo normal de operaciones.

5.3 PCharTools

A partir de la DLL previamente mencionada en el apartado anterior, se ha desarrollado una aplicación de interfaz de usuario diseñada específicamente para el control y monitoreo de la tarjeta electrónica NMB, esta recibe el nombre de PCharTools. Esta aplicación no solo actúa como un medio para interactuar con la electrónica, sino que también proporciona un acceso completo y detallado a toda la información que la tarjeta es capaz de ofrecer.

El desarrollo de esta aplicación se ha centrado en maximizar la utilidad y la accesibilidad de los datos proporcionados por la tarjeta electrónica. A través de la interfaz de usuario, los operadores pueden consultar en tiempo real el estado de diversos parámetros operativos, permitiendo una supervisión continua del dispositivo. Esto incluye la lectura de valores críticos, la verificación de estados de operación, y la recepción de cualquier mensaje de error o alarma generados por la tarjeta.

Además, la aplicación ofrece la posibilidad de enviar comandos y configuraciones a la tarjeta, lo que permite modificar su comportamiento y ajustar su operación según las necesidades del usuario. La integración de esta funcionalidad es fundamental, ya que permite un control total sobre el dispositivo sin necesidad de acceder físicamente al hardware.

La interfaz ha sido diseñada con el usuario final en mente, buscando una experiencia intuitiva y fácil de usar, lo que facilita tanto a técnicos como a ingenieros realizar pruebas, diagnósticos y ajustes en la tarjeta de manera eficiente. Este enfoque centrado en el usuario también asegura que la aplicación pueda ser utilizada en diversos entornos, ya sea en un laboratorio de desarrollo o en un entorno de producción.

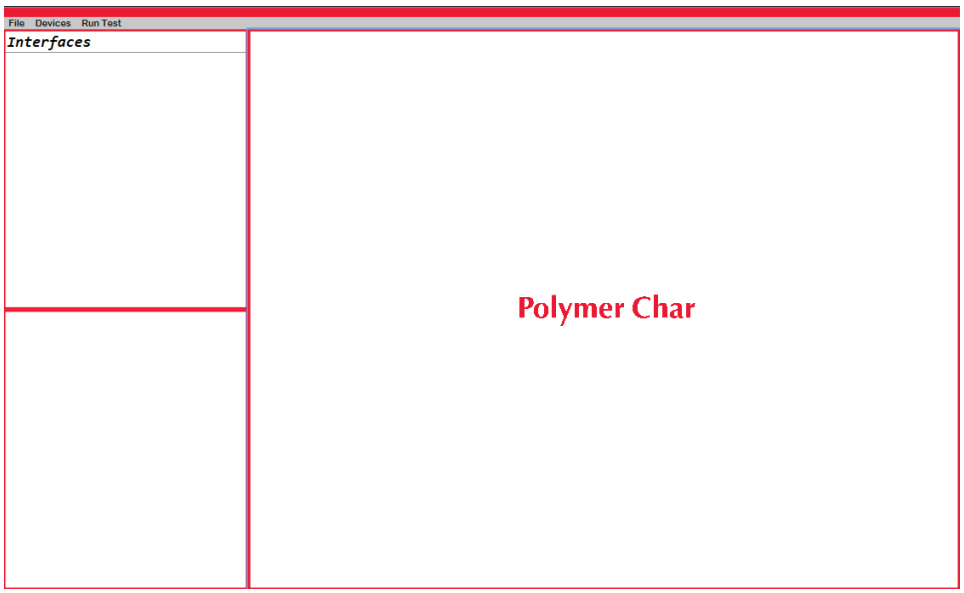
La aplicación se ha dividido en diferentes panales para así poder tener toda la información más estructurada.

5.3.1 Front Pannel

En esta sección de la aplicación se encuentra la pantalla principal, que sirve como el eje central desde el cual el usuario puede acceder a todos los dispositivos que han sido añadidos. Estos dispositivos se presentan en una vista de árbol, lo que significa que cada dispositivo aparece con

sus respectivas subventanas o subcategorías de funciones a las que el usuario puede acceder según sea necesario.

Además, es importante destacar que esta ventana principal siempre permanece visible en la interfaz, sin importar qué otras vistas o subventanas estén abiertas en la aplicación. Esto se debe a que la ventana principal actúa como el eje central o punto de referencia para todas las operaciones dentro de la aplicación. Cualquier vista adicional o subventana que el usuario abra se posiciona dentro de los espacios o paneles que la ventana principal define, asegurando que el acceso al árbol de dispositivos nunca se pierda de vista.

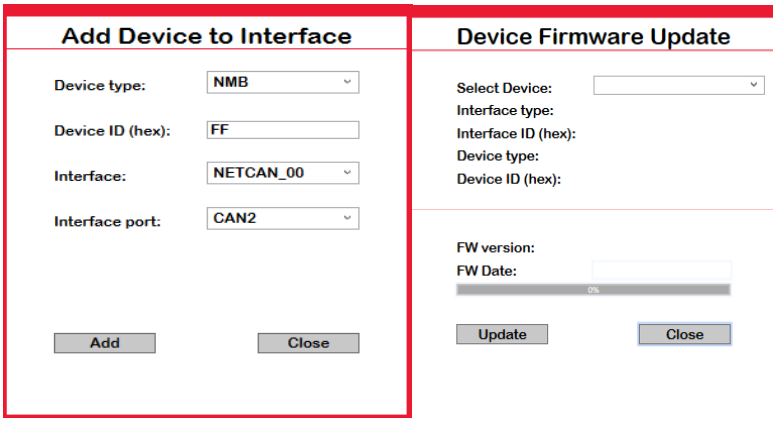


32. Front Pannel UI

En la pestaña "Devices" de la aplicación se encuentran disponibles varias ventanas comunes a todas las tarjetas electrónicas conectadas. Estas ventanas permiten realizar acciones esenciales de gestión y mantenimiento del hardware.

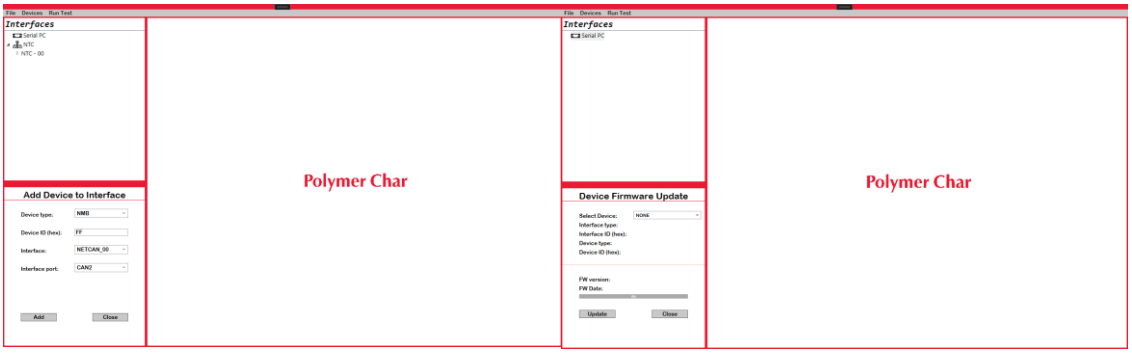
La primera ventana disponible en esta pestaña es la de cambio de ID del dispositivo. Esta funcionalidad es crucial para personalizar y diferenciar cada dispositivo dentro de la red, especialmente cuando se utilizan múltiples unidades similares. Cambiar el ID permite una identificación clara y precisa de cada tarjeta electrónica, facilitando su manejo en el sistema.

La segunda ventana es la que permite la actualización remota del firmware. Esta opción es fundamental para mantener el software de las tarjetas electrónicas al día, implementando mejoras, corrigiendo errores, y añadiendo nuevas funciones sin necesidad de acceso físico a las mismas. La capacidad de actualizar el firmware de forma remota optimiza el mantenimiento y mejora la operatividad de los dispositivos en el campo.



33. Change ID & Firmware Update

Estas ventanas adicionales están situadas en la parte inferior izquierda de la interfaz. Su objetivo principal es proporcionar al usuario información complementaria que no se muestra en el cuadro principal de la pantalla.



34. Change ID y Firmware Update en Front Pannel

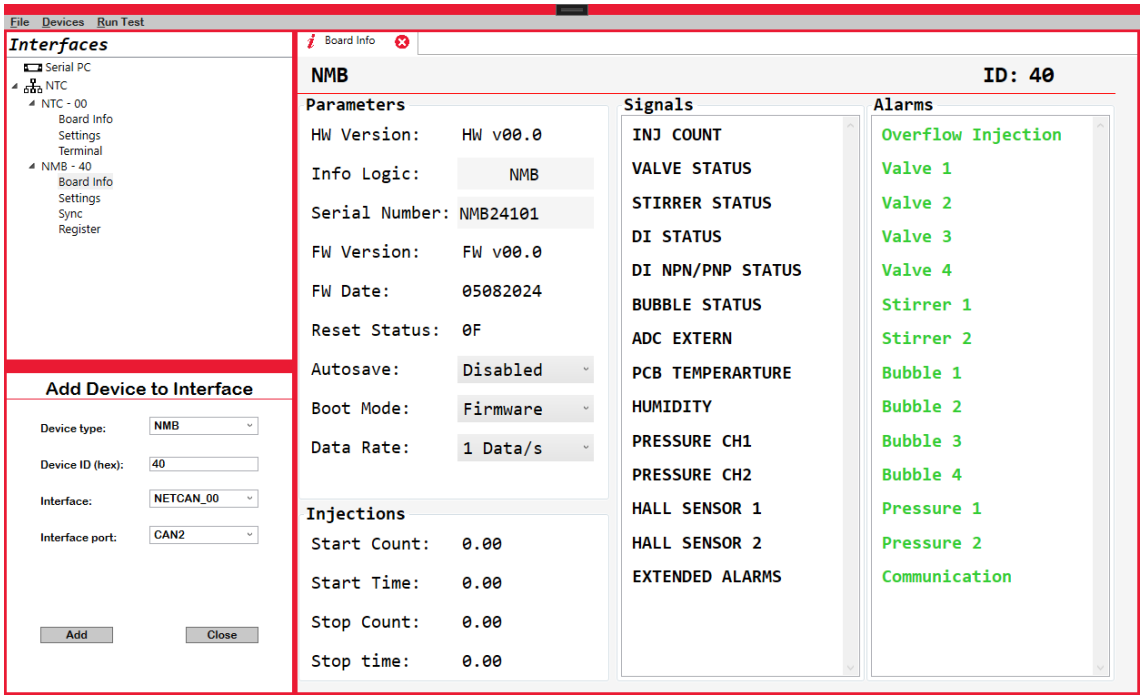
5.3.2 Comandos Genéricos

Para gestionar los comandos genéricos, se utiliza una ventana específica en la aplicación que proporciona acceso a toda la información relevante compartida por todas las tarjetas electrónicas diseñadas por la empresa.

En esta ventana, el usuario puede visualizar diversos detalles críticos de la tarjeta, como la versión de hardware, la versión de firmware y la fecha de compilación, así como el número de serie de la electrónica entre otros muchos valores.

Además, la ventana ofrece un listado de las señales que están siendo registradas en tiempo real.

Por último, se presenta un listado de todas las alarmas asociadas con la electrónica. A través de un sistema de *binding* en tiempo real, el usuario puede identificar de inmediato si alguna alarma se ha activado. En condiciones normales, el texto de las alarmas aparece en verde, indicando un funcionamiento correcto. Sin embargo, si se detecta un problema, el texto correspondiente cambia a rojo, alertando al usuario sobre la presencia de una alarma activa.

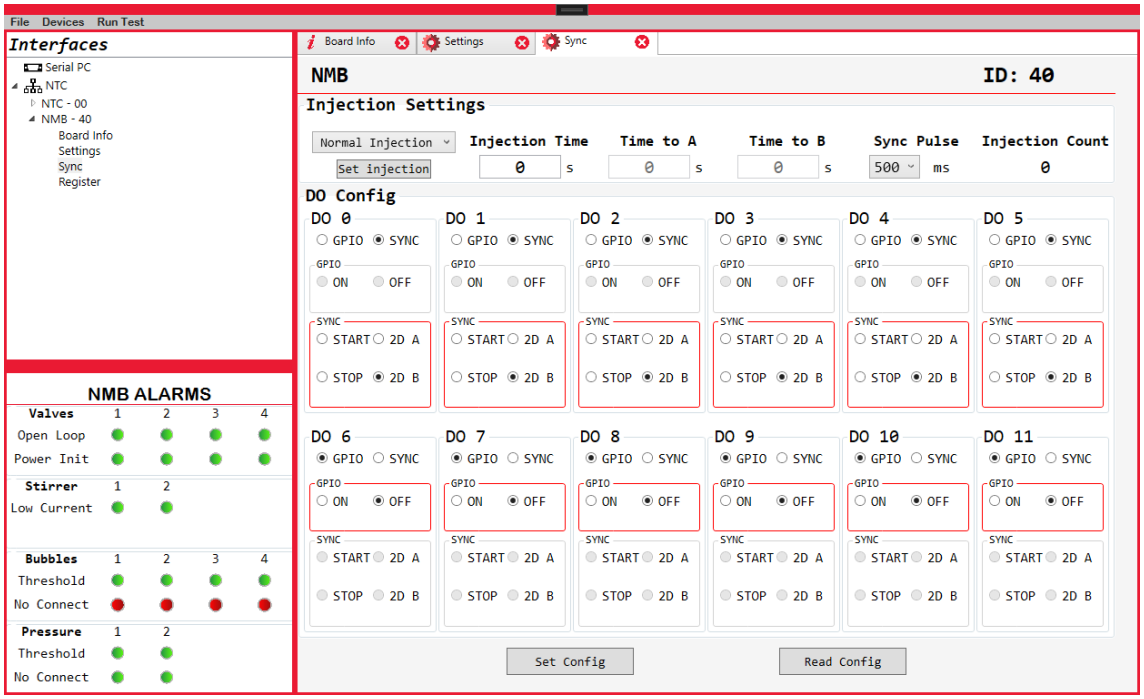


35. Common Data View

5.3.3 Comandos específicos

Por último, se encuentran las ventanas dedicadas a los comandos específicos. Estas ventanas son diferentes para cada tipo de tarjeta programada en la DLL, ya que cada una tiene una estructura particular debido a las distintas funciones que desempeñan. Esta diferenciación es necesaria para adaptar la organización y presentación de la información a las características únicas de cada dispositivo.

En primer lugar, se encuentra la ventana de las inyecciones, esta recibe el nombre de **Injection Settings**. Se decidió separar la configuración de las inyecciones en una ventana independiente debido a la importancia crítica de esta función en la electrónica, la cual debe operar sin margen de error.

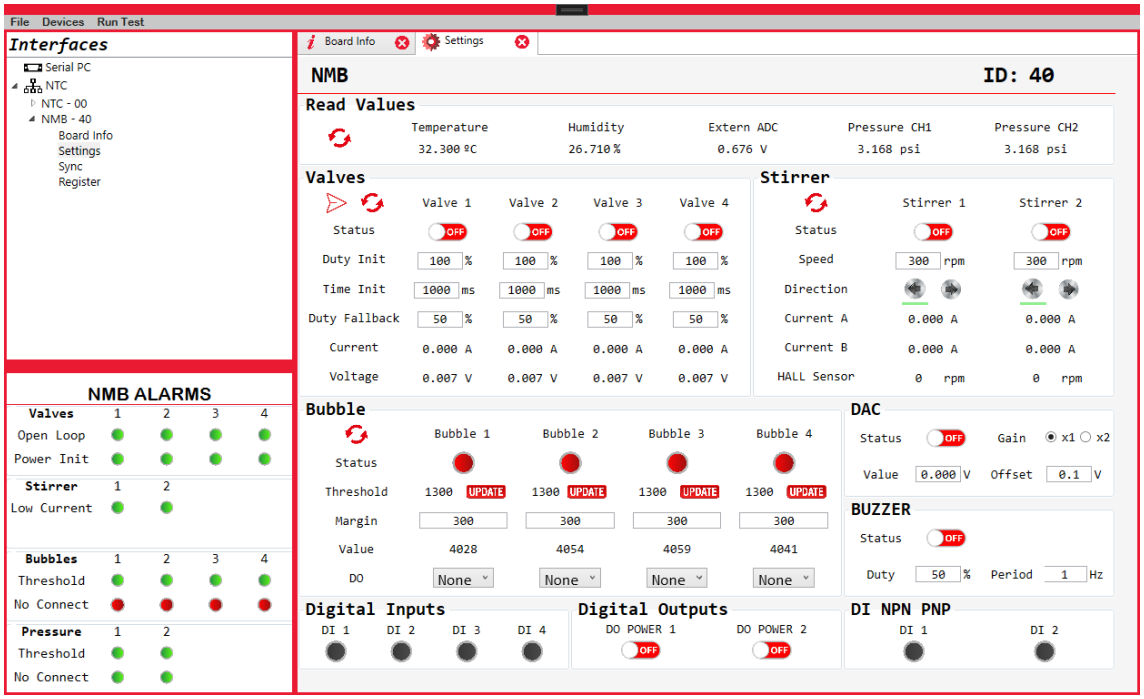


36.Injection Sync View

En esta ventana, es posible configurar el tipo de inyección deseado, ya sea una inyección normal o una inyección en dos dimensiones. Dependiendo de la opción seleccionada, la aplicación permitirá al usuario ajustar los tiempos de inyección que el sistema puede manejar, además de configurar las salidas digitales de manera específica según el tipo de inyección que se desee implementar.

Por otro lado, mientras una inyección esté en curso, la aplicación no permitirá realizar cambios en las salidas configuradas como sincronismo, con el fin de proteger el estado de las inyecciones. El número de inyecciones es un parámetro que se registra y, gracias a un sistema de binding, es posible conocer en tiempo real si hay alguna inyección en proceso.

Por último, se encuentra la ventana donde es posible obtener información del resto de sensores que hay en la electrónica. Esta ventana recibe el nombre de **NMB Settings**



37. NMB Settings View

En esta ventana, es posible realizar una serie de acciones fundamentales para el control y monitoreo en tiempo real de la tarjeta electrónica. A continuación, se detallan las principales funcionalidades:

- **Registro en tiempo real de señales**: La ventana permite visualizar en tiempo real varios parámetros cruciales, como la temperatura, la humedad, el valor del ADC externo, la presión en ambos canales y el estado de las entradas digitales, tanto de propósito general como de tipo NPN. Esto facilita el monitoreo constante de las condiciones operativas de la tarjeta.
- **Estado de las alarmas**: En la parte inferior izquierda de la ventana principal, aparece una sección dedicada al estado de las alarmas de la tarjeta cuando se está visualizando la configuración de NMB. Esta sección está vinculada directamente a la ventana principal y proporciona información instantánea sobre cualquier alerta que pueda surgir.
- **Control de válvulas**: Es posible configurar los parámetros iniciales de las válvulas, así como encenderlas y apagarlas. Además, se pueden monitorear la corriente y el voltaje que pasan por ellas, lo que permite detectar posibles fallos y asegurar un funcionamiento adecuado.
- **Control de los agitadores (Stirrer)**: La ventana permite encender y apagar los agitadores, ajustar su velocidad en revoluciones por minuto y definir la dirección de giro. También incluye lecturas de corriente y tensión para diagnosticar el funcionamiento correcto. Además, dos sensores Hall proporcionan información precisa sobre la velocidad de rotación de los agitadores.
- **Monitoreo de burbujas**: La ventana ofrece información sobre el estado de las burbujas y permite configurar el umbral (threshold) al pulsar el botón de actualización, lo que establecerá el valor actual como el nuevo umbral. También es posible ajustar el margen

para ofrecer mayor flexibilidad y asignar una salida digital a cada sensor de burbujas, lo que permite realizar una acción activa en caso de fallo.

- **Control del DAC (Convertidor Digital a Analógico)**: El DAC se puede encender y apagar desde la ventana. Además, es posible modificar parámetros como la ganancia, el offset y el valor de salida, ajustando el funcionamiento según las necesidades del sistema.
- **Control del buzzer**: Por defecto, el buzzer está apagado, pero es posible activarlo para pruebas y ajustes, modificando su período y ciclo de trabajo (duty cycle).
- **Gestión de las salidas digitales**: Las salidas digitales tienen una funcionalidad básica, permitiendo simplemente encenderlas o apagarlas. Al encender una salida digital, un LED indicativo se iluminará en la tarjeta, confirmando que está activa.

  Estas funcionalidades permiten un control exhaustivo y personalizado del sistema, asegurando que cada componente opere dentro de los parámetros deseados y ofreciendo la posibilidad de intervenir rápidamente en caso de detectar anomalías o necesidades de ajuste.

Capítulo VI: Conclusión y Líneas Futuras

Este proyecto ha logrado con éxito el diseño y desarrollo de un sistema integral compuesto por firmware, una Dynamic-Link Library (DLL) y una aplicación de prueba, cumpliendo con todos los objetivos inicialmente planteados.

El objetivo principal era desarrollar un firmware capaz de controlar una nueva tarjeta electrónica, destinada a sustituir a las electrónicas subcontratadas previamente por la empresa. Este cambio permite a la empresa tener un mayor control sobre su hardware, lo que conlleva importantes ventajas, como la capacidad de realizar mejoras continuas y personalizadas en los dispositivos. Además, somos conocedores de la capacidad ocupada en la memoria flash por el firmware, esta es de 38%, este factor es importante ya que en las electrónicas anteriores era un factor que limitante

El cumplimiento de este objetivo es evidente, ya que la tarjeta electrónica desarrollada ha sido depurada y validada, y posteriormente transferida al departamento de software de la empresa. Esto ha permitido que el equipo de desarrollo de software pueda diseñar una interfaz de usuario directamente interactiva con la nueva tarjeta, facilitando la operación y mejorando la experiencia del usuario final.

Además del firmware, la inclusión a la DLL corporativa fue otro de los objetivos clave del proyecto. La DLL desarrollada ha permitido una integración eficiente entre la nueva tarjeta electrónica y las aplicaciones de software de nivel superior. Esta biblioteca dinámica no solo facilita la comunicación entre el hardware y el software, sino que también proporciona una base sólida para futuras expansiones y mejoras en las funcionalidades del sistema. La DLL ha sido optimizada para garantizar una latencia mínima en la transmisión de datos y una alta fiabilidad en la gestión de las conexiones, lo que asegura que las operaciones se realicen de manera fluida y sin interrupciones. Esto gracias a la comunicación por bus CAN se ha podido mejorar a nivel de rapidez de comunicación ya que este bus tiene una tasa de transferencia de datos de hasta 1 megabit/s en comparación a los 9600 bits/s que funcionaba por puerto serie anteriormente.

Finalmente, la aplicación de prueba desarrollada como parte de este proyecto ha sido diseñada para cumplir con los requisitos de testeo y validación interna. Esta aplicación permite a Polymer Char realizar diagnósticos exhaustivos de la nueva tarjeta electrónica, garantizando que todas las funciones operen según lo previsto antes de su implementación en entornos de producción. Esto ha sido un gran avance ya que en las electrónicas antiguas carecíamos de herramientas sofisticadas que pudieran aportar información al usuario si había surgido algún error.

En cuanto a las líneas futuras, se plantean varios caminos para mejorar y expandir este proyecto. En primer lugar, la migración hacia plataformas de hardware más avanzadas y con mayor capacidad podría permitir la inclusión de nuevas funcionalidades que mejoren aún más el control y la precisión de los equipos. Asimismo, la integración de tecnologías emergentes como el Internet de las Cosas (IoT) podría potenciar la conectividad y el monitoreo remoto de los dispositivos, facilitando el mantenimiento preventivo y reduciendo los tiempos de inactividad. Finalmente, se recomienda explorar la implementación de inteligencia artificial para el análisis predictivo de datos obtenidos por los sensores, lo cual podría revolucionar la capacidad de anticipar fallos y optimizar los procesos en la industria petroquímica.

Capitulo VII: Bibliografía

[1] "Polymer Char - Wikipedia, la enciclopedia libre." Accessed: Sep. 02, 2024. [Online]. Available: https://es.wikipedia.org/wiki/Polymer_Char

[2] "Objetivos y metas de desarrollo sostenible - Desarrollo Sostenible." Accessed: Sep. 02, 2024. [Online]. Available: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[3] "GPC-IR® | High Temperature GPC/SEC System for Polyolefins - Polymer Char." Accessed: Sep. 02, 2024. [Online]. Available: https://polymerchar.com/products/analytical-instruments/gpc-ir

[4] "IR4 | Infrared Detector for Polyolefins - Polymer Char." Accessed: Sep. 02, 2024. [Online]. Available: https://polymerchar.com/products/ir4

[5] "STM32 Microcontrollers (MCUs) - STMicroelectronics." Accessed: Sep. 02, 2024. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

[6] "Visual Studio: IDE y Editor de código para desarrolladores de software y Teams." Accessed: Sep. 02, 2024. [Online]. Available: https://visualstudio.microsoft.com/es/

[7] "WPF vs. WinForms - The complete WPF tutorial." Accessed: Sep. 02, 2024. [Online]. Available: https://wpf-tutorial.com/es/2/acerca-de-wpf/wpf-vs-winforms/

[8] "¿Qué es la BIOS? (Basic Input Output System) - IONOS España." Accessed: Sep. 02, 2024. [Online]. Available: https://www.ionos.es/digitalguide/servidores/know-how/que-es-la-bios-de-un-ordenador/

[9] "Principios básicos del sensor de efecto Hall - Urany®." Accessed: Sep. 02, 2024. [Online]. Available: https://urany.net/blog/adquiere-precisi%C3%B3n-con-efecto-hall

[10] "Conversión analógica-digital - Wikipedia, la enciclopedia libre." Accessed: Sep. 02, 2024. [Online]. Available: https://es.wikipedia.org/wiki/Conversi%C3%B3n_anal%C3%B3gica-digital

[11] "Sensores de presión Archivi - Gefran s.p.a." Accessed: Sep. 02, 2024. [Online]. Available: https://www.gefran.es/productos/sensores-de-presion/

[12] "Honeywell - The Future Is What We Make It." Accessed: Sep. 02, 2024. [Online]. Available: https://www.honeywell.com/us/en

[13] "STM32F407VG - High-performance foundation line, Arm Cortex-M4 core with DSP and FPU, 1 Mbyte of Flash memory, 168 MHz CPU, ART Accelerator, Ethernet, FSMC - STMicroelectronics." Accessed: Sep. 02, 2024. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f407vg.html

[14] "Que es pwm y como funciona." Accessed: Sep. 02, 2024. [Online]. Available: https://www.shoptronica.com/curiosidades-tutoriales-y-gadgets/4517-que-es-pwm-y-como-funciona-0689593953254.html

[15] "Biblioteca de enlace dinámico - Wikipedia, la enciclopedia libre." Accessed: Sep. 02, 2024. [Online]. Available: https://es.wikipedia.org/wiki/Biblioteca_de_enlace_din%C3%A1mico

Capitulo VIII: Anexos

Tabla	CMD	Op.	Flash	WP	W	Default	DLC	Descripción	Info
TABLA 0X02 - COMANDOS GENERALES									
0x02	0x000		-	-	-	DataH: 0x20202020 DataL: 0x20202020 String value: " "	8	HW Version	Versión Identificativo PCB en formato char
0x02	0x001	0x04: Enable writing	X	X	X	DataH: 0x20202020 DataL: 0x20202020 String value: " "	8	Info Logic Device	Nombre identificativo del dispositivo
0x02	0x002	0x04: Enable writing	X	X	X	DataH: 0x4E4D4230 DataL: 0x30303030 String value: "NMB00000"	8	Número de serie	Número de serie de cada tarjeta
0x02	0x003	0x04: Enable writing	X	X	X	DataH: 0xFFFFFFFF DataL: 0xFFFFFFFF	1	Standard CAN ID (Low Address)	Dirección CAN asignada a la tarjeta: CAN_ID (b7 - b0)
0x02	0x004		-	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFFFFFF	8	CAN Mapping Buffer PCB	Registro para cambiar CAN_ID. D0 - D7 : Serial number que se quiere cambiar
0x02	0x005		-	-	-	DataH: 0x5658582E DataL: 0x58582020 String value: "VXX.XX "	8	FW Version	Versión identificativa del FW. "VXX.YY " donde X es el numero de versión. XX cambio grande YY cambio pequeño
0X02	0x006		-	-	-	-	8	FW Date	Fecha compilación del Firmware en ASCII y con el siguiente formato: DDMMAAAA
0x02	0x007	0x04: Reset board	-	-	-	-	1	Reset	Estado del registro interno del micro.
0x02	0x008		-	-	-	-	1	Start Injection count	Numero de inyecciones totales
0x02	0x009		-	-	-	-	8	Start Injection time	D0 - D3: Valor absoluto del tiempo de la tarjeta. Resolución acorde a Sequence_Timer D4 - D7: Numero de veces que desborda la variable T1_Sequence_Timer
0x02	0x00A		-	-	-	-	1	Stop Injection count	Numero de inyecciones totales
0x02	0x00B		-	-	-	-	8	Stop Injection time	D0 - D3: Valor absoluto del tiempo de la tarjeta. Resolución acorde a Sequence_Timer D4 - D7: Numero de veces que desborda la variable T1_Sequence_Timer
0x02	0x00C	0x04: Save now	-	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFFFF00 Logic value: NoAuto	1	Auto save flash	0: Auto save flash disable 1: Auto save flash enable
0x02	0x00D	0x04: Clear Alarms	-	-	-	-	2	Alarmas	Muestra las alarmas generadas: D0: b00: Overflow injection b01: Valve 1 b02: Valve 2 b03: Valve 3 b04: Valve 4 b05: Stirrer 1 b06: Stirrer 2 b07: Bubble 1 D1: b00: Bubble 2 b01: Bubble 3 b02: Bubble 4 b03: Pressure 1 b04: Pressure 2 b05: Communication b06: None b07: None
0x02	0x00E	0x04: Jump to FW	X	-	X		1	Boot Mode	Configura la forma de arranque: b0 = 0: Bootloader b0 = 1: Firmware b0 = 2 : Error
0x02	0x00F	0x04 : Fw package 0x05 : Save Fw	-	-	X		8	FW Update	D0-D1: Tamaño de tramas a recibir (size/8 - MAX: 256K) D2-D3: CRC *Habilitación solo en código BIOS
0x02	0x010	0x04: Reset time	-	-	-	-	8	Sequence Timer	D0 - D3: Valor absoluto del tiempo de la tarjeta. Resolución de 1ms D4 - D7: Numero de veces que desborda la variable T1_Sequence_Timer
0x02	0x011	0x04 : Send data	X	-	X	DataH: 0xFFFFFF00 DataL: 0x000000003	5	Send_Data	Configura como se envían los datos al BUS: bx = 0 : Manual bx = 1 : Registrador D0: b0 = none b1 = none b2 = none b3 = none b4 = none b5 = none b6 = none b7 = none --- Como respuesta a Send data: D0: Numero de páginas a enviar. D1 - D2: Alarmas de la tarjeta D3: Start Injection count D4: Stop injection count
0x02	0x012		X	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFF03E8 Numeric value: 1000	2	Send_Time	Tiempo en milisegundos para registrar datos.

TABLA 0X03 - COMANDOS ESPECÍFICOS								
0x03	0x000	**0x04** : Pone a GPIO las Salidas Digitales indicadas en D0 y D1 **0x05** : Pone a SYNC las Salidas Digitales indicadas en D0 y D1	X	X	X	DataH: 0xFFFFFFFF DataL: 0xFFFFFFFF Value: ALL SYNC	2	DO CONF	SABER SI ES SYNC O GPIO 0 = GPIO // 1 = SYNC
0x03	0x001	**0x04** : Pone a Disable las Salidas Digitales GPIO indicadas en D0 y D1 **0x05** : Pone a Enable las Salidas Digitales GPIO indicadas en D0 y D1	X	X	X	DataH: 0xFFFFFFFF DataL: 0xFFFF0000 Value: ALL DISABLE	2	DO GPIO ENABLE	Configuras de forma permanente la salida indicada como una GPIO asociado a un recurso 0 = Disable // 1 = Enable D0: b0: DO 0 b1: DO 1 b2: DO 2 b3: DO 3 b4: DO 4 b5: DO 5 b6: DO 6 b7: DO 7 D1: b0: DO 8 b1: DO 9 b2: DO 10 b3: DO 11
0x03	0x002	**0x04** : Pone a OFF las Salidas Digitales GPIO indicadas en D0 y D1 **0x05** : Pone a ON las Salidas Digitales GPIO indicadas en D0 y D1	-	-	X	-	2	DO STATUS	0 = Off // 1 = On D0: b0: DO 0 b1: DO 1 b2: DO 2 b3: DO 3 b4: DO 4 b5: DO 5 b6: DO 6 b7: DO 7 D1: b0: DO 8 b1: DO 9 b2: DO 10 b3: DO 11
0x03	0x003	**0x04** : Pone a OFF las Salidas Digitales 2D A indicadas en D0 y D1 **0x05** : Pone a ON las Salidas Digitales 2D A indicadas en D0 y D1	X	X	X	DataH: 0xFFFFFFFF DataL: 0xFFFF0000	2	DO 2D A	**Idem DO STATUS**
0x03	0x004	**0x04** : Pone a OFF las Salidas Digitales 2D B indicadas en D0 y D1 **0x05** : Pone a ON las Salidas Digitales 2D B indicadas en D0 y D1	X	X	X	DataH: 0xFFFFFFFF DataL: 0xFFFF0000	2	DO 2D B	**Idem DO STATUS**
0x03	0x005	**0x04** : Pone a OFF las Salidas Digitales START indicadas en D0 y D1 **0x05** : Pone a ON las Salidas Digitales START indicadas en D0 y D1	X	X	X	DataH: 0xFFFFFFFF DataL: 0xFFFF0000	2	DO START	**Idem DO STATUS**
0x03	0x006	**0x04** : Pone a OFF las Salidas Digitales STOP indicadas en D0 y D1 **0x05** : Pone a ON las Salidas Digitales STOP indicadas en D0 y D1	X	X	X	DataH: 0xFFFFFFFF DataL: 0xFFFF0000	2	DO STOP	**Idem DO STATUS**
0x03	0x007		X	X	X	DataH: 0xFFFFFFFF DataL: 0xFFFFFF05 Value: 500 ms	1	SYNC PULSE	Resolucion de 100 ms Rango de 100 ms a 900 ms

0x03	0x008		-	-	-	-	1	INJ Running Count	Numero de inyecciones activas o en cola Si la cuenta es mayor a 0 activo el bit WP de los registros: - DO CONF - DO 2D A - DO 2D B - DO START - DO STOP - SYNC PULSE Si es igual a 0 se desactiva el bit WP
0x03	0x009	0x04: Normal Injection 0x05: SGIC -2D Injection	-	-	-	-	8	INJECT CONF	Se reciben los datos de tiempo en resolucion de 100 ms D0 - D3: TOTAL TIME INJECTION D4 - D5: TIME TO A D6 - D7: TIME TO B
0x03	0x010	0x04 : Pone a OFF las Valvulas indicadas en D0 0x05 : Pone a ON las valvulas indicadas en D0	-	-	X	-	1	Valve Status	0 = Off // 1 = On b0: Valve 1 b1: Valve 2 b2: Valve 3 b3: Valve 4
0x03	0x011	0x04: Valve Parker 33% 0x05: Valve Neptum 50% 0x06: Valve Burkert 40%	X	-	X	DataH: 0xFFFFFFFF DataL: 0x03E83264 Value: 1 seg, 50%, 100%	4	Valve Config 1	D0: Duty Init. Value en % D1: Duty Fallback Value en % D2 - D3: Time Init. Value en ms
0x03	0x012	0x04: Valve Parker 0x05: Valve Neptum 0x06: Valve Burkert	X	-	X	DataH: 0xFFFFFFFF DataL: 0x03E83264 Value: 1 seg, 50%, 100%	4	Valve Config 2	**Idem a Config Valve 1**
0x03	0x013	0x04: Valve Parker 0x05: Valve Neptum 0x06: Valve Burkert	X	-	X	DataH: 0xFFFFFFFF DataL: 0x03E83264 Value: 1 seg, 50%, 100%	4	Valve Config 3	**Idem a Config Valve 1**
0x03	0x014	0x04: Valve Parker 0x05: Valve Neptum 0x06: Valve Burkert	X	-	X	DataH: 0xFFFFFFFF DataL: 0x03E83264 Value: 1 seg, 50%, 100%	4	Valve Config 4	**Idem a Config Valve 1**
0x03	0x015		-	-	-	-	2	Valve Voltage 1	D0 - D1: Voltage Value Valor en voltios de la valvula 1. Resolucion en 1mV
0x03	0x016		-	-	-	-	2	Valve Voltage 2	**Idem a Voltage Valve 1**
0x03	0x017		-	-	-	-	2	Valve Voltage 3	**Idem a Voltage Valve 1**
0x03	0x018		-	-	-	-	2	Valve Voltage 4	**Idem a Voltage Valve 1**
0x03	0x019		-	-	-	-	2	Valve Current 1	D0 -D1: Currente Value Valor en amperios de la valvula 1. Resolucion en 1mA
0x03	0x01A		-	-	-	-	2	Valve Current 2	**Idem a Current Valve 1**
0x03	0x01B		-	-	-	-	2	Valve Current 3	**Idem a Current Valve 1**
0x03	0x01C		-	-	-	-	2	Valve Current 4	**Idem a Current Valve 1**
0x03	0x01D	0x04 : Pone a OFF las Valvulas indicadas en D0 0x05 : Pone a ON las valvulas indicadas en D0	-	-	X	-	1	Stirrer Status 1	D0: Status 0 = Off 1 = On
0x03	0x01E		X	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFF012C Value: 300 rpm	2	Stirrer Speed 1	D0 - D1: Velocidad Stirrer en rpm; MAX: 1200 rpm Resolucion 1 rpm
0x03	0x01F		X	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFF012C Value: 300 rpm	2	Stirrer Speed 2	**Idem a Speed Stirrer 1**
0x03	0x020		X	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFFFFF0 Value: Derecha	1	Stirrer Direction 1	D0: Direccion del stirrer 0: Izquierda 1: Derecha
0x03	0x021		X	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFFFFF0 Value: Derecha	1	Stirrer Direction 2	**Idem a Direction Stirrer 1**
0x03	0x022		-	-	-	-	2	Stirrer Current 1 A	D0 - D1: Corriente Stirrer 1 Valor en Amperios Resolucion 1 mA
0x03	0x023		-	-	-	-	2	Stirrer Current 1 B	**Idem a Current Stirrer 1 A**
0x03	0x024		-	-	-	-	2	Stirrer Current 2 A	**Idem a Current Stirrer 1 A**
0x03	0x025		-	-	-	-	2	Stirrer Current 2 B	**Idem a Current Stirrer 1 A**
0x03	0x026		-	-	-	-	1	Digital Inputs	b0: DI 0 b1: DI 1 b2: DI 2 b3: DI 3 bx = 0 : OFF bx = 1 : ON

0x03	0x027	0x04 : Pone a "0" las salidas indicadas en D0 0x05 : Pone a "1" las salidas indicadas en D0	X	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFFFFF0 Value:OFF / OFF	1	Digital Output PWR	bx = 0 : OFF bx = 1 : ON
0x03	0x028		-	-	-	-	1	Digital Input NPN/PNP	b0: Status 0 = Connect 1 = No connect
0x03	0x029		-	-	-	-	1	Bubble Status	b0: Indica si hay burbuja o no 0 = No hay Burbuja 1 = Hay Burbuja
0x03	0x02A	0x04 calibrar	X	-	X	DataH: 0xFFFFF0C DataL: 0x0514012C Value: None;1300;300	5	Bubble Config 1	D0-D1: Margen D2-D3: Valor del umbral establecido D4: DO asociada 0 -> D0 1 -> D1 ... 11 -> D11 12 -> None
0x03	0x02B	0x04 calibrar	X	-	X	DataH: 0xFFFFFFFF DataL: 0x0514012C Value:1300;300	5	Bubble Config 2	**Idem a Config Bubble 1**
0x03	0x02C	0x04 calibrar	X	-	X	DataH: 0xFFFFFFFF DataL: 0x0514012C Value:1300;300	5	Bubble Config 3	**Idem a Config Bubble 1**
0x03	0x02D	0x04 calibrar	X	-	X	DataH: 0xFFFFFFFF DataL: 0x0514012C Value:1300;300	5	Bubble Config 4	**Idem a Config Bubble 1**
0x03	0x02E		-	-	-		2	Bubble value 1	Valor Digital de 0 a 4028
0x03	0x02F		-	-	-		2	Bubble Value 2	
0x03	0x030		-	-	-		2	Bubble Value 3	
0x03	0x031		-	-	-		2	Bubble Value 4	
0x03	0x032		-	-	-	-	2	Extern ADC	Valor en voltios, Resolucion en 100uV
0x03	0x033		-	-	-	-	2	Temperature PCB	Resolución en 0.01ºC
0x03	0x034		-	-	-	-	2	Humidity	Resolución en 0.01%
0x03	0x035		X	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFF02D0 Value:72.0	4	Pressure Config 1	D0 - D1: Contante K para utilizar en la conversion a psi D2 -D3: Threshold
0x03	0x036		X	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFF02D0 Value:72.0	4	Pressure Config 2	**Idem a Config Pressure 1**
0x03	0x037		-	-	-	-	2	Pressure Value 1	Valor en voltios del sensor de Presion. Resolución de 1mV.
0x03	0x038		-	-	-		2	Pressure Value 2	**Idem a Pressure 1**
0x03	0x039		-	-	X	-	1	Status DAC	D0: 0 = Disable 1 = Enable
0x03	0x03A		X	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFFFFF0 Value: x1	1	Gain DAC	D0: 0 = x1 1 = x2
0x03	0x03B		X	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFF0064 Value:100	2	Offset DAC	Resolucion de milivoltios
0x03	0x03C		-	-	X	-	2	Value DAC	Rango entre 0 y (2.5 x GAIN) V GAIN -> x1 o x2 D0 - D1: Valor en voltios. Resolución de 1mV.
0x03	0x03D		-	-	-	-	2	Value Hall 1	Rpm medidos
0x03	0x03E		-	-	-	-	2	Value Hall 2	**Idem a Value Hall 1**
0x03	0x03F	0x04: Clear Data	-	-	-	-	4	Alarms Ext	D0: Valve b0: Open Loop Valve 1 b1: Open Loop Valve 2 b2: Open Loop Valve 3 b3: Open Loop Valve 4 b4: Init Valve 1 b5: Init Valve 2 b6: Init Valve 3 b7: Init Valve 4 D1: Stirrer b0: Stirrer 1 Current b1: Stirrer 2 Current D2: Bubble b0: Bubble 1 b1: Bubble 2 b2: Bubble 3 b3: Bubble 4 b4: No connect Bubble 1 b5: No connect Bubble 2 b6: No connect Bubble 3 b7: No connect Bubble 4 D3: Pressure b0: Threshold Pressure 1 b1: Threshold Pressure 2 b2: No connect Pressure 1 b3: No connect Pressure 2
0X03	0x040	0x04: Clear Data	-	-	-	-	1	SD Card	D0: 0 = No hay tarjeta insertada 1 = Hay tarjeta insertada
0x03	0x041	0x04: Disable 0x05: Enable 0x06: Hacer sonar diferentes alarmas 0x07...	-	-	X	-	1	Buzzer Status	bx = 0 : OFF bx = 1 : ON
0x03	0x042		X	-	X	DataH: 0xFFFFFFFF DataL: 0xFFFFFF32 Value:50%	1	Buzzer Config	D0: Porcentaje de Duty encendido D1: Period