Universidad Nacional de Mar del Plata
Facultad de Ingeniería

Informe Final de Ingeniería en Computación e Ingeniería en Electrónica

Can Xplorer: Desarrollo de un Lector y Analizador Avanzado de Sistemas CAN Bus en Vehículos

Bauer Gonzalo Ramiro (Ingeniería en Computación),
Cecchetti Dante Agustín (Ingeniería en Electrónica)

Director: Ing. Walter Andrés Gemin
Co-director: Ing. Juan Manuel López

Definiciones, acrónimos y abreviaturas

Nombre	Descripción
ADC	Analog to Digital Converter
AEC-Q	Automotive Electronics Council
BLE	Bluetooth Low Energy
CAN	Control Area Network
DC	Direct Current
DTC	Diagnostic Trouble Code
ECU	Engine Control Unit
ESD	Electrostatic Discharge
ESL	Equivalent Series Inductance
FDCAN	Flexible Data-Rate Controller Area Network
HS	High Speed
ISO	International Organization for Standardization
ISR	Interrupt Service Routine
LED	Light Emisor Diode
LS	Low Speed
MCU	Microcontroller unit
MS	Medium Speed
OBD-II	On-Board Diagnostics (version 2)
PCB	Printed Circuit Board

PID	Parameter ID
Ppm	Partes por millón
SAE	Society of Automotive Engineers
SCL	Serial Clock Line
SDA	Serial Data Line
SMD	Surface Mount Device
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus

Resumen

El presente proyecto tiene como objetivo el desarrollo de un sistema avanzado para la lectura y análisis de datos provenientes del bus CAN en vehículos, denominado Can Xplorer. El sistema se compone de un dispositivo físico, capaz de establecer comunicación con la red vehicular mediante la interfaz OBD-II, y una aplicación móvil compatible con Android, encargada de brindar una interfaz gráfica amigable e interactiva para el usuario final.

El dispositivo desarrollado se compone de dos microcontroladores interconectados: uno dedicado a la comunicación con el vehículo, encargado de la adquisición y procesamiento de datos provenientes del bus CAN, y otro orientado a la gestión de la comunicación inalámbrica con un dispositivo móvil mediante tecnología Bluetooth. Esta arquitectura permite separar funciones críticas y optimizar el rendimiento del sistema.

El software del proyecto incluye un firmware modular, encargado de gestionar la comunicación bidireccional entre el bus CAN de alta velocidad del vehículo y la aplicación móvil, y una aplicación desarrollada en Java con funciones de visualización en tiempo real, lectura y borrado de códigos de error (DTC), y personalización de los parámetros visualizados por el usuario.

Como resultado, se obtiene un sistema portátil y de bajo costo, diseñado para técnicos, ingenieros y usuarios particulares. Una característica distintiva del dispositivo es su capacidad de escalabilidad, sustentada en un hardware preparado para admitir futuras ampliaciones. Esta plataforma ofrece el potencial de incorporar protocolos de comunicación actuales, explorar buses no estandarizados y extender las capacidades de diagnóstico hacia niveles cercanos a los utilizados en entornos profesionales e industriales. Esto habilita un desarrollo posterior sin necesidad de rediseños estructurales, permitiendo que el sistema crezca en funcionalidad a medida que lo requieran nuevas aplicaciones o tecnologías automotrices.

Capítulo 1

Introducción

1.1. Marco contextual

El proyecto implica la creación de un sistema de lectura, escritura y visualización de datos para sistemas CAN[10] Bus en vehículos. El sistema está compuesto por un dispositivo físico capaz de conectarse a la interfaz OBD-II de los vehículos, a fin de interactuar con el bus de datos estandarizado del protocolo CAN, y por una aplicación móvil desarrollada para la plataforma Android que actuará como interfaz gráfica e interactiva para el usuario final. La comunicación entre el dispositivo y la aplicación móvil se establecerá de forma inalámbrica mediante tecnología Bluetooth LE (Low Energy) , permitiendo así un uso práctico para el usuario final. El producto resultante está destinado a los siguientes usuarios:

- Técnicos, Ingenieros y/o aficionados que utilicen esta herramienta como instrumento de adquisición de datos.
- Talleres mecánicos para el chequeo del vehículo automotor y sus funcionalidades electrónicas
- Usuario de vehículo para conocer el estado actual de su vehículo y/o de otros vehículos de interés.

Este proyecto surge a partir de la experiencia de uno de los desarrolladores de este proyecto, Dante Cecchetti, quien utilizo lector OBD-II comercial en su vehículo para obtener información desde la ECU. Sin embargo, detectó que la velocidad de transferencia de datos de dicho dispositivo resultaba insuficiente. Por lo tanto, el objetivo principal del presente desarrollo es lograr una mayor tasa de adquisición de datos, así como diseñar el hardware necesario para comunicarse con buses no estandarizados, a fin de crear una placa de desarrollo versátil que sirva como base para un futuro producto comercial.

En cuanto a la metodología, se adoptó una estrategia de desarrollo por etapas, iniciando con una fase de capacitación en la cual se estudiaron los estándares normativos que rigen la comunicación CAN, como la norma ISO 11898[12], además del análisis de bibliotecas y recursos de código abierto para la implementación de módulos CAN. A partir de allí, se avanzó con el diseño del hardware, tomando como referencia desarrollos previos y realizando una selección crítica de componentes que garantizara la escalabilidad del sistema a futuro.

De forma paralela al diseño del hardware, se desarrolló el software del sistema, abarcando tanto la aplicación móvil como el firmware del microcontrolador. Durante este proceso se incorporó la comunicación con el bus CAN, junto con la comunicación inalámbrica mediante Bluetooth Low Energy y la comunicación interna entre microcontroladores mediante I²C, permitiendo el envío de los datos del vehículo hacia la aplicación. Una vez alcanzada la estabilidad funcional del firmware, se realizaron pruebas tanto en un entorno simulado como en un vehículo real, lo que permitió ajustar el sistema y mejorar su desempeño general.

En cuanto a la organización del presente informe, el mismo se estructura de la siguiente manera:

- Capítulo 1: Presenta la introducción y el marco contextual del proyecto.
- Capítulo 2: Describe el anteproyecto, incluyendo los requerimientos, especificaciones funcionales y restricciones.
- Capítulo 3: Expone el diseño electrónico y la evolución del sistema desarrollado.
- Capítulo 4: Detalla el diseño del software, abarcando tanto el firmware como la aplicación móvil.
- Capítulo 5: Explica el plan de pruebas llevado a cabo y expone los resultados obtenidos.
- Capítulo 6: Se exponen las conclusiones del trabajo realizado.

Adicionalmente, los anexos contienen la información complementaria correspondiente a cálculos, diseños detallados y justificaciones técnicas de cada bloque del hardware implementado.

Finalmente, resulta pertinente señalar que el presente proyecto, al ser llevado adelante por estudiantes pertenecientes a dos disciplinas distintas, Ingeniería

Electrónica e Ingeniería en Computación, podría plantear tanto desafíos como oportunidades a lo largo del desarrollo. La necesidad de articular enfoques, metodologías y modos de trabajo propios de cada área podría implicar ciertas dificultades iniciales, especialmente al momento de unificar criterios técnicos o de coordinar responsabilidades. No obstante, esta misma diversidad formativa podría constituir una fortaleza significativa, al aportar perspectivas complementarias para el análisis y la resolución de problemas. Asimismo, el trabajo conjunto permitiría abordar de manera paralela etapas diferenciadas del proyecto, como el diseño del hardware y el desarrollo del software, favoreciendo una distribución más equilibrada de tareas.

1.2. Protocolo CAN

1.2.1. Principios y fundamentos del protocolo

El protocolo Controller Area Network (CAN) es un estándar de comunicación en red desarrollado por la empresa Bosch a mediados de la década de 1980, inicialmente destinado al sector automotriz. Desde entonces, se ha consolidado como una solución robusta, eficiente y económica para la interconexión de unidades electrónicas dentro de vehículos y sistemas industriales. Este protocolo implementa una arquitectura de topología en bus y se caracteriza por ser un sistema de comunicación multi-maestro orientado a mensajes, lo cual permite que cualquier nodo conectado al bus pueda iniciar una transmisión siempre que el medio se encuentre libre. Esta característica posibilita un acceso descentralizado al canal, favoreciendo la flexibilidad del sistema.

A diferencia de otros protocolos que utilizan direcciones físicas para identificar dispositivos, CAN emplea identificadores de mensaje, lo que facilita la configuración dinámica, la jerarquización de prioridades y una escalabilidad eficiente de la red. Entre las principales ventajas de este tipo de comunicación se destacan:

- **Simplicidad y bajo costo:** Las Unidades de Control Electrónico (ECU) se comunican a través de un único bus compartido, eliminando la necesidad de múltiples conexiones punto a punto. Esto reduce significativamente el

cableado, el peso total del sistema, la probabilidad de fallos y los costos de implementación.

- **Arquitectura centralizada:** El bus CAN actúa como un punto común de acceso para todas las ECU, lo que permite implementar funcionalidades como diagnóstico centralizado, registro de datos en tiempo real y configuración remota.
- **Alta robustez:** Gracias al uso de una comunicación diferencial sobre dos líneas (CAN H y CAN L), el sistema presenta una elevada inmunidad frente a interferencias electromagnéticas y perturbaciones eléctricas, siendo ideal para aplicaciones críticas en entornos hostiles.
- **Eficiencia en la transmisión:** Las tramas CAN se priorizan mediante identificadores, permitiendo que los mensajes más importantes accedan al bus sin demoras, sin comprometer la integridad de las transmisiones restantes.

En cuanto al formato de los mensajes, el protocolo CAN define la estructura de la información mediante tramas normalizadas, tal como lo establece la especificación CAN 2.0, que contempla dos variantes:

- El formato estándar (CAN 2.0A), con identificadores de 11 bits, ampliamente utilizado en automóviles.
- El formato extendido (CAN 2.0B), con identificadores de 29 bits, destinado a aplicaciones industriales y vehículos de gran porte.

Por otro lado, ante la creciente demanda de mayor ancho de banda y velocidad en aplicaciones modernas, se introdujo una evolución del estándar: CAN FD (Flexible Data Rate). Este protocolo permite velocidades de transmisión de hasta 8 Mbit/s, una carga útil de hasta 64 bytes por trama (en contraste con los 8 bytes de CAN 2.0) y ofrece mejoras relacionadas con la seguridad de los datos, incluyendo mecanismos de autenticación. Estas características convierten a CAN FD en la tecnología preferida para los sistemas automotrices de última generación.

1.2.2. Interfaz OBD-II: acceso físico al bus CAN

El puerto OBD-II (On-Board Diagnostics, segunda generación) es un estándar de diagnóstico a bordo implementado en vehículos a partir de 1996 en Estados Unidos, como evolución del sistema OBD-I. Este sistema permite la supervisión continua y el reporte en tiempo real del estado operativo de múltiples subsistemas del vehículo, incluyendo el control del motor, la gestión de emisiones y otros componentes electrónicos esenciales para el funcionamiento y la seguridad del automóvil.

El conector físico asociado al sistema OBD-II está normalizado bajo la especificación SAE J1962[11], la cual define un conector hembra de 16 pines (Figura 1) con dimensiones, distribución de contactos y características eléctricas estandarizadas. Este conector se encuentra generalmente en el habitáculo, en una posición de fácil acceso para tareas de diagnóstico, mantenimiento o inspección técnica vehicular.

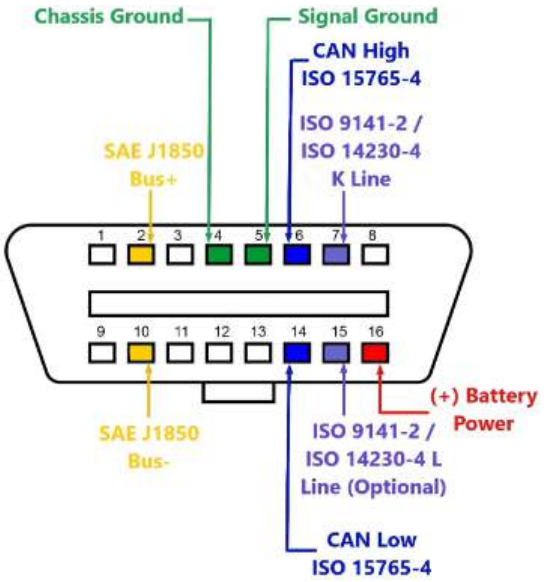


Figura 1. Puerto OBD-II con sus pines estándar señalizados.

Como se muestra en la Figura 1, algunos de los pines del conector están estandarizados y reservados para protocolos específicos de comunicación automotriz:

- **Pin 2 – SAE J1850 Bus (+)**: Línea positiva de comunicación para vehículos que utilizan el protocolo J1850 (PWM o VPW).
- **Pin 4 – Chassis Ground**: Tierra del chasis del vehículo, proporciona referencia de masa para el sistema.
- **Pin 5 – Signal Ground**: Tierra de señales electrónicas, referencia de masa para dispositivos de diagnóstico.
- **Pin 6 – CAN High (ISO 15765-4)**: Línea alta del bus CAN de alta velocidad, trabaja en modo diferencial junto con CAN Low.
- **Pin 7 – K-Line (ISO 9141-2 / ISO 14230-4)**: Línea de comunicación serial usada en protocolos ISO más antiguos (KWP y OBD en vehículos previos al CAN).
- **Pin 10 – SAE J1850 Bus (-)**: Línea negativa de comunicación para el protocolo J1850 PWM (no se utiliza en VPW).
- **Pin 14 – CAN Low (ISO 15765-4)**: Línea baja del bus CAN, complementaria a CAN High para transmisión diferencial.
- **Pin 15 – L-Line (ISO 9141-2 / ISO 14230-4)**: Línea secundaria utilizada para inicialización de algunos protocolos ISO.
- **Pin 16 – Battery Power (+12 V)**: Alimentación directa de la batería del vehículo.

Los pines no listados corresponden a conexiones de uso libre por parte de los fabricantes, quienes pueden asignarles funciones personalizadas según las características del vehículo o los sistemas propietarios que implementen.
En el contexto del presente proyecto, los pines relevantes son: el pin 16 (alimentación de batería), el pin 5 (masa lógica), y el par 6 y 14, correspondientes al bus CAN de alta velocidad, que es además el único bus CAN estandarizado dentro del conector OBD-II, correspondiente con la norma ISO 15765-4.
Es importante destacar que muchos vehículos modernos integran tres tipos de buses CAN que operan a distintas velocidades y cumplen funciones diferenciadas:

- **CAN de alta velocidad (hasta 1 Mbit/s)**: utilizado en sistemas críticos como motor, frenos ABS y airbags.

- **CAN de velocidad media (típicamente hasta 500 kbit/s)**: destinado a funciones de confort, como aire acondicionado, cierre centralizado o iluminación.
- **CAN de baja velocidad (hasta 125 kbit/s, con tolerancia a fallos)**: empleado en subsistemas secundarios, donde la robustez es prioritaria frente a la velocidad de transmisión.

No obstante, el presente proyecto se enfoca exclusivamente en el bus CAN de alta velocidad, ya que es el encargado de transportar datos relevantes para el diagnóstico dinámico y el monitoreo de variables fundamentales del funcionamiento vehicular.

Capítulo 2:

Anteproyecto

2.1. Análisis de soluciones

A partir de la experiencia previa de Dante Cecchetti, quien instaló en su vehículo un módulo destinado a obtener información de la ECU mediante un lector OBD-II comercial basado en el microcontrolador ELM327, una plataforma que, si bien fue ampliamente utilizada en sus inicios, hoy presenta limitaciones frente a las demandas actuales de adquisición de datos. En particular, se detectó que la velocidad de transferencia de datos obtenida, aproximadamente 23 datos por segundo, era insuficiente para sus necesidades.

Esta situación motivó la necesidad de desarrollar una solución propia que permitiera superar estas restricciones y ofrecer un mayor nivel de control sobre todos los aspectos del proceso de adquisición. En consecuencia, se planteó la creación de un sistema integral compuesto por tres entregables principales:

- Hardware de dispositivo adquisidor
- Firmware de dispositivo adquisidor
- Aplicación móvil

2.2. Requerimientos

En esta sección se definen y describen los requerimientos funcionales y no funcionales del proyecto "Desarrollo de un Lector y Analizador Avanzado de Sistemas CAN Bus en Vehículos".

2.3. Especificaciones funcionales

2.3.1 RF01: Lectura de datos del bus CAN de forma óptima

El dispositivo tiene que tener la posibilidad adquirir con una alta tasa de

velocidad de datos y leer las tramas del bus CAN

2.3.2 RF02: Decodificación y acondicionamiento de datos

El dispositivo tiene que tener la posibilidad de decodificar y acondicionar los datos a valores su respectivo formato correspondiente junto a sus unidades

2.3.3 RF03: Aplicación móvil interactiva con el dispositivo

Se debe elaborar una aplicación móvil interactiva con el dispositivo mediante comunicación inalámbrica

2.3.4 RF04: Interfaz personalizable por el usuario

Se debe de desarrollar una interfaz que permita, según las necesidades del usuario, filtrar y acondicionar el método de visualización de los datos obtenidos del módulo adquisidor

2.3.5 RF05: Base de datos de códigos estándar

Se deberá contar con una base de datos donde estén almacenados códigos estándar utilizados por el protocolo CAN Bus para permitir la visualización de los parámetros vehiculares dentro de la aplicación móvil.

2.4. Restricciones de diseño

1. El dispositivo de adquisición debe ser portátil.
2. La aplicación móvil debe funcionar para dispositivos con versiones de SO Android 13 en adelante.
3. Se deberá implementar las protecciones electrónicas pertinentes para evitar sobrecarga, corrientes inversas y cortocircuitos que afecten tanto al dispositivo de adquisición como a la electrónica del vehículo.
4. Se plantea como requerimiento de escalabilidad futura del proyecto, que el dispositivo de adquisición deberá contar con el hardware suficiente para manipular los tres canales de informaciones utilizados por el protocolo CAN

Bus, a pesar de que este proyecto implique la utilización de únicamente el bus estándar.

5. El dispositivo de adquisición debe pesar lo mínimo posible.

2.5. Diagrama en bloques

En la Figura 2 se detalla una versión básica del diagrama en bloques del dispositivo.

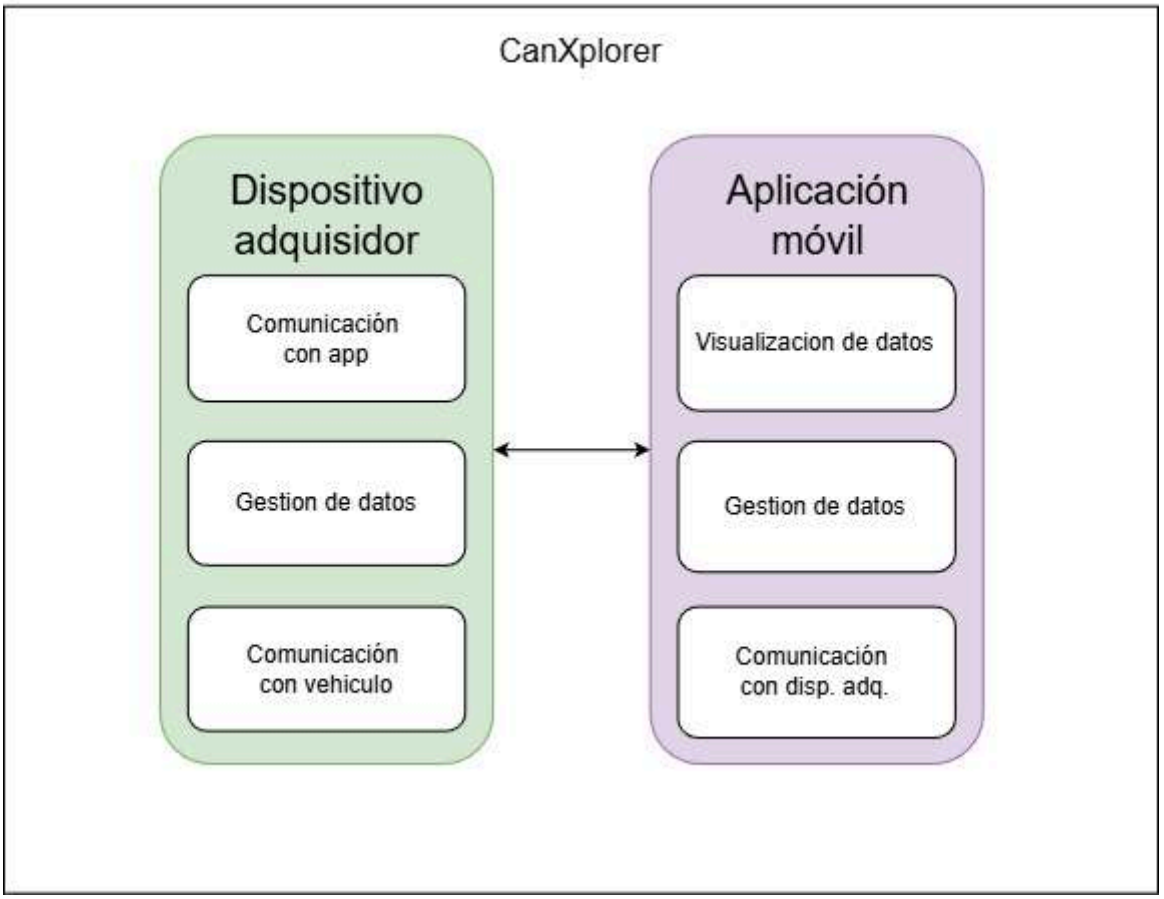


Figura 2. Diagrama en bloque simplificado del dispositivo.

3.6. Plan de trabajo

El proyecto se dividió en 5 etapas en las que se contempla la totalidad de las tareas necesarias para el inicio y finalización del mismo. Además, se propuso utilizar la plataforma Youtrack para llevar las tareas y la documentación diaria del avance en

el proyecto con el fin de tener medidas reales del tiempo empleado en este. En la Figura 3 se puede ver un diagrama de Gantt de la planificación propuesta.

Nombre de las Etapas	Inicio	Fin
Gestión y Documentación del Proyecto	04/03/2024	28/04/2025
1- Etapa Inicial	04/03/2024	31/12/2024
2- Capacitación	01/01/2025	01/03/2025
3- Análisis de Diseño de Hardware y Software	1/01/2025	17/02/2025
4- Construcción y Pruebas	17/02/2025	21/04/2025
5- Etapa Final	24/03/2025	28/04/2025

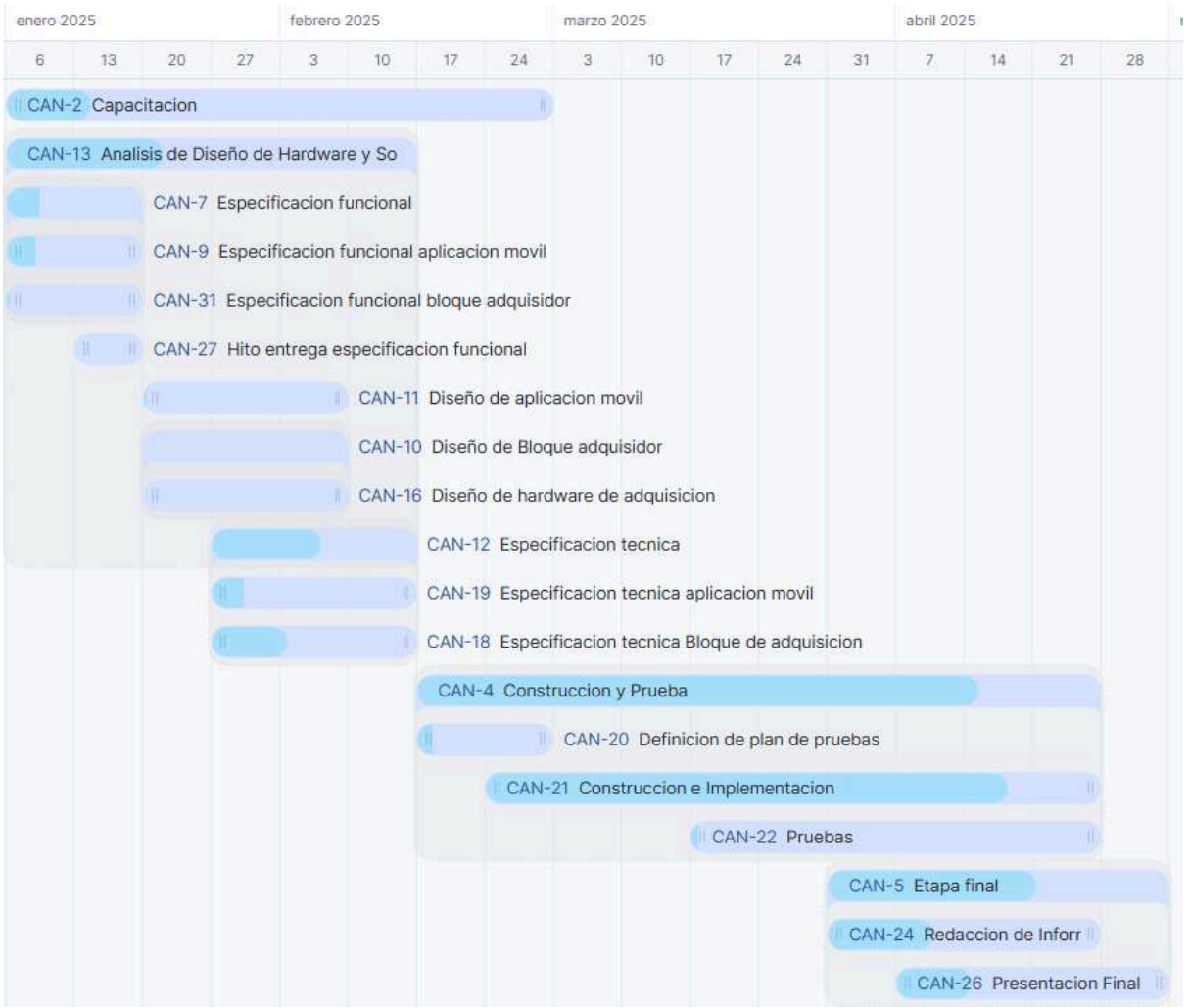


Figura 3.Diagrama de Gantt de la planificación propuesta.

La asignación de tareas se definió en función de las especialidades de cada integrante. Las actividades relacionadas principalmente con hardware fueron asignadas a Dante Cecchetti, mientras que aquellas vinculadas al desarrollo de software quedaron a cargo de Gonzalo Bauer. Esta distribución buscó optimizar el tiempo total del proyecto dividiendo el trabajo en dos áreas complementarias, garantizando al mismo tiempo una adecuada comunicación entre ambas partes y una correcta documentación de todo el proceso. Además, se estimó que cada integrante dedicaría 4 horas diarias al proyecto, de lunes a viernes, para cumplir con los plazos establecidos.

Capítulo 3:

Evolución del Diseño Electrónico del Sistema

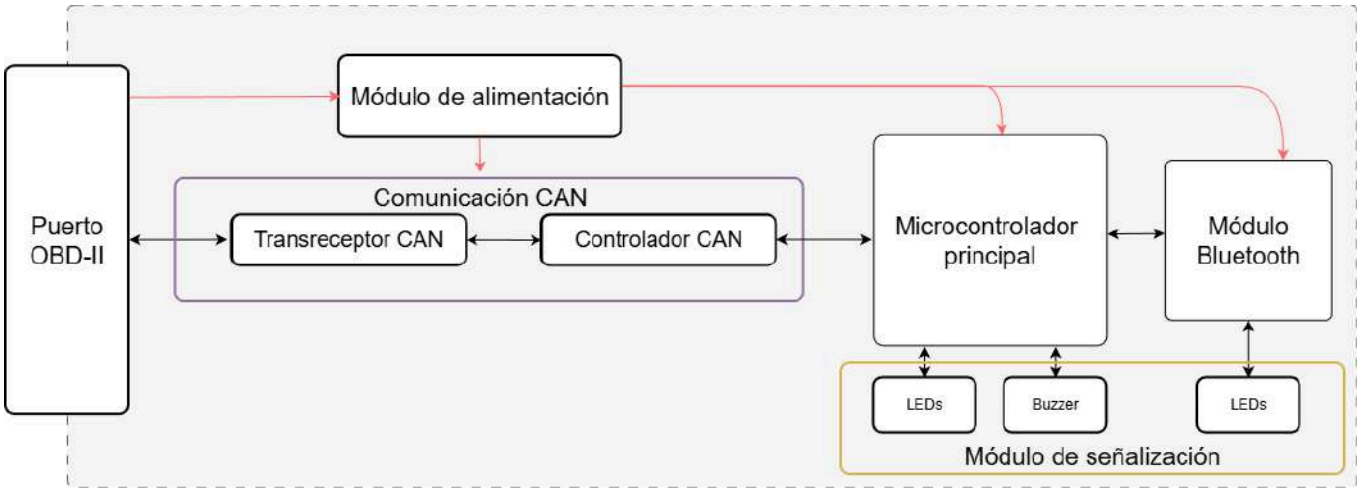


Diagrama 1. Diagrama general del diseño electrónico.

3.1. Análisis preliminar del sistema

Con el objetivo de establecer una base sólida para el desarrollo del sistema, se llevó a cabo un análisis preliminar orientado a identificar los bloques funcionales requeridos por el hardware del prototipo, en concordancia con las especificaciones funcionales del dispositivo. Esta etapa incluyó la revisión de documentación técnica, el estudio de componentes utilizados en proyectos similares y la evaluación de soluciones existentes.

Como resultado de este análisis, se determinó que el diseño debía contemplar cinco bloques principales:

- Bloque de comunicación con el vehículo: responsable de la transmisión y recepción de datos, permitiendo la adquisición de información desde el entorno vehicular.
- Bloque de procesamiento: compuesto por un microcontrolador central, encargado de interpretar, gestionar y procesar los datos capturados, además de coordinar el funcionamiento general del sistema.

- Bloque de señalización y alerta: encargado de informar visual o auditivamente el estado del sistema y posibles eventos relevantes mediante indicadores o actuadores.
- Bloque de comunicación inalámbrica: basado en un módulo Bluetooth, permite la transmisión de información procesada hacia un dispositivo móvil para su monitoreo y análisis en tiempo real.
- Bloque de regulación de tensión: encargado de convertir la tensión proveniente de la batería del vehículo a niveles compatibles con los circuitos electrónicos del sistema, asegurando un funcionamiento estable y seguro.

Adicionalmente, se definió que el enfoque del diseño estaría orientado al desarrollo de una plataforma versátil, permitiendo que funcionalidades no utilizadas en una primera etapa puedan ser habilitadas en futuras versiones. De este modo, se facilita la expansión y reconfiguración del sistema sin necesidad de rediseñar el hardware base. Además, durante la selección de componentes se priorizó que estos cumplieran con la normativa AEC-Q, un estándar automotriz que certifica su funcionamiento confiable en entornos vehiculares. Adicionalmente, se estableció como criterio que los capacitores posean un coeficiente de temperatura X7R, a fin de garantizar una mayor estabilidad de sus características eléctricas ante variaciones térmicas propias de ambientes de alta temperatura.

A continuación, se detallan los distintos bloques funcionales que conforman el sistema, abordando su propósito, los criterios considerados para su diseño, y las decisiones adoptadas en función de los requerimientos específicos del proyecto. Cada bloque se presenta por separado para facilitar la comprensión de su rol dentro del conjunto y su evolución a lo largo del desarrollo.

Finalmente, en el Anexo se encuentra documentado en detalle el diseño, cálculo y justificación técnica de cada uno de los bloques que integran el sistema.

3.2 Bloque de comunicación con el vehículo

El bloque de comunicación con el vehículo tiene como objetivo establecer el enlace físico y lógico entre el sistema desarrollado y la red de datos del entorno automotriz. Este módulo permite la transmisión y recepción de información a través

del conector OBD-II, posibilitando la adquisición de parámetros relevantes del vehículo mediante el protocolo CAN.

En las etapas iniciales del proyecto se evaluó la utilización de transceptores disponibles en el mercado local, con el fin de realizar pruebas experimentales sin necesidad de contar con el hardware final completamente desarrollado. La primera opción considerada fue el TJA1050, ampliamente empleado en entornos de prototipado basados en Arduino. Sin embargo, fue descartado debido a su tensión de operación de 5 V, lo que implicaba la incorporación de un segundo bloque de regulación de tensión, ya que se planeaba utilizar un microcontrolador con niveles lógicos de 3.3 V. Posteriormente se evaluó el transceptor SN65HVD230, cuya compatibilidad con el nivel de tensión de la placa lo convertía en una opción viable para trabajar con CAN 2.0. Sin embargo, al analizar el estado actual de la tecnología, se observó que desde 2015 el estándar CAN FD se ha consolidado como la evolución adoptada a nivel mundial. Dado que uno de los objetivos del proyecto es incorporar un hardware robusto y preparado para futuras ampliaciones, se decidió descartar dicha alternativa y adoptar el transceptor TCAN3413 [5], compatible tanto con CAN 2.0 como con CAN FD, garantizando así mayor flexibilidad y proyección a largo plazo.

A partir del análisis de su hoja de datos, se implementaron tres transceptores independientes con el fin de permitir la conexión simultánea a los tres buses CAN presentes en el entorno vehicular. No obstante, debido a que las líneas correspondientes a los buses CAN MS y CAN LS no se encuentran estandarizadas en el conector OBD-II y varían según el fabricante y modelo del vehículo, se optó por derivar sus pines a conectores tipo *header*, permitiendo su vinculación futura en función de la configuración específica del vehículo en cuestión.

3.3 Bloque de procesamiento:

El bloque de procesamiento tiene como finalidad centralizar la gestión de los datos adquiridos y coordinar el funcionamiento de los distintos módulos del sistema. Su función principal es interpretar las señales provenientes del vehículo, realizar el procesamiento correspondiente y gestionar la comunicación con los bloques restantes.

Al igual que en la elección de los transceptores, este bloque experimentó modificaciones durante el desarrollo. Inicialmente, se había optado por un microcontrolador STM32F413, cuya principal ventaja era la integración de tres controladores compatibles con el protocolo CAN 2.0, lo que permitía prescindir de controladores externos, optimizando así el espacio ocupado en la placa. Sin embargo, tras identificar la creciente adopción del protocolo CAN FD en la industria, se decidió sustituirlo por un STM32G473[2], el cual incorpora tres controladores compatibles con CAN FD, garantizando de este modo mayor escalabilidad y compatibilidad con estándares más recientes.

Este bloque también incluye los elementos auxiliares necesarios para que el microcontrolador funcione conforme a las especificaciones del sistema. Entre los aspectos más relevantes, se incorporó un cristal externo de 48 MHz. Si bien la hoja de datos del microcontrolador no lo exige para la configuración implementada, su inclusión mejora la exactitud temporal del sistema, ofreciendo una base sólida para futuras ampliaciones que puedan requerir temporización más estricta, como la incorporación de nuevos protocolos de comunicación.

Adicionalmente, se rediseñó el circuito de medición de la tensión de batería. Inicialmente, se había implementado un divisor resistivo conectado entre la línea de 12 V y una entrada ADC del microcontrolador, sin considerar adecuadamente las especificaciones de impedancia. Tras la revisión del director y codirector del proyecto, se reemplazaron las resistencias del divisor, pasando de valores en el orden de megohm a valores en kilohm, con el objetivo de minimizar el efecto de la impedancia de entrada del ADC. En esta etapa también se propuso incorporar un MOSFET[6] en la alimentación del divisor resistivo, con el fin de desconectarlo cuando el vehículo se encontrara apagado y así evitar un consumo innecesario. No obstante, durante la implementación se detectó un error en el diseño del circuito de control del MOSFET, lo que impidió su correcto funcionamiento. Por este motivo, se decidió eliminar dicho componente y conectar el divisor resistivo directamente a la línea de 12 V. La corrección de este inconveniente y la propuesta de un rediseño adecuado del bloque se detallan posteriormente en la sección “Problemas de hardware resueltos”.

Finalmente, y en línea con la filosofía del diseño orientado al desarrollo y expansión futura, todos los pines del microcontrolador que no fueron utilizados en la implementación inicial se derivaron a terminales tipo Dupont hembra. Esta decisión

permite su utilización en futuras modificaciones o ampliaciones del sistema sin necesidad de realizar alteraciones en el diseño original de la placa.

3.4 Bloque de señalización y alerta:

El bloque de señalización complementa el procesamiento de datos mediante la generación de alertas locales, facilitando la supervisión directa del estado del sistema sin depender exclusivamente de interfaces de comunicación remota. Su función principal es informar de manera inmediata y visual sobre el funcionamiento general del dispositivo y la ocurrencia de eventos relevantes.

Para ello, se implementó una serie de indicadores luminosos mediante luces LED de bajo consumo, destinados a representar distintas condiciones de operación: un LED verde para indicar el encendido del sistema, un LED rojo para alertar sobre errores de comunicación con el bus CAN, un LED azul para señalar la conexión exitosa vía Bluetooth con un dispositivo móvil, y un LED naranja para indicar la transferencia activa de datos hacia el mismo. Además, se incorporó un *buzzer* piezoeléctrico controlado por el microcontrolador, conforme a las recomendaciones del fabricante. En este caso, se optó por reemplazar el transistor bipolar sugerido en la hoja de datos por un MOSFET IRLML6344, seleccionado por sus menores pérdidas en conducción y su capacidad de activación con corrientes de control reducidas, mejorando así la eficiencia del circuito.

Finalmente, es importante destacar que el diseño de este bloque no recibió observaciones ni modificaciones durante el proceso de revisión del proyecto, manteniéndose sin alteraciones respecto a la propuesta inicial.

3.5 Bloque de comunicación inalámbrica

Este bloque se encarga de establecer el vínculo inalámbrico entre el sistema embarcado y un dispositivo externo, facilitando la supervisión remota mediante la transmisión continua de la información procesada.

Para la elección del módulo se consideraron como criterios principales el tamaño reducido del encapsulado y la compatibilidad con Bluetooth Low Energy (BLE), a fin de minimizar tanto el espacio ocupado en la PCB como el consumo

energético del dispositivo. La primera opción evaluada fue el DA14531MOD, que cumplía con estos requisitos; sin embargo, se optó finalmente por el ESP32-C3-MINI-1U[7], que además incorpora conectividad Wi-Fi, otorgando mayor flexibilidad de aplicación al sistema.

Este chip se comercializa en dos variantes de encapsulado: una con antena integrada y otra con conector para antena externa tipo SMD. Se seleccionó esta última opción para disponer de mayor flexibilidad en el posicionamiento del módulo sobre la PCB. No obstante, debido a una omisión en la revisión detallada del *datasheet*, no se advirtió que el conector de antena utilizado es de tipo MHF3, un formato de difícil acceso en el mercado argentino. Esta situación generó un retraso de 13 días, ya que, siguiendo la recomendación del equipo directivo del proyecto, se decidió posponer la validación de la placa hasta contar con la antena adecuada, evitando posibles daños en el conector por falta de adaptación de impedancia.

3.6 Bloque de regulación de tensión

El diseño de este bloque comenzó con la evaluación de diferentes estrategias para reducir y estabilizar la tensión de entrada del sistema. Si bien inicialmente se consideró el uso de un regulador lineal por su simplicidad, su baja eficiencia y elevada disipación térmica bajo condiciones típicas del entorno automotriz motivaron la elección de una fuente conmutada, más eficiente y robusta frente a variaciones en la tensión de la batería, aunque con mayor complejidad de implementación.

En una primera instancia se seleccionó un convertidor buck LM2734Z, capaz de suministrar hasta 1 A, valor superior al consumo máximo estimado del sistema, que alcanza los 750 mA bajo carga plena. Su capacidad de operar a altas frecuencias de conmutación también representaba una ventaja, ya que permite utilizar componentes pasivos de menor tamaño, optimizando así el espacio ocupado en la PCB.

No obstante, tras sucesivas consultas con docentes de la cátedra de Trabajo Final, así como con el director y codirector del proyecto, se decidió reemplazar el integrado. Entre las observaciones destacadas se señaló una sobredimensión innecesaria de la corriente de salida, dado que las condiciones de carga máxima no se presentan en el funcionamiento normal del sistema. Además, el diseño inicial del

layout (la ubicación real que van a tener los componentes sobre la placa), que disponía componentes en ambas caras de la placa para reducir el área ocupada, no contemplaba aspectos críticos vinculados a emisiones electromagnéticas, lo cual podría generar interferencias en bloques sensibles, como el de comunicación inalámbrica.

Por recomendación de uno de los docentes, se optó por utilizar un convertidor sincronico AP64060[4], el cual ademas de poder suministrar una corriente de 600mA y operar en alta frecuencia, su capacidad de ser sincrónico evita el uso de componentes electrónicos que estaban presentes en la anterior implementación logrando una mejor miniaturización del bloque en cuestión.

Para su implementación, se siguió la guía de diseño proporcionada por el fabricante, que establece los criterios y ecuaciones necesarias para dimensionar los componentes clave, como el inductor, el capacitor de salida y las resistencias de realimentación, entre otros. En una primera etapa, y tras verificar que cumplía con los requisitos de capacidad, se evaluó la posibilidad de reutilizar un capacitor cerámico de 22 µF proveniente de la implementación anterior. Sin embargo, el análisis de su curva de impedancia en función de la frecuencia reveló que, en el rango de operación del nuevo convertidor, el componente presentaba un comportamiento levemente inductivo, lo cual podía comprometer tanto la eficiencia como la estabilidad del sistema.

Con el objetivo de garantizar un rendimiento óptimo, se descartó su uso y se seleccionaron nuevos capacitores que mantuvieran un comportamiento capacitivo en la frecuencia de conmutación del AP64060. Finalmente, se optó por utilizar dos capacitores cerámicos de 10 µF modelo GRM21BR71A106KA73L de Murata en paralelo, solución que ofrece mejor respuesta en frecuencia y menor inductancia serie equivalente (ESL).

3.7 Problemas de Hardware resueltos

Durante la revisión del diseño de la placa, se detectó que el circuito destinado a habilitar o deshabilitar el ADC presentaba un funcionamiento incorrecto. El MOSFET empleado nunca alcanzaba el estado activo, debido a que no se cumplía la relación Vgs requerida para un MOSFET de canal N. Por esta razón, en la versión

actualmente implementada, se decidió retirar dicho MOSFET y conectar el divisor resistivo directamente a los 12 V de la batería.

Como solución alternativa, se propone un esquema que emplea un MOSFET de canal P junto con un transistor que actúa como driver, permitiendo controlar de manera correcta la activación y desactivación del ADC del microcontrolador.

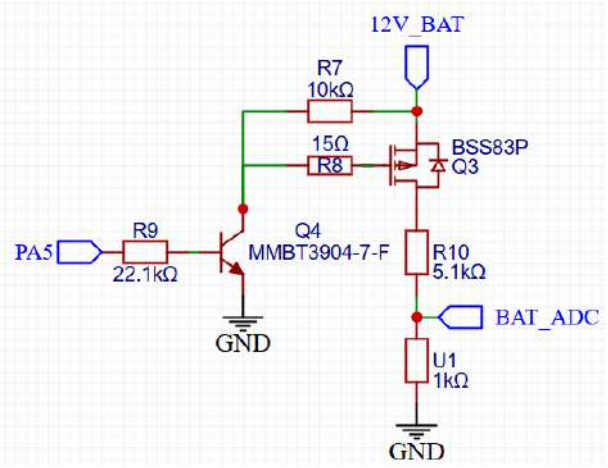


Figura 4. Esquemático del divisor resistivo para medir el voltaje de la batería corregido.

Por un descuido durante el diseño, la placa final no incluyó protecciones contra inversión de polaridad ni protecciones ESD en los pines de programación. Inicialmente, el microcontrolador ESP32-C3 Mini se programó utilizando los pines USB de programación en lugar de la UART, dado que esta era la configuración por defecto del dispositivo, lo que hacía especialmente necesario proteger dichos pines. Esta situación se resolvió posteriormente mediante el desarrollo de una placa externa que incorpora todas estas protecciones, incluyendo ESD en los pines de programación USB. El esquema de las protecciones implementadas se presenta en las siguientes figuras.

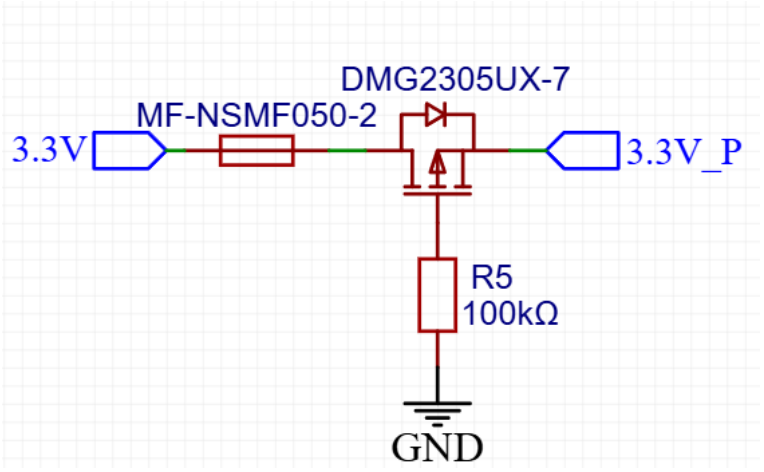


Figura 5. Esquemático de circuito de protección contra inversión de voltaje.

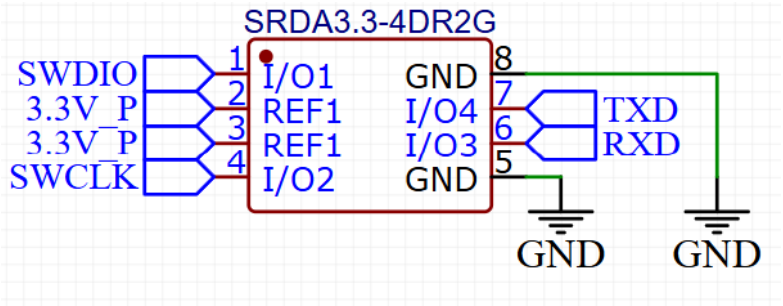


Figura 6. Esquemático de circuito de protección ESD para la programación via UART y SWD.

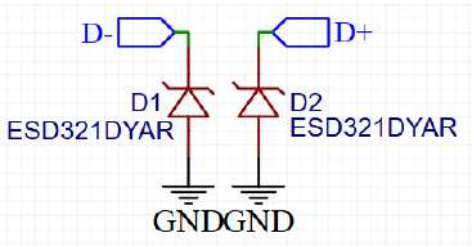


Figura 7. Esquemático de circuito de protección ESD para la programación via USB.

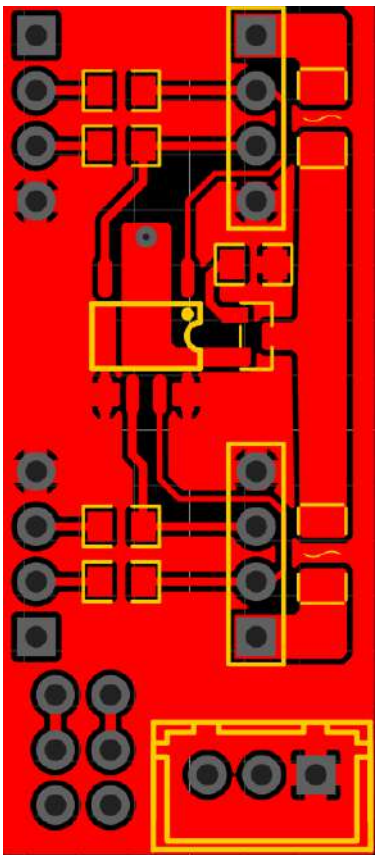


Figura 8. Layout superior de la placa externa con protecciones.

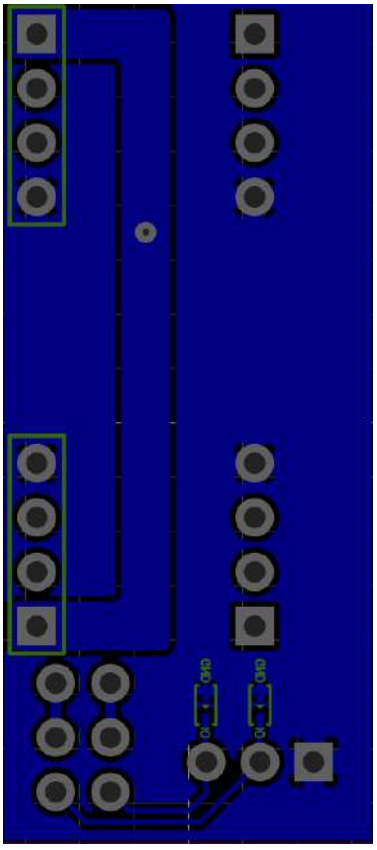


Figura 9. Layout inferior de la placa externa con protecciones..

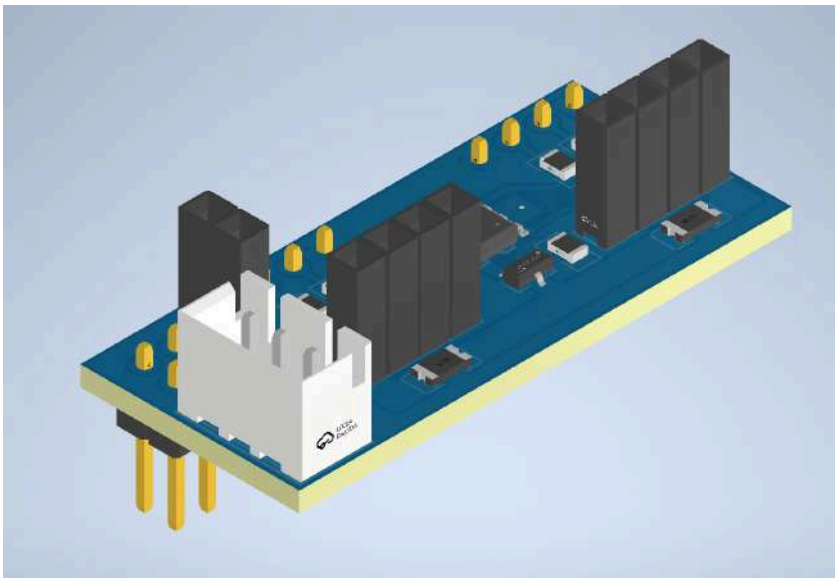


Figura 10. Montaje 3D de la placa externa (perspectiva lateral derecha).

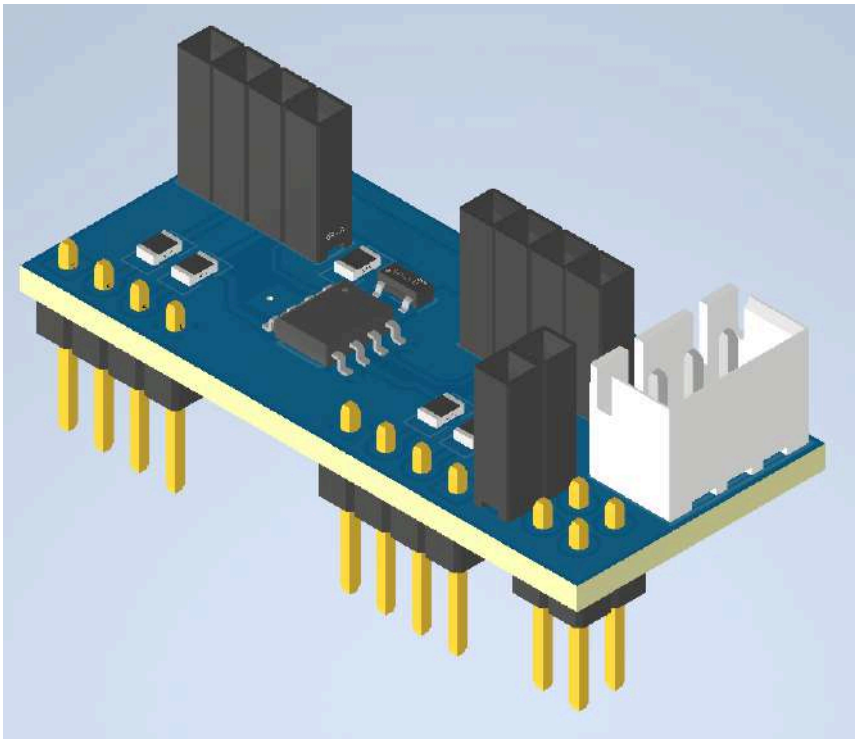


Figura 11. Montaje 3D de la placa externa (perspectiva lateral izquierda).

3.8 Diseño del Layout

3.8.1 Adaptador de 90º del conector OBD-II

El diseño de esta placa se fundamentó en las limitaciones identificadas en un desarrollo previo, particularmente relacionadas con la disposición de los pines de los conectores OBD-II disponibles en el mercado argentino (Figura 12).

Figura 12. Disposición de pines del conector OBD-II

En base a dicha configuración, se desarrolló un diseño inicial implementado en una arquitectura de doble capa (dos niveles de PCB), con el objetivo de lograr un diseño compacto y permitir la interconexión directa entre los pines del conector y las correspondientes entradas del circuito.

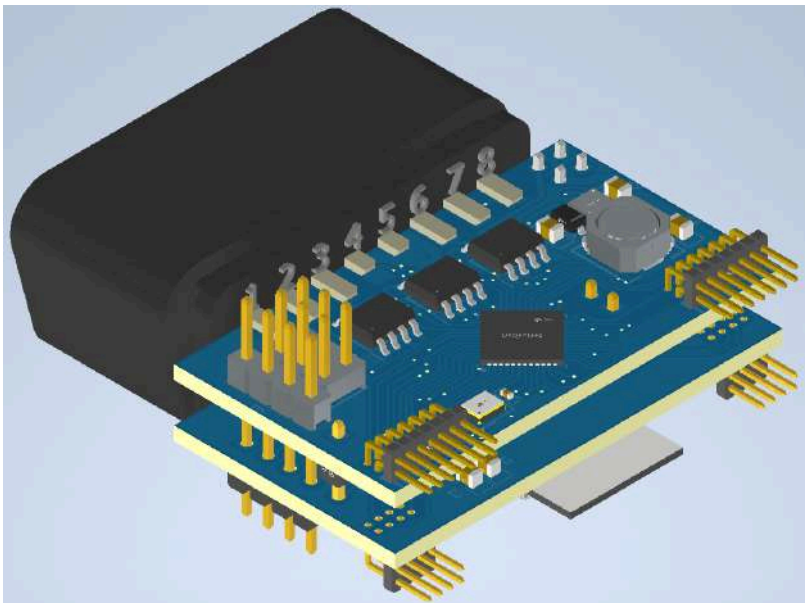


Figura 13. Vista superior del primer prototipo desarrollado.

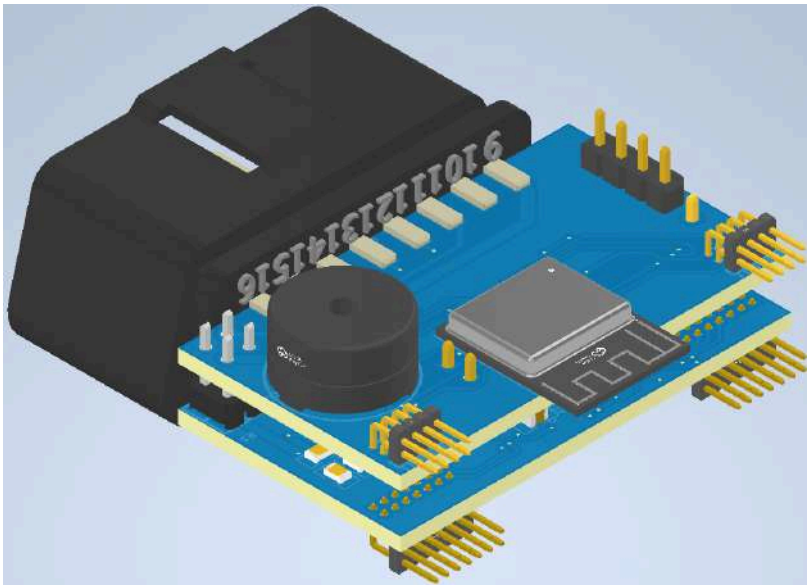


Figura 14. Vista inferior del primer prototipo desarrollado.

Tal como se observa en las figuras 13 y 14, ambos niveles estaban interconectados mediante pines tipo header, lo cual introdujo dificultades en la conexión del bus CAN. Estas complicaciones se originan en la naturaleza diferencial de la comunicación CAN, que exige una implementación cuidadosa en términos de diseño físico. En particular, resulta crítico mantener la simetría en la longitud de las pistas y garantizar su cercanía mutua. Diferencias en la longitud de las pistas pueden provocar desfases temporales entre las señales, generando errores en la recepción. Asimismo, la proximidad entre las pistas asegura que ambas estén expuestas a interferencias electromagnéticas similares, permitiendo así al transreceptor diferenciar eficazmente la señal útil del ruido. En consecuencia, se optó por diseñar un adaptador en ángulo de 90° que permitiera disponer todos los pines del conector en una única PCB (Figura 15). Esta configuración facilitó el ruteo de cada pista asociada a señales diferenciales, permitiendo aplicar criterios de diseño más rigurosos en cuanto a la longitud y proximidad entre pistas, tal como lo requiere la comunicación CAN.

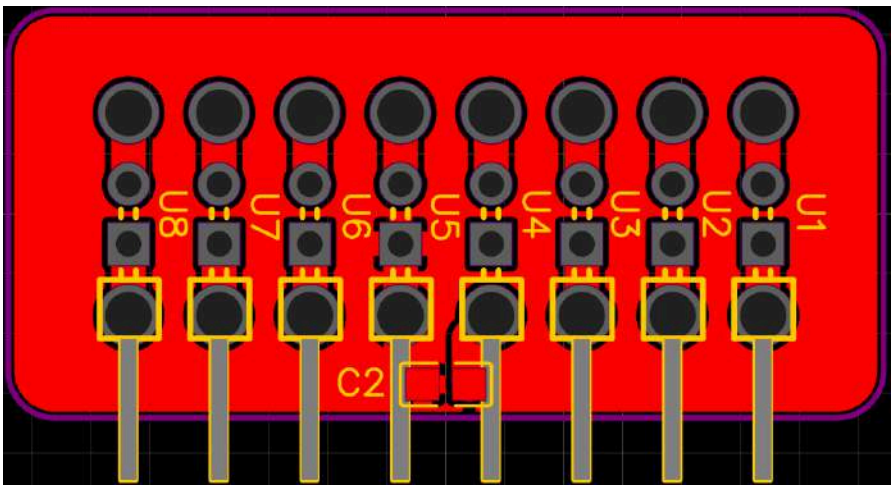


Figura 15. Layout de la placa adaptadora del conector OBD-II del prototipo final.

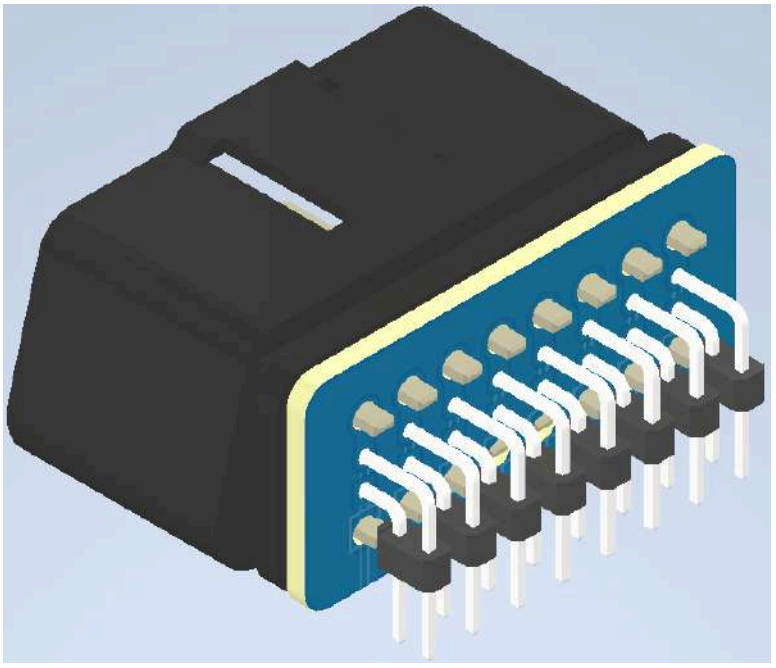


Figura 16. Montaje 3D de la placa desarrollada sobre el conector OBD-II del prototipo final.

3.8.2 Placa de desarrollo

Este diseño tiene como objetivo integrar la totalidad del sistema en una única PCB que facilite la manipulación del circuito ante eventuales correcciones, y que además cuente con puntos de medición para señales críticas. Asimismo, se contemplaron las consideraciones previamente mencionadas en relación con el

ruteo de señales diferenciales, aplicando criterios que aseguren su correcta transmisión. Adicionalmente, se implementaron planos de masa en ambas caras de la placa con el fin de minimizar el ruido en cada línea de datos. También se incorporaron pines tipo header para permitir la interconexión, previa evaluación de las señales presentes en cada terminal, de aquellos pines no estandarizados con los transceptores correspondientes a los buses CAN MS y CAN LS. Por otro lado, tal como se especificó en el diseño del microcontrolador y del módulo Bluetooth, aquellos pines que no se utilizan en la implementación fueron ruteados a terminales tipo Dupont hembra, con el objetivo de permitir su uso en futuras ampliaciones.

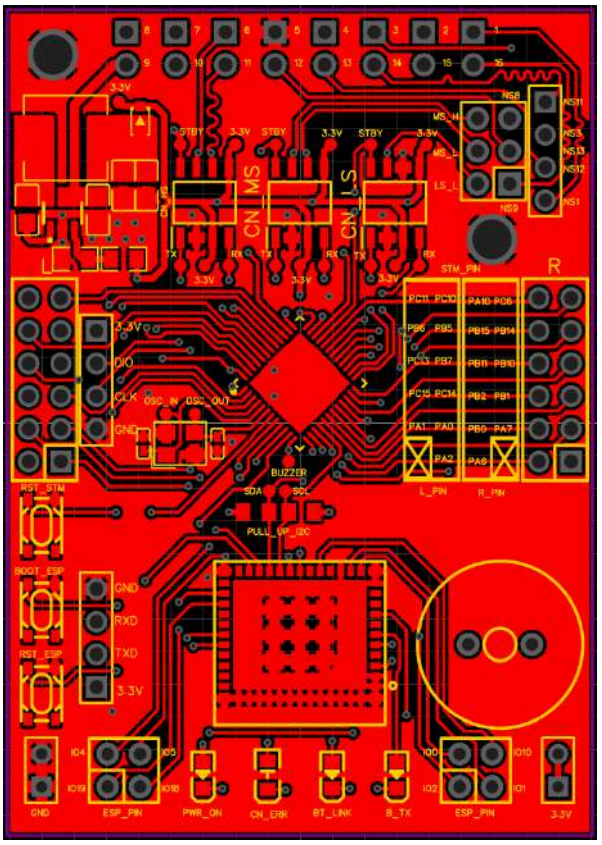


Figura 17. Layout superior de la placa de desarrollo del prototipo final.

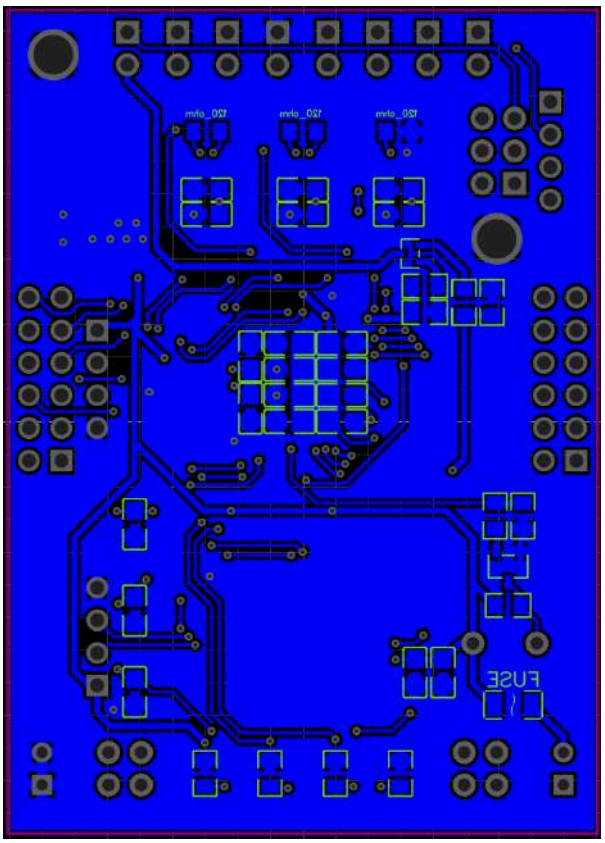


Figura 18. Layout inferior de la placa de desarrollo del prototipo final.

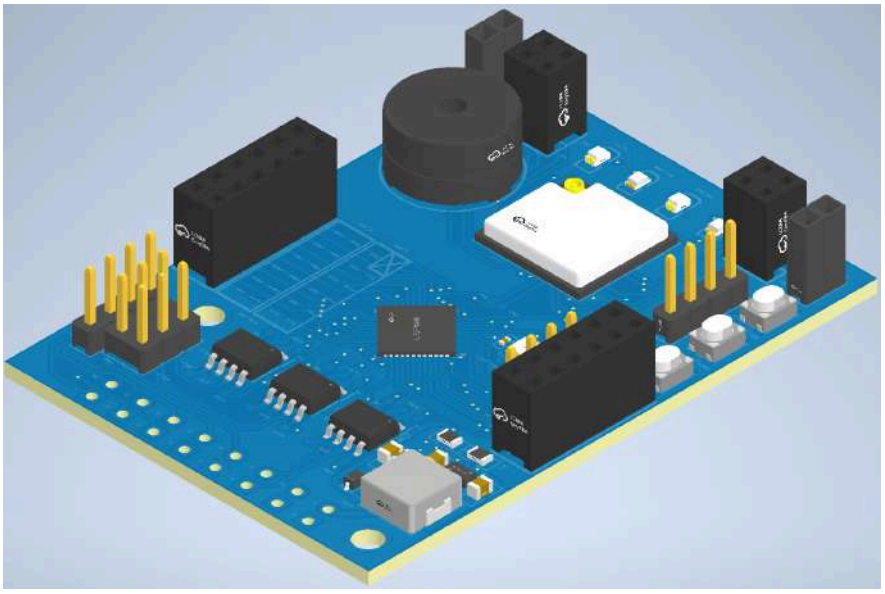


Figura 19. Montaje 3D de la placa de desarrollo del prototipo final.

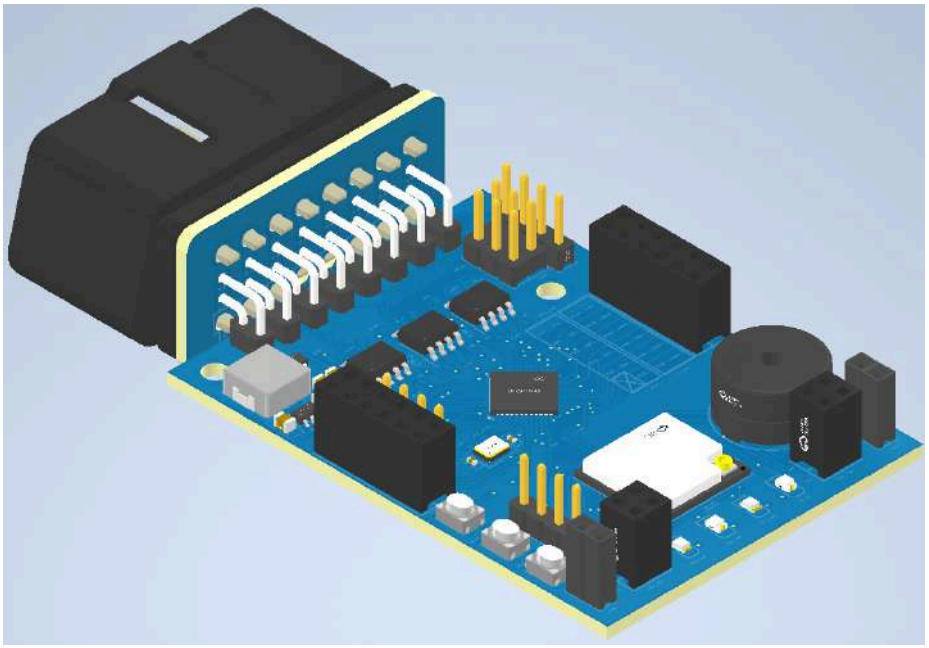


Figura 20. Ensamble 3D de la placa de desarrollo con el adaptador a 90º y el conector OBD-II del prototipo final.

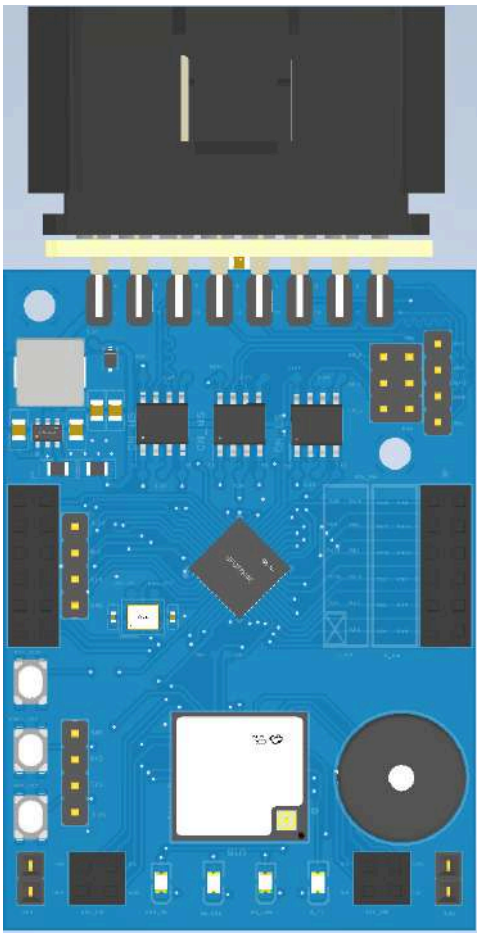


Figura 21. Vista superior de ensamble 3D de la placa de desarrollo con el adaptador a 90º y el conector OBD-II del prototipo final.

Capítulo 4:

Evolución del Diseño de Software

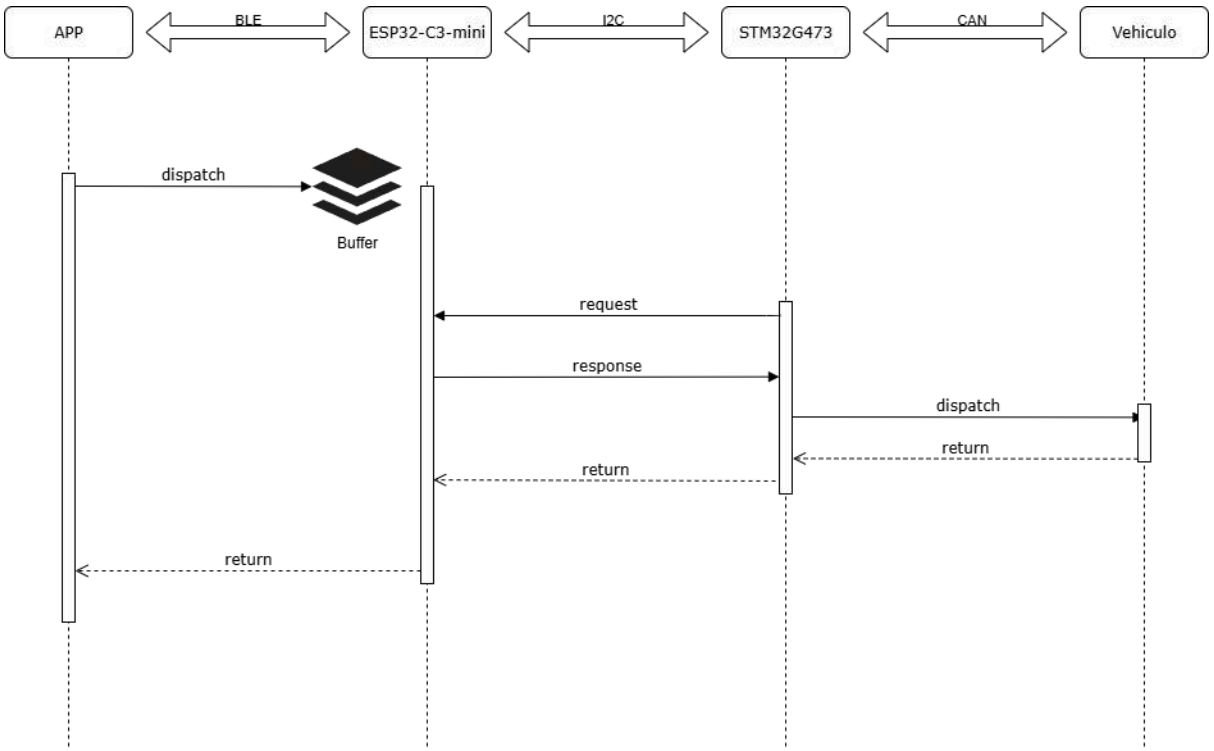


Figura 26. Diagrama de secuencia de la solución.

4.1. Análisis de la Arquitectura del Software

Previo iniciar con la codificación, se llevó a cabo una planificación estructurada de la arquitectura de software, con el objetivo de establecer un marco de desarrollo ordenado, escalable y alineado con los requerimientos funcionales del sistema. Esta fase fue fundamental para anticipar la interacción entre los distintos componentes, definir responsabilidades claras y minimizar ambigüedades en etapas posteriores.

El sistema completo se abordó desde una perspectiva modular, identificando desde el comienzo tres unidades de procesamiento con funciones bien diferenciadas: el microcontrolador STM32G473, el módulo ESP32-C3 Mini y la

aplicación móvil. En función de sus capacidades técnicas y de las exigencias del sistema, se distribuyeron las tareas entre estos componentes, priorizando la independencia funcional de cada módulo.

Con el objetivo de lograr una separación lógica de responsabilidades, se definieron los siguientes bloques funcionales:

- Bloque de adquisición CAN, exclusivo del STM32G473, encargado de la lectura y escritura sobre el bus del vehículo.
- Bloque de interfaz entre microcontroladores, compartido entre el STM32 y el ESP32, basado en el protocolo I²C.
- Bloque de señalización y control de estado, también compartido entre el microcontrolador y el módulo Bluetooth.
- Bloque de comunicación con la aplicación, gestionado entre el ESP32 y el dispositivo móvil a través de Bluetooth Low Energy (BLE).
- Bloque de visualización e interacción, correspondiente exclusivamente a la aplicación móvil.

Una vez definidos estos bloques, se estableció que la trama de datos utilizada sería la estandarizada para OBD-II con una longitud de 8 bytes. Esta estructura de paquete permite encapsular de forma estandarizada los datos del vehículo, facilitando su transmisión desde el STM32 hasta la aplicación móvil sin necesidad de reinterpretación en los nodos intermedios.

Luego, se tuvo que decidir qué datos iban a ser solicitados y mostrados al usuario por la aplicación móvil, dado que el protocolo OBD-II tiene 10 modos/servicios(Cuadro 1) descritos en el estándar SAE J1979[13] , y luego de hablar con mecánicos experimentados y usuarios de dispositivos similares al que se busca de desarrollar, se optó por tratar con aquellos modos de servicios que son más útiles o de uso más común para los usuarios , estos son los modos 1, 3 y 4, correspondientes a la muestra de datos actuales, la muestra de códigos de error de diagnóstico almacenados y a la limpieza de códigos de error de diagnóstico almacenados respectivamente.

Servicio/Modo (hexadecimal)	Descripción
01	Mostrar datos actuales
02	Mostrar datos de imagen fija
03	Mostrar códigos de diagnóstico de problemas almacenados
04	Borrar códigos de diagnóstico de problemas y valores almacenados
05	Resultados de la prueba, monitoreo del sensor de oxígeno (solo para CAN)
06	Resultados de la prueba, monitoreo de otros componentes/sistemas (solo para CAN)
07	Mostrar códigos de diagnóstico de problemas pendientes (detectados durante el ciclo de conducción actual o anterior)
08	Controlar el funcionamiento del componente/sistema integrado
09	Solicitar información del vehículo
0A	Códigos de diagnóstico de problemas (DTC) permanentes (DTC borrados)

Cuadro 1. Modos de servicio de OBD-II.

Con este enfoque, se organizó la planificación del software como una sucesión de capas, en la que cada una conoce únicamente sus entradas y salidas, sin depender del funcionamiento interno de las demás. A nivel práctico, esto favorece la implementación en paralelo y facilita la realización de pruebas unitarias. Gracias a esta planificación anticipada, se establecieron los criterios para el diseño de cada firmware y de la aplicación móvil, y se sentaron las bases para una arquitectura robusta, capaz de escalar e integrar nuevas funcionalidades en el futuro.

4.2 Desarrollo de la aplicación móvil

4.2.1 Desarrollo de interfaz gráfica

El desarrollo de la aplicación móvil se inició con la construcción de la interfaz gráfica, priorizando la claridad visual y la usabilidad. Para ello, se utilizó la plataforma Flutter[3], un framework de desarrollo multiplataforma que permitía construir interfaces a partir de componentes visuales reutilizables. Esta herramienta facilitó el diseño y permitió exportar directamente el layout de la aplicación en formato de código, acelerando significativamente las primeras fases del desarrollo. Cabe mencionar que, dado que el equipo de trabajo no contaba con una formación específica en diseño de interfaces ni en criterios avanzados de experiencia de usuario, se optó por una propuesta visual sencilla, basada en la intuición y en la experiencia como usuarios habituales de aplicaciones móviles, asegurando así una navegación clara y funcional desde las etapas iniciales.

De esta forma se logró una aplicación móvil con tres ventanas de navegación:

4.2.1.1 Panel de lecturas en tiempo real

Esta pantalla(Figura 22) muestra los datos del vehículo en tiempo real que sean deseados por el usuario, además que al presionar sobre el botón editar permite elegir qué parámetros se desean ver a partir de una lista de selección(Figura 23).

Figura 22. Pantalla de visualización de datos

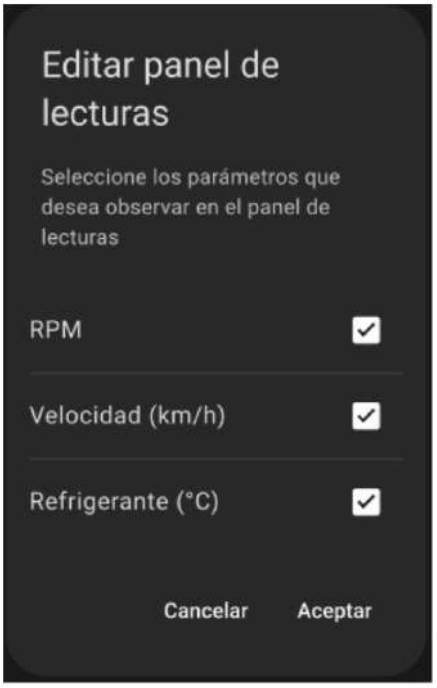


Figura 23. Pantalla de selección de valores a visualizar.

4.2.1.2 Panel de errores

Este permite ver los códigos de error de diagnóstico almacenados en el vehículo y solicitar su limpieza de memoria. Cuenta con un botón para solicitar los códigos y otro para solicitar limpiar la memoria de errores.

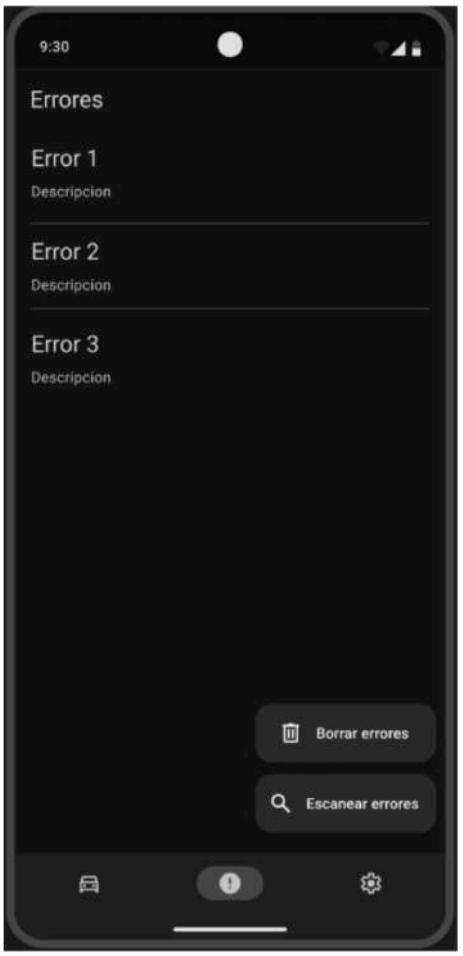


Figura 24. Pantalla de panel de errores

4.2.1.3 Panel de configuración

Permite mostrar si se está conectado al dispositivo adquisidor y permitir la conexión BLE en caso de ser necesario.

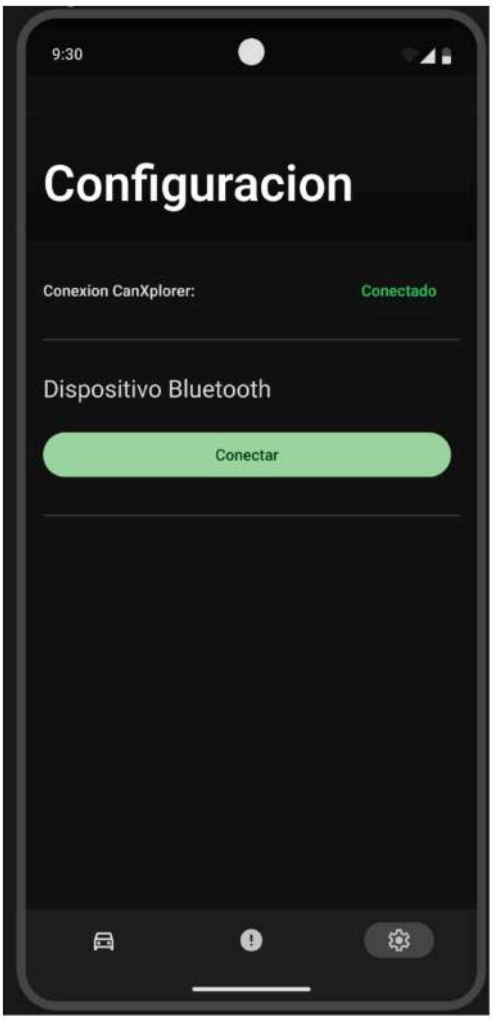


Figura 25. Pantalla de conectividad bluetooth.

4.2.2 Desarrollo de back-end

Una vez consolidado el diseño visual, el código generado en Flutter se importó a Android Studio, el entorno de desarrollo oficial de Google, donde se implementó la lógica funcional de la aplicación. Esta etapa incluyó la programación de las pantallas principales, la estructura de navegación interna y la organización del back-end, incorporando las rutinas de comunicación necesarias para el intercambio de datos con el dispositivo físico.

Uno de los principales desafíos enfrentados fue la integración del protocolo Bluetooth Low Energy (BLE), requerido por el módulo ESP32-C3 utilizado en el

hardware. Si bien inicialmente no se contaba con experiencia previa en el uso de BLE en Android, se llevó a cabo una etapa de investigación técnica que permitió comprender la estructura del protocolo y los métodos necesarios para su correcta implementación.

Durante las primeras pruebas de comunicación entre la aplicación móvil y el dispositivo, se detectaron ciertas limitaciones operativas. En particular, se observó que el protocolo BLE impone restricciones en el tamaño máximo de los paquetes de datos (20 bytes por trama) y en la frecuencia de transmisión (una trama cada 50 ms, aproximadamente 20 tramas por segundo). Si bien la limitación de tamaño no implicó un inconveniente crítico, ya que las tramas definidas por el sistema estaban por debajo de ese umbral, la frecuencia de transmisión representó una limitación para el rendimiento esperado.

Este inconveniente se resolvió desactivando el mecanismo de confirmación de recepción automática, implementado por defecto en el protocolo BLE. Si bien este mecanismo resulta útil en sistemas donde la fiabilidad es prioritaria, se consideró innecesario en aplicaciones orientadas a la transmisión continua de datos en tiempo real, como es el caso del sistema desarrollado. Tras realizar pruebas exhaustivas, se determinó que es posible transmitir tramas con un intervalo mínimo de 900 nanosegundos sin pérdida de datos, mejorando así la tasa efectiva de actualización.

Con la comunicación BLE estabilizada, se avanzó en la incorporación de las funcionalidades específicas del sistema, tales como la recepción de tramas en tiempo real, la solicitud de códigos de error almacenados (DTCs) y la limpieza de estos últimos en memoria. Estas funciones fueron desarrolladas sobre una arquitectura modular, lo que permite una posterior ampliación o modificación del sistema con relativa facilidad. Si bien se logró una velocidad alta de datos, es importante aclarar que la Organización Internacional de Normalización (ISO) en su norma 15765-4 en la que se pautan los requisitos de comunicación para sistemas de diagnóstico a bordo (OBD) en vehículos, se establece que el tiempo de espera entre la recepción de una respuesta y el envío de la próxima solicitud debe ser mayor a 50 ms y que el tiempo de espera de respuesta a una solicitud debe ser de al menos 200 ms, por lo tanto, esto se tuvo que respetar para la implementación de todas las funcionalidades mencionadas anteriormente.

4.3 Desarrollo del firmware del bloque adquisidor

El desarrollo del firmware se inició en paralelo con el diseño del hardware, pero dado que este último no se encontraba disponible en las primeras etapas, se implementó una plataforma de prueba compuesta por un ESP32-S3, un Arduino UNO y un módulo de comunicación CAN basado en el controlador MCP2515 en conjunto con un transceptor CAN TJA1050. Este conjunto permitió emular el comportamiento esperado del bloque adquisidor definitivo, garantizando así la continuidad del proceso de desarrollo. La elección de estas plataformas se basó en el hecho de que el firmware implementado en el microcontrolador ESP32-S3 podía posteriormente adaptarse al ESP32-C3-Mini sin modificaciones significativas, gracias a la compatibilidad técnica entre ambos dispositivos. De manera análoga, el uso del Arduino UNO permitió anticipar la lógica de control prevista para el microcontrolador STM32. Durante las primeras semanas, luego de distintos ensayos sin éxito con los módulos CAN, se optó por avanzar con el desarrollo de la aplicación móvil, comenzando con la creación del front-end y continuando posteriormente con el back-end. Finalmente, se trabajó sobre la comunicación inalámbrica mediante BLE, utilizando un ESP32-S3 para establecer la interacción entre el microcontrolador y la aplicación móvil. Tras recibir asesoramiento y guía por parte del cuerpo directivo del proyecto, se logró completar la configuración y puesta en marcha de los módulos CAN, lo cual permitió finalizar el desarrollo del firmware encargado de la comunicación entre el microcontrolador y el transceptor CAN.

Una vez se tuvo el hardware final fabricado se pasó a validar el funcionamiento integral del firmware utilizando un simulador de ECU desarrollado por el equipo sobre una placa Arduino UNO, se avanzó con las pruebas en un vehículo real. A través de sucesivas iteraciones, se logró optimizar el firmware para incrementar la tasa efectiva de adquisición de datos y, paralelamente, corregir diversas inconsistencias presentes en la visualización dentro de la aplicación móvil.
En cuanto a la estructura funcional del firmware, se distinguieron tres bloques principales de comunicación, los cuales fueron desarrollados de manera progresiva: en primer lugar, la comunicación BLE, seguida por la comunicación entre

microcontroladores mediante el protocolo I²C, y finalmente la comunicación a través del bus CAN.

4.3.1 Comunicación BLE

A la hora de implementarse, el protocolo BLE es diferente al Bluetooth clásico; este último está optimizado para conexiones continuas de alta tasa de datos, mientras que BLE está optimizado para comunicaciones intermitentes de baja energía y bajo volumen de datos, ideales para dispositivos IoT. Para lograr lo planteado anteriormente, BLE organiza la comunicación mediante una arquitectura basada en perfiles, servicios y características, donde cada elemento se identifica mediante un UUID. Un servicio representa un conjunto lógico de funcionalidades (por ejemplo, un servicio de sensor de temperatura), y dentro de él cada característica actúa como una unidad mínima de datos que puede leerse, escribirse o notificarse. Cada característica tiene propiedades específicas (como leer, escribir o notificar) y se comporta como una dirección o punto de acceso a una porción concreta de información.

Dado que BLE gestiona el intercambio de información a través de estas características y utiliza mecanismos de notificación que pueden generar tráfico asíncrono, el uso de una única característica para enviar y recibir datos puede provocar duplicaciones o estados inconsistentes, especialmente si se mezclan operaciones escribir y notificar sobre el mismo punto de datos. Luego de realizar pruebas utilizando una sola dirección para ambos sentidos de comunicación y observar mensajes duplicados debido a esta naturaleza asíncrona y orientada a eventos de BLE, se optó por utilizar dos characteristics separadas: una dedicada exclusivamente al envío y otra dedicada a la recepción de datos. Esta separación permite controlar el flujo de datos, evita colisiones lógicas y se ajusta al modelo de comunicación que se necesita para el proyecto.

4.3.2 Comunicación I²C entre microcontroladores

Dado que la comunicación con el vehículo se realizaría en el chip STM32 [9], se definió que dentro de la comunicación I²C este sea el microcontrolador responsable de asumir el rol de maestro. En este esquema, el microcontrolador ESP32 adopta el rol de nodo secundario dentro de la comunicación I²C,

incorporando un buffer dedicado al almacenamiento temporal de las tramas que deben ser entregadas al controlador STM32 a velocidad de 400kBit/s, optimizando así la gestión de datos entre los distintos módulos del sistema. Por su parte, el ESP32 se enfoca exclusivamente en la comunicación inalámbrica mediante BLE, recibiendo a través del bus I²C únicamente las tramas ya procesadas y empaquetadas por el STM32.

4.3.3 Comunicación CAN

Al centralizar la lógica de adquisición y filtrado de las tramas CAN en el STM32, que cuenta con periféricos FD-CAN [8] nativos diseñados para aplicaciones automotrices, se garantiza una captura más confiable y en tiempo real de los mensajes del vehículo. Es importante aclarar que para la implementación se siguió con lo pautado en la norma ISO 14765-4[1] en las que se plantean los requerimientos temporales para el envío de bit en la comunicación OBD-II para bus CAN de 500 kBit/s (Cuadro 2).

t_Q	t_{SJW}	t_{SEG1}	t_{SEG2}	**Nominal sample point position**
ns				%
100	300	1 500	400	80
125	375	1 500	375	81,25

Cuadro 2.Requerimientos temporales establecido por la norma ISO 14765-4.

Esta separación de responsabilidades reduce la carga de procesamiento en el ESP32, evita bloqueos en la comunicación BLE y permite mantener un flujo de datos consistentes entre la red CAN del vehículo, el bus I²C interno y la interfaz inalámbrica ofrecida al usuario final. A continuación(Figura 26) se puede ver un diagrama general de la comunicación implementada en la solución.

.

Capítulo 5:

Plan de pruebas y resultados

5.1. Ambiente de prueba

Se definen como ambientes de prueba las distintas versiones de software y hardware disponibles en el momento de la evaluación, considerando que estas pueden variar a lo largo del proceso de desarrollo.

5.2. Instrumentos de pruebas

Para llevar a cabo las etapas de validación y verificación del sistema desarrollado, se utilizaron los siguientes instrumentos y recursos:
Para el dispositivo adquisidor:

a. Vehículo
b. Simulador de OBD-II:
c. Programador STM
d. Programador ESP

Para la aplicación móvil:

a. Dispositivo móvil con Android 13 o superior

5.3. Tipos de pruebas

Para garantizar la calidad y funcionalidad del sistema desarrollado, se definieron distintas categorías de pruebas, las cuales se agrupan en:

- Pruebas unitarias
- Pruebas integrales
- Pruebas de homologación
- Pruebas de defectos
- Pruebas de fuerza

- Pruebas exploratorias

Las tres primeras se estructuran de forma secuencial, bajo un enfoque en cascada. En este esquema, la ejecución de pruebas integrales depende de la validación exitosa de las pruebas unitarias, y del mismo modo, las pruebas de homologación se llevan a cabo únicamente una vez superadas las pruebas integrales. Estas pruebas se realizan utilizando una estrategia de caja negra, lo cual implica que el tester no cuenta con conocimiento previo del funcionamiento interno del sistema. Para asegurar la objetividad de los resultados, se establece que los testers asignados a cada etapa sean distintos, o en su defecto, que adopten enfoques diferenciados para evitar sesgos en la evaluación.

Por otro lado, las pruebas de defectos, de fuerza y exploratorias no presentan dependencia con las anteriores y se ejecutan utilizando técnicas de caja blanca, enfocadas en detectar fallos internos y fortalecer la robustez del sistema. En particular, las pruebas exploratorias deberán ser realizadas por dos perfiles distintos: usuarios con conocimiento técnico del dispositivo y usuarios sin experiencia previa. Esta diversidad en el enfoque busca enriquecer la retroalimentación obtenida y ampliar la cobertura de posibles escenarios de uso.

5.4. Casos de prueba

Los casos de prueba definidos para la validación del sistema se encuentran detallados en el siguiente enlace.

5.5. Resultados

A lo largo del proceso de validación, cada ciclo de pruebas aportó información clave para refinar el diseño y optimizar el desempeño del sistema. En la primera etapa, centrada en el hardware, se detectaron problemas fundamentales: por un lado, la ausencia de resistencias pull-up en los botones BOOT y RESET de ambos microcontroladores generaba comportamientos erráticos durante la programación y la inicialización; por otro lado, se identificó un error en la selección del transceptor CAN. Este último inconveniente se originó a partir de una interpretación incorrecta de la hoja de datos, donde se confundió el valor real de

alimentación nominal del integrado, de 5V, con la referencia a un pin auxiliar capaz de operar a 3.3 V. Como consecuencia, se incorporó inicialmente un transceptor incompatible con los niveles lógicos del resto del sistema, imposibilitando el establecimiento de la comunicación CAN. La detección temprana de estas fallas permitió aplicar las correcciones correspondientes y mejorar de forma significativa la estabilidad y confiabilidad del dispositivo.

La integración con un simulador de ECU desarrollado sobre Arduino introdujo nuevos desafíos. La principal complicación fue la configuración errónea de la velocidad de comunicación, lo que no permitía el intercambio de tramas y evidenciaba la necesidad de revisar la sincronización entre dispositivos. Tras corregir estos parámetros y verificar el correcto funcionamiento en el entorno simulado, el proyecto avanzó hacia pruebas en un vehículo real, donde surgieron limitaciones más complejas.

Durante las evaluaciones en condiciones reales, la tasa de adquisición de datos resultó considerablemente menor a la prevista. Esto se vinculó tanto al incumplimiento de ciertos tiempos definidos por el estándar como a una configuración inadecuada de los controladores CAN del microcontrolador STM. Además, se identificaron problemas en la gestión de memoria asociada al protocolo y en la estrategia de transmisión entre microcontroladores y entre el dispositivo y la aplicación móvil. Estas fallas producían bloqueos inesperados y obligaron a replantear el uso de buffers, la estructura de los mensajes y la lógica general de comunicación.

Finalmente, al evaluar la aplicación móvil en su entorno de uso final, emergieron errores que no se manifestaban durante las pruebas controladas: cierres inesperados, fallas en la visualización y solicitud de PIDs, y ausencia de retroalimentación en algunos procesos. Esto derivó en una depuración integral de la interfaz y del flujo interno de la app para asegurar estabilidad y una experiencia de uso consistente.

En conjunto, todos estos hallazgos retroalimentaron el proceso de diseño, permitiendo fortalecer el hardware, estabilizar el software y consolidar una versión final del sistema con un desempeño confiable en escenarios reales.

Capítulo 6:

Conclusión

El desarrollo de este proyecto permitió alcanzar los objetivos planteados, logrando implementar un sistema capaz de adquirir, procesar y visualizar información vehicular a través del protocolo CAN con una arquitectura moderna y orientada a la eficiencia. El dispositivo resultante integra tecnologías actuales como Bluetooth Low Energy 5.0, CAN-FD y un diseño electrónico preparado para operar en entornos automotrices reales, lo que constituye una base sólida para futuras extensiones y mejoras.

La plataforma desarrollada no solo cumplió con los requisitos funcionales definidos inicialmente, sino que también demostró un rendimiento estable en escenarios reales de uso. La correcta interacción entre hardware, firmware y aplicación móvil permitió validar la robustez del sistema y su capacidad para manejar tasas de datos sostenidas sin comprometer la integridad de la comunicación.

Asimismo, la arquitectura del dispositivo fue concebida con una fuerte orientación a la escalabilidad, incorporando componentes y esquemas de diseño que habilitan la integración de nuevos protocolos, ampliación de funcionalidades y adaptación a diferentes entornos vehiculares sin necesidad de rediseños estructurales. Esto convierte al sistema en una plataforma versátil, apta no solo para diagnóstico vehicular, sino también para investigación, prototipado y desarrollo de herramientas avanzadas de análisis.

En conjunto, los resultados alcanzados evidencian la efectividad del enfoque adoptado y demuestran que la integración de tecnologías modernas, acompañada de un diseño modular y proyectado a futuro, constituye un camino sólido para el desarrollo de soluciones automotrices de alto rendimiento.

6.1 Aspectos Mejorables

6.1.1 Comunicación entre microcontroladores.

Si bien se logró una comunicación funcional y rápida entre los chips utilizando el protocolo I²C, se notó que dadas las prestaciones ofrecidas por los controladores elegidos se podrían utilizar otros protocolos.

6.1.2 Implementación de más protocolos.

Dado que este proyecto fue planteado para ser continuado y escalado, se le podrían implementar mayor cantidad de protocolos para el diagnóstico de vehículo, pensando sobre todo en vehículos más recientes.

6.1.3 Caracterización de pines no estándar del conector OBD-II.

Una línea de trabajo futura consiste en desarrollar una herramienta capaz de identificar y analizar los pines no estándar del conector OBD-II. Este enfoque permitiría reconocer los protocolos presentes en esos pines y determinar si conforman un bus de comunicación completo, ampliando el alcance del diagnóstico más allá de lo definido por el estándar.

6.1.4 Registro y análisis de tramas no estándar.

Otra mejora posible consiste en incorporar una función destinada a capturar y estudiar tramas propietarias que circulan por buses no estandarizados. Mediante este mecanismo sería posible registrar, clasificar y analizar mensajes que no se encuentran documentados públicamente, lo cual facilitaría tareas de ingeniería inversa y permitiría alcanzar una comprensión más profunda del comportamiento interno del vehículo.

6.1.5 Adquisición directa de variables vehiculares.

Otro camino de mejora consiste en implementar un método de adquisición directa sobre el bus estándar del vehículo, aplicando las mismas técnicas de registro y análisis utilizadas en buses no estandarizados. Este enfoque permitiría obtener las mismas variables que se consultan mediante OBD-II, pero sin necesidad de solicitar

a la ECU que genere tramas bajo ese formato, evitando así la sobrecarga propia del protocolo. Al capturar la información tal como circula de manera nativa en el bus, sería posible alcanzar tasas de actualización considerablemente superiores, junto con una lectura más precisa y continua de los parámetros vehiculares, sin depender del sistema de consultas tradicional.

6.2 Plan ejecutado y proyectado

El plan ejecutado se puede ver en las Figuras 27 y 28. Si bien en la planificación original planteada en el *Plan de proyecto* (Apéndice A.1.1), se propuso como finalización del mismo en abril del 2025, debido a diferentes problemas enfrentados durante el transcurso del mismo, los cuales se ven a continuación.

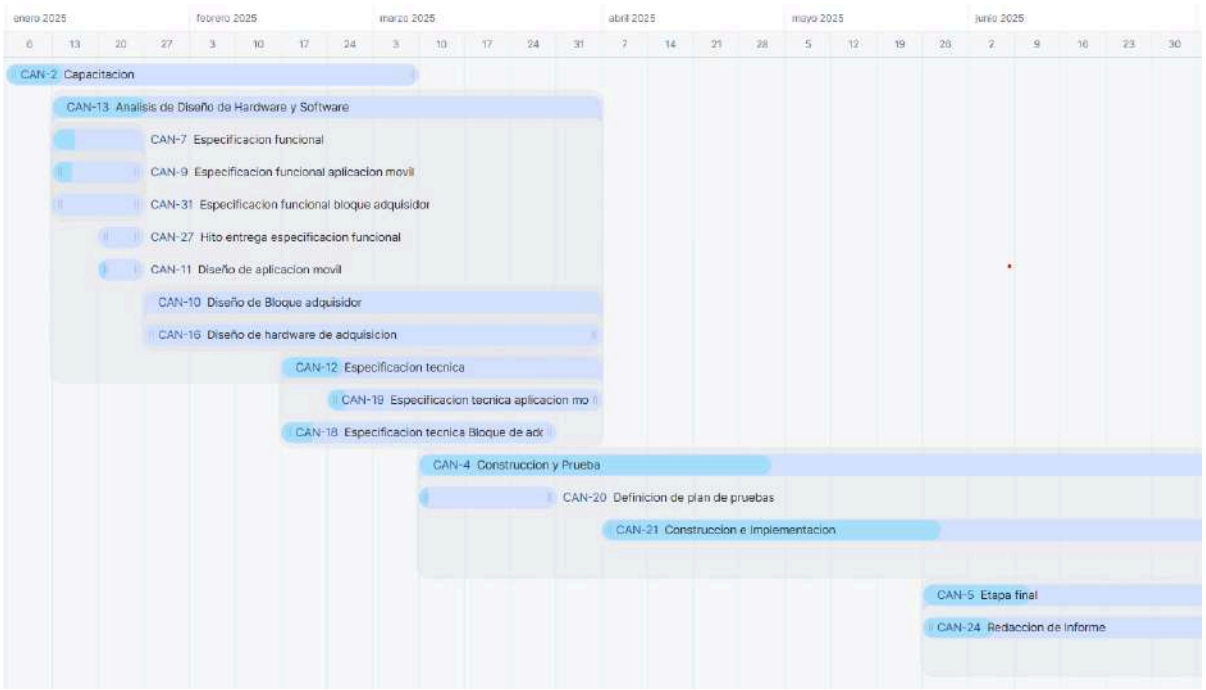


Figura 27. Plan ejecutado.

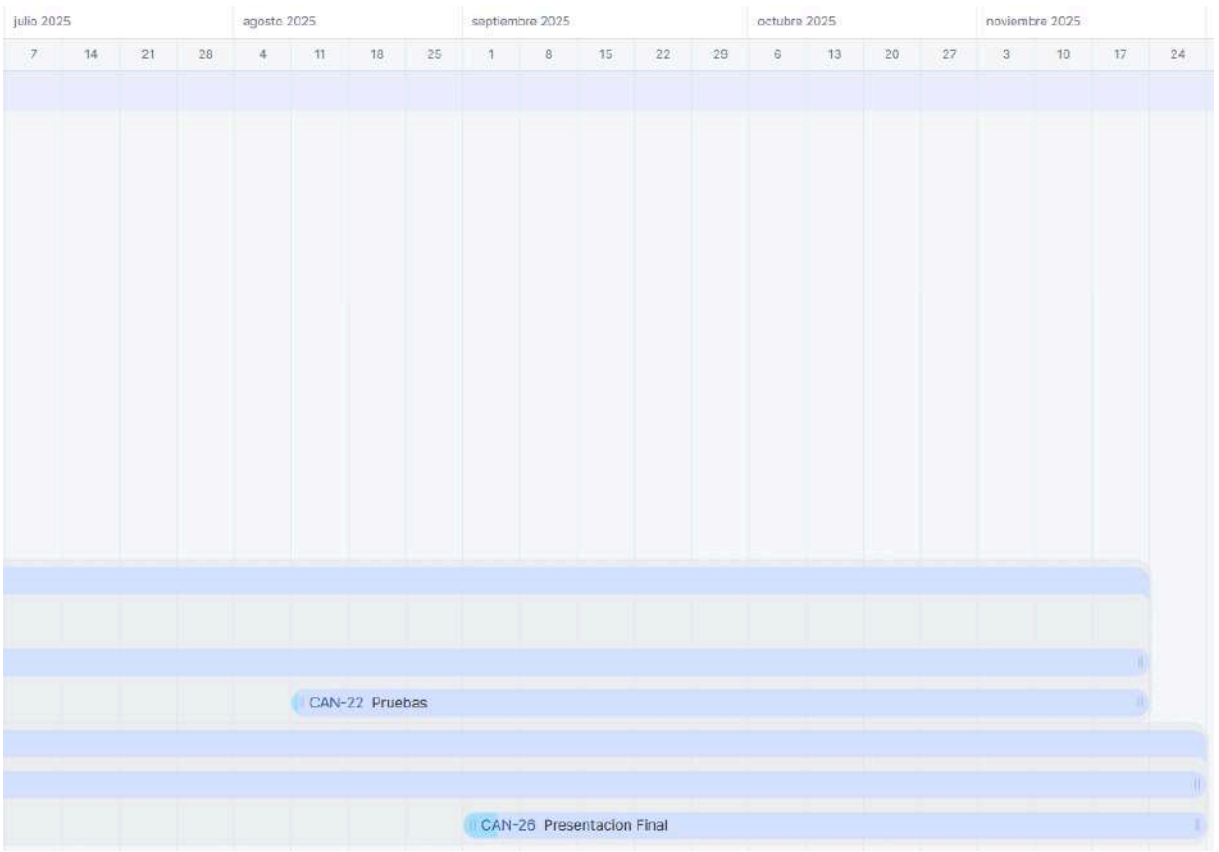


Figura 28. Plan ejecutado.

6.2.1 Módulos de comunicación CAN defectuosos en placa experimental

Como punto de partida del proyecto, se planteó establecer comunicación entre dos placas experimentales mediante el protocolo CAN, con el propósito de comprender su funcionamiento y características fundamentales. Para ello se emplearon módulos comerciales basados en el controlador MCP2515 junto con el transceptor TJA1050.

Durante esta etapa inicial, y tras verificar los módulos utilizando instrumental electrónico, se detectó que uno de ellos presentaba fallas: el controlador MCP2515 no respondía correctamente. Identificar este inconveniente y gestionar la adquisición de un nuevo módulo implicó un retraso aproximado de tres semanas en el avance previsto del proyecto.

6.2.2 Dificultad para conseguir antena SMD en Argentina.

Una vez fabricada la placa del dispositivo adquisidor, surgió un inconveniente vinculado al módulo Bluetooth: el conector de antena requerido no se encontraba disponible en el mercado local. Esta situación obligó a gestionar la compra del componente en el exterior, generando un retraso adicional de aproximadamente una semana en el cronograma de trabajo.

6.2.3 Elección incorrecta de transceptores en la placa implementada

Durante la implementación de la comunicación CAN entre el dispositivo adquisidor y el simulador de ECU, y mediante el uso de un osciloscopio, se detectó que el transceptor CAN integrado en la placa no estaba transmitiendo tramas al bus. Tras una revisión exhaustiva de la configuración del microcontrolador STM32 y del propio transceptor, se identificó un error de diseño: se había seleccionado un integrado de 5 V para una placa cuya electrónica operaba íntegramente a 3.3 V.

Confirmada la causa, se reemplazó el transceptor por un modelo compatible, lo cual implicó un retraso adicional de aproximadamente una semana en el desarrollo general del proyecto.

6.2.4 Implementación comunicación Bluetooth Low Energy

Dada la poca documentación encontrada acerca de implementaciones de este protocolo, su incorporación correcta llevó un tiempo considerablemente mayor al esperado.

6.2.5 Configuración controlador CAN

Dado el problema inicial de haber seleccionado mal el transreceptor CAN, se generaron problemas al probar la configuración óptima para el controlador encargado de dicha comunicación.

6.2.6 Llamados bloqueantes en el firmware

Al momento de probar la implementación del firmware en un simulador de ECU vehicular se pudo lograr fluidez a la hora de mostrar datos, pero cuando se

llevaba a un entorno de prueba real, es decir, en un vehículo, comenzaron a aparecer bloqueos en la comunicación entre el STM32G473 (maestro I²C) y el ESP32-C3 (esclavo I²C).

El origen del problema estaba en que el ESP32 ejecutaba operaciones lentas dentro de las rutinas de interrupción asociadas tanto al bus I²C como a eventos BLE.

Una Interrupt Service Routine (ISR) es una rutina que se ejecuta automáticamente cuando ocurre un evento de hardware, como la llegada de datos por I²C. Estas rutinas deben ser muy breves y eficientes, ya que mientras están ejecutándose, el microcontrolador no puede atender otras tareas críticas. Cuando dentro de una ISR se incluyen operaciones pesadas, como delay(), Serial.print(), uso de objetos String, lógica compleja o incluso el envío de notificaciones BLE, la interrupción se prolonga demasiado.

En esta implementación, dichas operaciones dentro de las ISR del ESP32 impedían que el dispositivo respondiera a tiempo al maestro I²C, dejando las líneas SDA o SCL en estado LOW y bloqueando físicamente el bus. En consecuencia, el STM32 quedaba esperando una transferencia que nunca se completaba, entrando en un estado de bloqueo del periférico I²C. Como el bus quedaba ocupado, el sistema solo podía recuperarse reiniciando ambos micros.

La solución consistió en eliminar toda operación lenta dentro de las interrupciones del ESP32, limitando las ISR a acciones rápidas y atómicas, y trasladando la lógica pesada al ciclo principal del programa. Esto evitó que el esclavo retenga el bus y permitió que la comunicación entre ambos microcontroladores funcione de manera estable tanto en simulación como en un entorno vehicular real.

6.2.7 Análisis de bitácora

Para lograr una documentación detallada del progreso del proyecto, se planteó el uso de bitácoras diarias para evidenciar las tareas realizadas, cuánto tiempo demandaron, quién las hizo, hacer observaciones, entre otras cosas.

Las tareas realizadas fueron clasificadas según su tipo, dividiendo estas en 5 categorías.

- Investigación: hace referencia al tiempo empleado para investigar y adquirir los conocimientos necesarios para el desarrollo del proyecto.
- Documentación: referencia al tiempo empleados para la documentación del proyecto
- Pruebas: horas invertidas en la realización de pruebas
- Implementación: tiempo usado para la construcción e implementación del proyecto
- Diseño: Diseño del hardware y firmware del bloque adquisidor y aplicación móvil.

En la Figura 29 se puede ver en detalle las horas dedicadas para cada etapa. La etapa de investigación, con 97 horas, muestra un esfuerzo moderado y adecuado para comprender las tecnologías y requisitos necesarios antes de avanzar, especialmente en un proyecto de complejidad media o alta como es el caso. La documentación, con 219 horas, representa una porción significativa del total, lo cual sugiere un trabajo detallado a la hora de plasmar las especificaciones y diseños desarrollados. Por su parte, el diseño acumuló 184.05 horas, un valor elevado que indica un fuerte enfoque en la arquitectura, la estructura del sistema y la definición de interfaces, lo cual se puede ver reflejado en el poco tiempo utilizado para prueba, ya que a pesar de los problemas atravesados se realizaron pocas iteraciones de las mismas. La fase de implementación, con 476 horas, es la que concentró la mayor inversión de tiempo, lo que resulta esperable dada la naturaleza práctica del desarrollo.

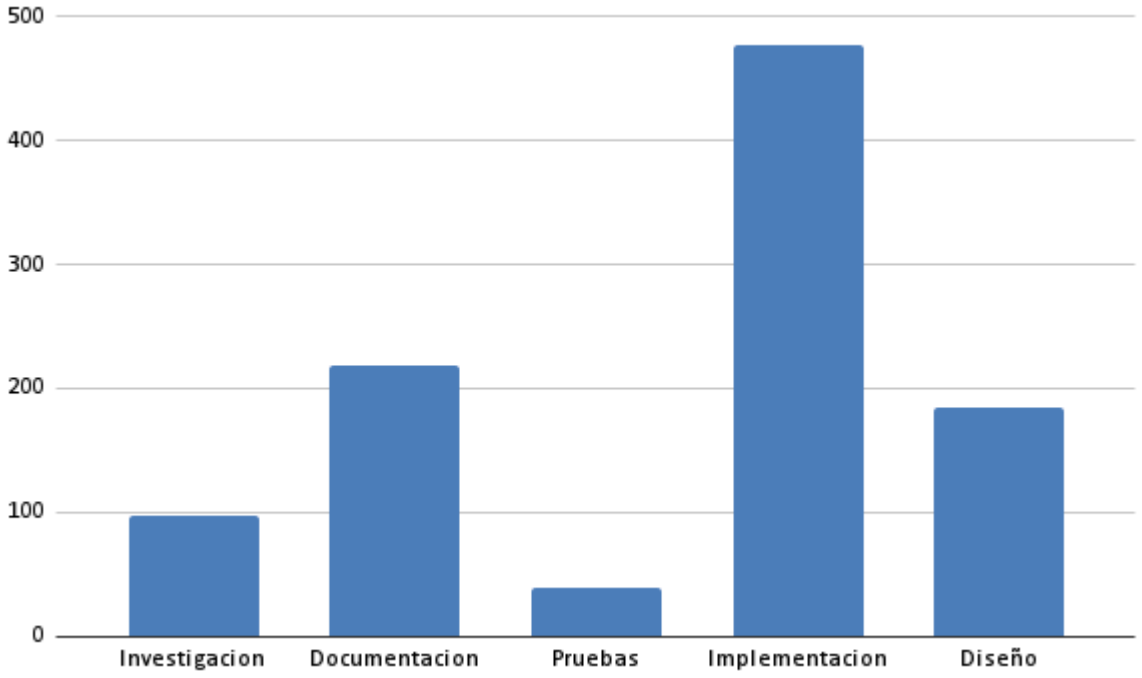


Figura 29. Detalle las horas dedicadas para cada etapa

Por otro lado, al analizar las horas de dedicación diaria al proyecto, representadas en la Figura 30, se observa que tampoco se cumplió con las 4 horas diarias planificadas. A esto se suma que algunos integrantes debieron interrumpir su participación debido a viajes y períodos de estudio para exámenes universitarios. Estos factores, en conjunto, contribuyeron a extender la duración total del proyecto.

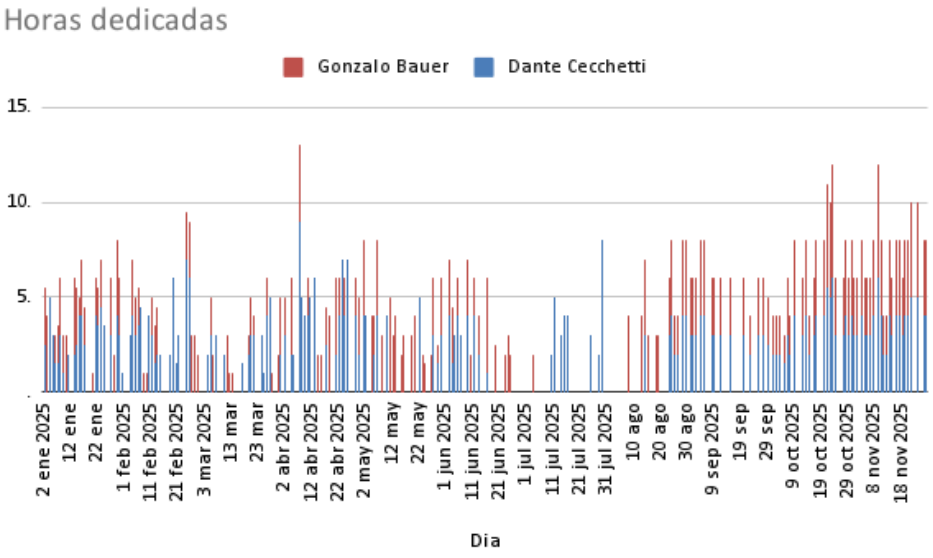


Figura 30. Horas de dedicación diaria al proyecto.

6.3 Aprendizajes

A lo largo del desarrollo del proyecto Can Xplorer, el proceso completo, desde la concepción inicial hasta la validación final del sistema, nos permitió adquirir conocimientos técnicos y metodológicos que difícilmente podrían haberse obtenido únicamente desde la teoría. Cada etapa sumó aprendizajes valiosos que fortalecieron nuestras habilidades tanto en el ámbito del diseño electrónico como en el desarrollo de software embebido y aplicaciones móviles.

En primer lugar, comprendimos en profundidad el funcionamiento del protocolo CAN y el estándar OBD-II, no solo desde su marco teórico, sino también desde los desafíos reales que implica su implementación. El estudio de normas como ISO 11898 y ISO 15765-4 nos permitió internalizar los requisitos temporales y eléctricos que condicionan la comunicación vehicular.

El proceso de diseño del hardware nos permitió comprender la relevancia que tiene la selección adecuada de componentes y la atención a detalles que, si bien suelen pasar desapercibidos en la teoría, definen la calidad de la implementación final. Aprendimos a evaluar especificaciones más allá de los valores nominales, considerando parámetros como la impedancia de entrada del ADC, el comportamiento real de los capacitores según su curva de impedancia y la elección de componentes compatibles con estándares actuales y futuros. Esta etapa también evidenció que los errores de diseño, como la falta de pull-ups, no solo son inevitables, sino fundamentales para el aprendizaje: cada fallo detectado durante las pruebas nos obligó a revisar supuestos, reformular soluciones y perfeccionar el diseño hasta lograr una implementación robusta y confiable.

En lo referente al software, aprendimos a planificar una arquitectura modular, escalable y distribuida, donde cada microcontrolador cumple un rol específico. Esto nos permitió valorar la importancia de separar responsabilidades entre adquisición, procesamiento y transmisión. La implementación de comunicación BLE nos mostró las limitaciones reales del protocolo, sus mecanismos de confirmación y cómo optimizarlo para transmisión en tiempo real sin comprometer estabilidad. También adquirimos experiencia en el diseño de aplicaciones móviles orientadas a la interacción con hardware, enfrentando desafíos como la gestión de paquetes, la visualización en tiempo real y la detección de fallos en condiciones reales de uso.

Uno de los aprendizajes más significativos surgió durante la fase de pruebas: entendimos que un sistema puede funcionar correctamente en un entorno controlado y aun así fallar en un vehículo real. Esto reforzó la importancia de diseñar ciclos iterativos de verificación, validar constantemente los supuestos teóricos y documentar detalladamente cada ajuste realizado.

Finalmente, es importante destacar que este trabajo, al haber sido desarrollado por dos integrantes provenientes de distintas ramas de la ingeniería (Ingeniería Electrónica e Ingeniería en Computación), resultó más dinámico y eficiente. El intercambio constante de ideas, el feedback mutuo y la colaboración permanente permitieron superar obstáculos con mayor rapidez. La diversidad de formaciones aportó enfoques complementarios para el análisis y la resolución de problemas, dando lugar a soluciones que difícilmente habrían surgido de manera individual. Asimismo, disponer de dos áreas de especialización permitió abordar en paralelo distintas etapas del proyecto, como la construcción del hardware final y el desarrollo del software definitivo, optimizando tiempos y mejorando la organización general del trabajo.

Bibliografía

[1] URL: https://www.iso.org/standard/67245.html

[2] URL: https://www.st.com/resource/en/datasheet/stm32g473cb.pdf

[3] URL: https://www.flutterflow.io/product

[4] URL: https://www.diodes.com/datasheet/download/AP64060_AP64062.pdf

[5] URL: https://www.ti.com/lit/ds/symlink/tcan3414.pdf?ts=1764560068135&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fes-mx%252FTCAN3414

[6] URL: https://www.infineon.com/assets/row/public/documents/24/49/infineon-irlml6344-datasheet-en.pdf?fileId=5546d462533600a4015356689c44262c

[7] URL: https://documentation.espressif.com/esp32-c3-mini-1_datasheet_en.pdf

[8] URL: https://www.st.com/resource/en/application_note/an5348-introduction-to-fdcan-peripherals-for-stm32-mcus-stmicroelectronics.pdf

[9] URL: https://www.st.com/resource/en/product_training/STM32G0-Peripheral-Inter-Integrated-Circuit-I2C.pdf

[10] URL: https://www.can-cia.org/

[11] URL: https://www.sae.org/standards/j1962_201509-diagnostic-connector

[12] URL: https://www.iso.org/standard/41284.html

[13] URL: https://saemobilus.sae.org/standards/j1979_201702-e-e-diagnostic-test-modes

Apéndice

A.1. Documentos anexos

A.1.1. Plan de proyecto

Plan de proyecto-Bauer-Cecchetti

A.1.1. Especificación de requerimientos

Especificacion de requerimientos-Bauer-Cecchetti

A.1.2. Especificaciones funcionales

Especificacion funcional Bauer-Cecchetti

A.1.3. Especificaciones técnicas

Especificacion tecnica Bauer-Cecchetti

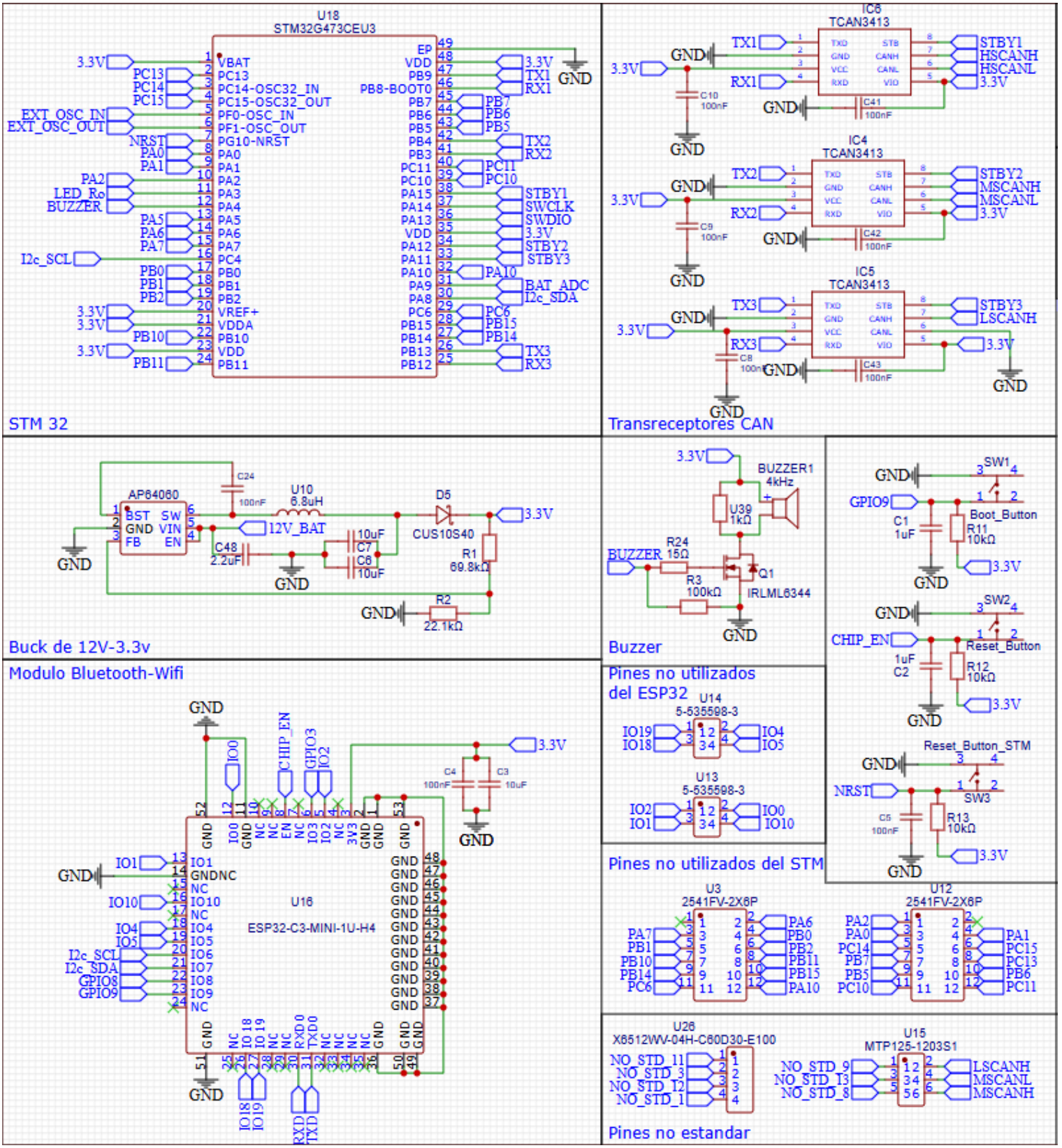
A.1.4. Plan de pruebas

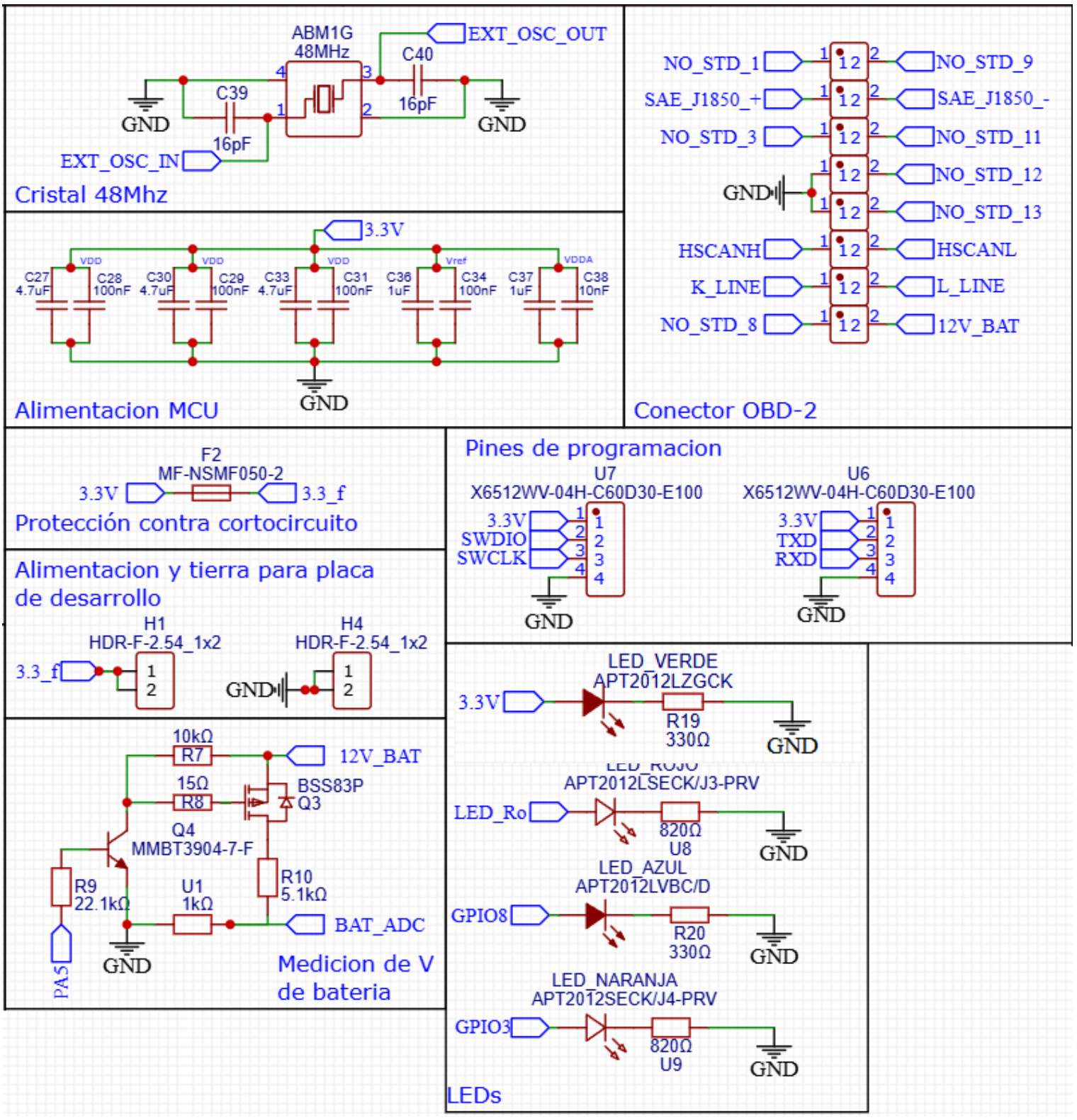
Plan de Pruebas-Bauer-Cecchetti

A.1.5. Implementación del firmware y la aplicación móvil

La implementación del firmware y la aplicación móvil se pueden ver al ingresar al siguiente link.

A.1.6. Esquemático del hardware principal




ABM1G
48MHz
EXT_OSC_OUT
C40
16pF
C39
16pF
GND
GND
EXT_OSC_IN
Cristal 48Mhz
3.3V
VDD
VDD
VDD
Vref
VDDA
C27 4.7uF
C28 100nF
C30 4.7uF
C29 100nF
C33 4.7uF
C31 100nF
C36 1uF
C34 100nF
C37 1uF
C38 10nF
GND
Alimentacion MCU
NO_STD_1
NO_STD_9
SAE_J1850_+
SAE_J1850_-
NO_STD_3
NO_STD_11
GND
NO_STD_12
NO_STD_13
HSCANH
HSCANL
K_LINE
L_LINE
NO_STD_8
12V_BAT
Conector OBD-2
F2
MF-NSMF050-2
3.3V
3.3_f
Protección contra cortocircuito
Pines de programacion
U7
X6512WV-04H-C60D30-E100
3.3V
SWDIO
SWCLK
GND
U6
X6512WV-04H-C60D30-E100
3.3V
TXD
RXD
GND
Alimentacion y tierra para placa de desarrollo
H1
HDR-F-2.54_1x2
3.3_f
H4
HDR-F-2.54_1x2
GND
10kΩ
R7
12V_BAT
15Ω
R8
BSS83P
Q3
Q4
MMBT3904-7-F
R9
22.1kΩ
U1
1kΩ
R10
5.1kΩ
BAT_ADC
PA5
GND
Medicion de V de bateria
LED_VERDE
APT2012LZGCK
3.3V
R19
330Ω
GND
APT2012LSECK/J3-PRV
LED_Ro
820Ω
U8
GND
LED_AZUL
APT2012LVBC/D
GPIO8
R20
330Ω
GND
LED_NARANJA
APT2012SECK/J4-PRV
GPIO3
820Ω
U9
GND
LEDs

A.1.7. Esquemático del adaptador de 90º del conector OBD-II