Raúl Carayol Rodríguez

Desarrollo de un Testing framework para apoyar la prueba de integración de software continua en una plataforma Microsoft Azure DevOps.

Trabajo Final de Máster

dirigido por Dr. José Luis Ramírez Falo

Máster en Ingeniería Industrial

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2024

Preámbulo

Este trabajo final de máster (TFM) contiene partes confidenciales que son propiedad de LEAR CORPORATION HOLDING SPAIN SLU. En este documento se muestra la **versión reducida**, donde se han quitado todas las partes que de una forma directa o indirecta están sujetas a esta confidencialidad.

Índice

1 Introducción

1.1 Lear Corporation

LEAR Corporation es una multinacional líder en el diseño y fabricación de sistemas eléctricos y electrónicos para automóviles, así como de asientos. Fundada en 1917 en Detroit, Michigan, la compañía ha evolucionado desde la fabricación de piezas metálicas para la industria de la aviación hasta convertirse en un gigante del sector automotriz. Hoy en día, LEAR está presente en 37 países, con más de 243 localizaciones y alrededor de 150,000 empleados, generando ventas anuales por valor de 18.6 mil millones de dólares.

LEAR también se destaca en la creación de sistemas de distribución eléctrica y electrónica, vitales para la operación de vehículos modernos. Estos sistemas incluyen arneses de cableado, módulos electrónicos y sistemas de gestión de energía, integrando tanto hardware como software para asegurar un rendimiento óptimo.

La planta de LEAR Corporation en Valls, ubicada en la provincia de Tarragona, es una de las instalaciones más importantes de la compañía en España. Esta planta se especializa en la fabricación de componentes electrónicos y sistemas eléctricos para automóviles. Dentro del entorno industrial de Tarragona, la planta de Valls juega un papel crucial, no solo por su capacidad de producción, sino también por su innovación tecnológica y su contribución al desarrollo económico local.

Figura 1. LEAR Valls.

En Valls, LEAR produce placas de circuitos y otros componentes electrónicos esenciales para el funcionamiento de los vehículos. Estos componentes forman parte de sistemas críticos como unidades de control del motor, sistemas de entretenimiento y módulos de seguridad.

Además de la producción de hardware, la planta de Valls es un centro de desarrollo de software para los componentes electrónicos que fabrican. El software desarrollado en esta instalación garantiza que los sistemas eléctricos y electrónicos funcionen correctamente y de manera eficiente, integrando tecnologías avanzadas y soluciones innovadoras.

La planta de Valls es un referente en la aplicación de nuevas tecnologías y procesos de fabricación avanzados. Su capacidad para innovar en el diseño y producción de componentes electrónicos sitúa a la región de Tarragona como un centro importante para la industria automotriz.

1.2 Motivación del proyecto

Este trabajo se ha realizado en la empresa LEAR Corporation, mencionada anteriormente, como parte del programa Trainee. Desde finales de julio hasta mayo, he estado en la empresa realizando prácticas y desarrollando el proyecto descrito en esta memoria del trabajo final de máster. Durante este periodo, he llevado a cabo el proyecto y la formación necesaria para desarrollarlo.

El proyecto surge a partir de la metodología utilizada en LEAR Corporation para testear el software de las placas desarrolladas. Para comprender mejor la situación, veamos el esquema general de una prueba típica de las que se utilizan actualmente.

Confidencial para Lear Corporation

2 Objetivo del proyecto

El objetivo principal del proyecto es desarrollar un framework de testing para realizar pruebas de software en los diversos proyectos de la empresa. Este framework debe ser lo suficientemente versátil para servir a cualquier proyecto, proporcionando un entorno integral que gestione todo el setup necesario, permitiendo el uso de los mismos tests en diferentes contextos. Un objetivo crucial es la integración sencilla con Microsoft Azure DevOps, facilitando el lanzamiento de pruebas desde esta plataforma y habilitando la implementación de CI/CD (Integración Continua y Despliegue Continuo) para los proyectos.

2.1 ¿Qué es un Framework?

Un framework es un conjunto de herramientas, librerías y buenas prácticas que proporcionan una estructura estándar para el desarrollo y pruebas de software. En el contexto de este proyecto, un framework de testing es una plataforma que facilita la creación, ejecución y gestión de pruebas automatizadas. Un buen framework de testing incluye componentes modulares que permiten realizar diferentes tipos de pruebas, integrar fácilmente nuevas funcionalidades y adaptarse a diversos proyectos y necesidades.

2.2 Principales Características del Testing Framework

Para cumplir con los objetivos planteados, el Testing Framework debe poseer las siguientes características:

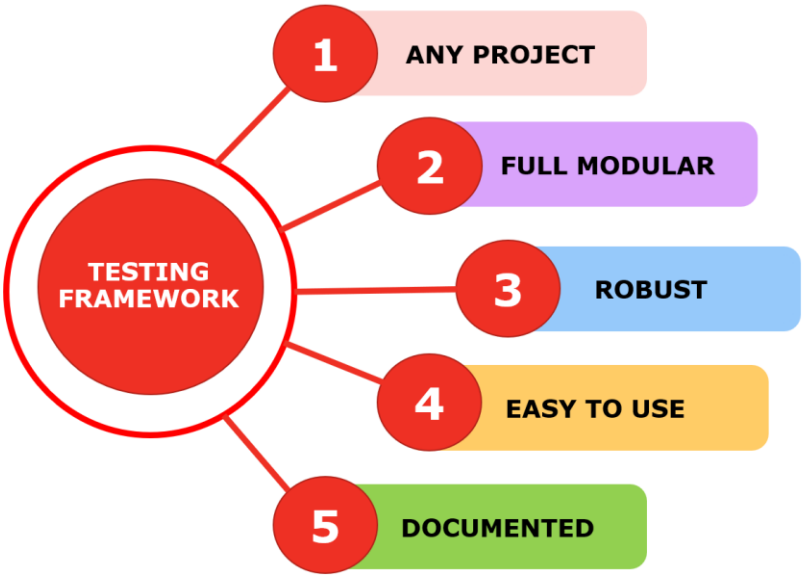


Figura 2. Caracteristicas del testing framework

1. **Compatibilidad con Cualquier Proyecto**:
   - El framework debe ser adaptable a una amplia variedad de proyectos dentro de la empresa, independientemente de su tamaño o complejidad. Esto asegura que los recursos y esfuerzos invertidos en el desarrollo del framework beneficien a toda la organización.

2. **Modularidad Completa**:
   - El diseño modular es esencial para que el framework sea escalable y personalizable. Cada componente del framework debe ser independiente, permitiendo su actualización o reemplazo sin afectar al resto del sistema. Esto facilita la adición de nuevas funcionalidades y la adaptación a nuevos requerimientos de prueba.
3. **Robustez**:
   - El framework debe ser robusto, capaz de manejar una amplia gama de escenarios de prueba y resistir fallos o errores inesperados. Esto incluye la capacidad de recuperarse de fallos y proporcionar información clara sobre cualquier problema que surja durante la ejecución de las pruebas.
4. **Facilidad de Uso**:
   - La usabilidad es una prioridad para asegurar que el framework pueda ser utilizado eficazmente por los desarrolladores y equipos de prueba. Una interfaz intuitiva y documentación clara son fundamentales para reducir la curva de aprendizaje y asegurar la adopción del framework en toda la empresa.
5. **Documentación Completa**:
   - Una documentación exhaustiva es clave para el éxito del framework. Esto incluye guías de usuario, ejemplos de uso, documentación de API y guías de integración. La documentación debe estar actualizada y accesible, facilitando la resolución de problemas y el aprovechamiento máximo de las funcionalidades del framework.

3 Planificación

La planificación del proyecto se diseñó para desarrollar primero las partes más específicas e independientes entre sí, como las librerías, y luego avanzar gradualmente hacia los componentes más integrados y dependientes, culminando en los AssemblyMethods y los Test Steps. Este enfoque asegura que las partes fundamentales estén completamente funcionales antes de abordar los niveles superiores de la estructura del proyecto, lo que es esencial para la integridad y fiabilidad de las pruebas finales.

La planificación del proyecto es la siguiente:

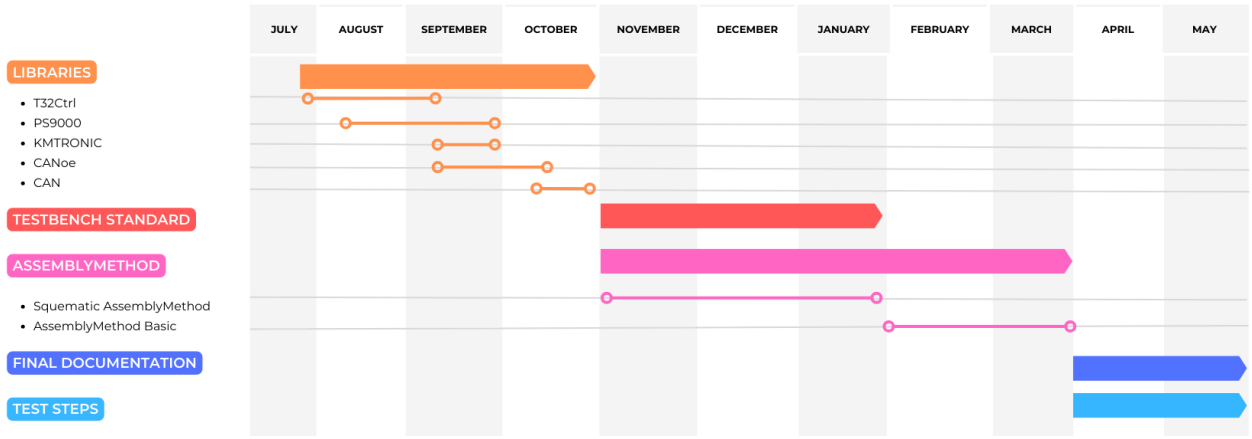


Figura 3. Diagrama de Gantt del proyecto

3.1 Fases de la Planificación del Proyecto

1. **Desarrollo de Librerías**:
    - **Objetivo**: Crear y validar las librerías fundamentales necesarias para la comunicación y el control de los dispositivos.
    - **Componentes**:
        - T32Ctrl
        - PS9000
        - KMTRONIC
        - CANoe
        - CAN
    - **Actividades**:
        - Programación de funciones básicas.
        - Pruebas unitarias y de integración para asegurar la funcionalidad.
2. **Desarrollo deL TestBench Standard**:
    - **Objetivo**: Implementar el Testbench que gestiona y controla la ejecución a un nivel más alto utilizando las librerías desarrolladas.
    - **Componentes**:
        - TestBench.
        - Controllers.
        - ProjectConfig.
    - **Actividades**:
        - Desarrollo de funciones de control y gestión de ejecución.
        - Desarrollo del Setup de los diferentes componentes.
        - Pruebas integradas con las librerías desarrolladas.
3. **Integración de Componentes**:
    - **Objetivo**: Integrar todas las librerías y módulos desarrollados para formar un sistema cohesivo.
    - **Componentes**:
        - Ensamblado de módulos en un framework unificado.
        - Pruebas de integración y validación del sistema completo.
    - **Actividades**:
        - Ensamblaje del sistema.
        - Pruebas de funcionalidad y robustez.
4. **Desarrollo de AssemblyMethods y Test Steps**:
    - **Objetivo**: Crear los métodos y pasos de prueba específicos necesarios para validar los proyectos.
    - **Requisitos Previos**: Las librerías y módulos integrados deben estar completamente funcionales.
    - **Actividades**:
        - Desarrollo de métodos de ensamblaje y pasos de prueba.
        - Pruebas exhaustivas para asegurar la fiabilidad y precisión

3.2 Planificación Inicial vs. Realidad

La planificación inicial se aproximó mucho al cronograma seguido durante todo el proceso del proyecto. Gracias a una gestión eficiente del tiempo y recursos, se logró completar las fases críticas dentro del tiempo estimado, dejando suficiente margen para realizar pruebas en proyectos reales. Esto permitió no solo validar el framework en condiciones prácticas, sino también hacer ajustes y mejoras basadas en escenarios de uso real.

4 Estado del arte

4.1 El mundo del automóvil hoy en día

El sector del automóvil, un pilar fundamental de la industria manufacturera global, está experimentando una transformación profunda y multifacética. Hoy en día, hablar de automóviles no es solo hablar de vehículos en sí, sino de una confluencia de tecnología, sostenibilidad, y cambios en la dinámica del consumidor que están remodelando este campo de manera radical.

Uno de los cambios más notables es la electrificación. La preocupación por el cambio climático ha llevado a una transición acelerada hacia los vehículos eléctricos (VE). Empresas como Tesla, Nissan y BMW están a la vanguardia, desarrollando automóviles que no solo son eficientes, sino que también buscan ser más amigables con el medio ambiente. Este cambio hacia la electrificación no solo está afectando la producción de vehículos, sino que también está transformando las cadenas de suministro, especialmente en la producción de baterías y sistemas de gestión de energía. Además, la infraestructura de carga, un componente crucial para la adopción masiva de VEs, está en rápida expansión con estaciones de carga rápida y redes de carga domiciliaria surgiendo en todo el mundo.

La conectividad y digitalización son otras fuerzas motrices en el sector. Los vehículos de hoy están cada vez más conectados, lo que significa que pueden comunicarse entre sí y con la infraestructura vial, mejorando la seguridad y la eficiencia del tráfico. La incorporación de tecnologías de la información y la comunicación en los automóviles ha permitido avances significativos en áreas como el mantenimiento predictivo y la personalización de la experiencia del usuario. La recopilación y análisis de datos en tiempo real se han vuelto esenciales, proporcionando insights que permiten monitorear el estado del vehículo y anticipar posibles problemas antes de que ocurran.

Figura 4. Ejemplo de cableado de turismo común.

En paralelo, la automatización y la conducción autónoma están revolucionando la manera en que concebimos la movilidad. Los niveles de autonomía varían desde la asistencia básica al conductor hasta la capacidad de conducción completamente autónoma, con vehículos equipados con avanzados sensores, cámaras y sistemas de inteligencia artificial que permiten la navegación sin intervención humana. Este avance está cambiando no solo la forma en que usamos los vehículos, sino también cómo se fabrican, requiriendo un alto grado de precisión y calidad en los componentes electrónicos y sistemas de software.

Los cambios en las preferencias de movilidad también juegan un papel crucial. La propiedad individual de vehículos está dando paso a modelos de movilidad compartida, como el car-sharing y ride-hailing. Los consumidores valoran cada vez más la experiencia del usuario, la conectividad y las características de seguridad avanzadas en los vehículos, lo que está forzando a los fabricantes a innovar constantemente.

La sostenibilidad y las regulaciones ambientales están impulsando a las empresas a adoptar tecnologías más limpias y eficientes. Las normativas de emisiones son cada vez más estrictas, y esto ha llevado a una mayor inversión en tecnologías que promueven la sostenibilidad. Además, conceptos como la economía circular están ganando terreno, promoviendo la reutilización y reciclaje de materiales y el diseño de vehículos más sostenibles a lo largo de su ciclo de vida.

En cuanto a la producción industrial, la Industria 4.0 está dejando una huella significativa. La automatización y robótica están optimizando las líneas de producción, aumentando la eficiencia y reduciendo costos. La fabricación aditiva, o impresión 3D, está permitiendo la creación de componentes complejos con mayor rapidez y menor coste. El Internet de las Cosas (IoT) conecta maquinaria y equipos, facilitando la monitorización y optimización en tiempo real de los procesos de fabricación.

Los sistemas de producción también se están volviendo más flexibles para adaptarse a la demanda de vehículos personalizados. La personalización en masa y la metodología Lean Manufacturing son esenciales para eliminar desperdicios y mejorar continuamente los procesos. Estas innovaciones no solo mejoran la eficiencia, sino que también permiten a los fabricantes responder rápidamente a las necesidades cambiantes del mercado.

No obstante, estos avances vienen con desafíos. Las cadenas de suministro han mostrado ser vulnerables a interrupciones globales, como lo evidenció la pandemia de COVID-19. Por ello, las empresas están adoptando estrategias para aumentar la resiliencia y diversificar sus proveedores. Además, la sostenibilidad en la cadena de suministro es una prioridad, con un enfoque en la reducción de la huella de carbono y la trazabilidad de los materiales.

Finalmente, la capacitación de la fuerza laboral es crucial. La digitalización y automatización requieren nuevas habilidades en áreas como inteligencia artificial, análisis de datos y mantenimiento de sistemas automatizados. La atracción y retención de talento especializado son vitales para mantener la competitividad en un sector que evoluciona rápidamente.

4.2 Software Testing

4.2.1 ¿Qué es la prueba de software?

La prueba de software se define como una actividad para verificar si los resultados reales coinciden con los resultados esperados y asegurar que el sistema de software esté libre de defectos. Esto implica ejecutar una parte o todo el sistema de software para evaluar una o más propiedades de interés. Además, ayuda a identificar errores, brechas o requisitos faltantes en comparación con los requisitos originales. Las pruebas de software pueden hacerse manualmente o utilizando herramientas automatizadas. A veces, se habla de pruebas de software como pruebas de Caja Blanca y Caja Negra.

4.2.2 Pruebas de Software Embebido

Las pruebas de software embebido se refieren a probar sistemas embebidos. Este tipo de pruebas es similar a otras pruebas de software. Se evalúa el software embebido para verificar su rendimiento, consistencia y cumplimiento con los requisitos del cliente del equipo de desarrollo de software. Estas pruebas aseguran que el software cumpla con los estándares de calidad y los requisitos necesarios.

4.2.3 Razones para realizar pruebas de software

1. Encontrar errores en el software.
2. Reducir el riesgo tanto para los usuarios como para la empresa.
3. Reducir los costos de desarrollo y mantenimiento.
4. Mejorar el rendimiento del software.

4.2.4 Tipos de pruebas de software

Las pruebas de software se clasifican en tres categorías principales:

Confidencial para Lear Corporation

4.2.5 Tipos de Pruebas de Software Embebido

Fundamentalmente, hay cinco niveles de pruebas que se pueden aplicar al software embebido:

1. **Pruebas Unitarias de Software**
    - Se prueba un módulo de unidad, que puede ser una función o una clase. Estas pruebas las realiza principalmente el equipo de desarrollo, en un modelo de revisión por pares. Se desarrollan casos de prueba basados en las especificaciones del módulo.
2. **Pruebas de Integración**
    - Se dividen en dos segmentos: pruebas de integración de software y pruebas de integración de software/hardware. Se evalúa la interacción entre los dispositivos periféricos integrados y el software.
3. **Pruebas Unitarias del Sistema**

   - Se prueba un marco completo que incluye todo el código de software y todas las piezas relacionadas con el sistema operativo en tiempo real (RTOS), como interrupciones, mecanismos de tareas y comunicaciones. Se observa el uso de recursos del sistema para evaluar su capacidad de soportar la ejecución del sistema embebido. Para esto, se prefiere la prueba de caja gris.
4. **Pruebas de Integración del Sistema**
   - Se prueba un conjunto de componentes dentro de un solo nodo. Los Puntos de Control y Observación (PCOs) son una mezcla de protocolos de comunicación relacionados con la red y RTOS.
5. **Pruebas de Validación del Sistema**
   - Se prueba un subsistema completo o el sistema embebido completo. El objetivo es cumplir con los requisitos funcionales de entidades externas, que pueden ser una persona o un dispositivo en una red de telecomunicaciones.

4.2.6 ¿Qué hace un tester de software?

En un día típico de trabajo, estarás ocupado entendiendo los documentos de requisitos, creando casos de prueba, ejecutándolos, reportando y volviendo a probar errores, asistiendo a reuniones de revisión y participando en otras actividades de equipo. Una vez que te familiarices con las pruebas manuales, puedes especializarte en:

- **Pruebas de Automatización**: Serás responsable de automatizar la ejecución de casos de prueba manuales, ahorrando tiempo.
- **Pruebas de Rendimiento**: Evaluarás la capacidad de respuesta del sistema, el tiempo de carga, la carga máxima que puede manejar, etc.
- **Analista de Negocios**: Los testers tienen un conocimiento amplio del negocio. Un progreso natural en la carrera es convertirse en analista de negocios, responsable de analizar y evaluar el modelo de negocio y los flujos de trabajo de la empresa.

Comparación entre Pruebas de Software y Pruebas Embebidas

Pruebas de Software	Pruebas Embebidas
Relacionadas solo con software.	Relacionadas tanto con software como con hardware.
El 90% de las pruebas son manuales de caja negra.	Se realizan en sistemas embebidos o chips, pueden ser de caja negra o blanca.
Se enfocan en comprobar la interfaz gráfica, la funcionalidad, la validación y algunas pruebas de bases de datos.	Se enfocan en el comportamiento del hardware según la cantidad de entradas que recibe.

Taula 2. Comparación entre Pruebas de Software y Pruebas Embebidas

4.2.7 7 principios de las Pruebas de Software

1. **Las pruebas exhaustivas no son posibles**
   - No podemos probar todas las combinaciones posibles. Necesitamos una cantidad óptima de pruebas basada en una evaluación de riesgos.
2. **Agrupamiento de Defectos**
   - La mayoría de los defectos se encuentran en un pequeño número de módulos. Es la aplicación del Principio de Pareto: el 80% de los problemas se encuentran en el 20% de los módulos.

3. **Paradoja del Pesticida**
    - Usar las mismas pruebas repetidamente será menos efectivo para encontrar nuevos errores. Las pruebas deben revisarse y actualizarse regularmente.
4. **Las pruebas muestran la presencia de defectos**
    - Las pruebas pueden mostrar la presencia de defectos, no la ausencia. Aunque no se encuentren errores, no es prueba de que el software sea perfecto.
5. **Falacia de la Ausencia de Errores**
    - Un software con pocos errores puede ser inutilizable si no cumple con los requisitos del cliente.
6. **Pruebas Tempranas**
    - Las pruebas deben comenzar lo antes posible en el ciclo de desarrollo para identificar defectos temprano, cuando son más baratos de solucionar.
7. **Las pruebas dependen del contexto**
    - Diferentes aplicaciones requieren diferentes enfoques, metodologías y tipos de pruebas.

4.3 Continuos Testing

4.3.1 ¿Qué es el Continuous Testing?

El Continuous Testing es un tipo de prueba de software que se basa en probar desde el principio, probar con frecuencia, en todos lados, automatizar y gestionar de manera ágil. Implica ejecutar pruebas de manera continua e integrar a los testers en equipos multifuncionales para probar funcionalidades y ofrecer retroalimentación rápida.

Esta estrategia evalúa la calidad en cada paso del proceso de entrega continua e involucra a desarrolladores, DevOps, QA y sistemas operativos. Aunque DevOps acelera los procesos y los hace más eficientes, las empresas deben enfocarse tanto en la calidad como en la velocidad. QA no debe estar fuera del entorno DevOps; debe ser una parte fundamental.

4.3.2 ¿En qué se diferencia el Continuous Testing?

Antes, las pruebas se realizaban de manera escalonada. El software pasaba de un equipo a otro, con fases definidas de desarrollo y QA. Los equipos de QA siempre pedían más tiempo para asegurar la calidad, con el objetivo de que esta prevaleciera sobre el cronograma del proyecto.

El Continuous Testing, por otro lado, implica pruebas ininterrumpidas. En un proceso continuo de DevOps, un cambio en el software (candidato a lanzamiento) se mueve constantemente desde el desarrollo hasta las pruebas y la implementación.

Confidencial para Lear Corporation

Por ejemplo, cada vez que un desarrollador sube código al repositorio, se ejecutan automáticamente pruebas unitarias con Azure DevOps. Si las pruebas fallan, se rechaza la compilación y se notifica al desarrollador. Si pasa, se despliega en servidores de rendimiento y QA para pruebas funcionales y de carga exhaustivas, que se ejecutan en paralelo. Si todo está bien, el software se despliega en preproducción para ser entregado en cualquier momento.

4.3.3 Diferencias entre Continuous Testing y Test Automation

Parámetro	Test Automation	Continuous Testing
Definición	Uso de herramientas para automatizar tareas.	Metodología de pruebas de software enfocada en calidad continua.
Propósito	Ejecutar tareas repetitivas rápidamente y con menos errores.	Identificar riesgos, abordarlos y mejorar la calidad del producto.
Requisito previo	Posible sin integrar pruebas continuas.	No se puede implementar sin automatización de pruebas.
Tiempo	Lanzamientos de software pueden tardar de meses a años.	Lanzamientos pueden ser semanales o incluso horarios.
Retroalimentación	Retroalimentación regular tras cada lanzamiento.	Retroalimentación instantánea en cada etapa.
Historia	Pruebas automatizadas se realizan desde hace décadas.	Concepto relativamente nuevo.

Taula 2.Diferencias entre Continuous Testing y Test Automation

4.3.4 Cómo hacer Continuous Testing

- Utiliza herramientas para generar suites de pruebas automáticas a partir de historias de usuario/requisitos.
- Crea un entorno de pruebas.
- Copia y anonimiza datos de producción para crear un banco de datos de prueba.
- Usa virtualización de servicios para probar APIs.
- Realiza pruebas de rendimiento en paralelo.

4.3.5 Beneficios del Continuous Testing

- Acelera la entrega de software.
- Mejora la calidad del código.
- Ayuda a evaluar los riesgos empresariales.
- Se integra sin problemas en el proceso de DevOps.
- Crea un proceso ágil y confiable en horas en lugar de meses.
- Acelera el tiempo de lanzamiento con un mecanismo de retroalimentación continua.
- Une equipos que tradicionalmente estaban aislados, superando la desconexión entre desarrollo, pruebas y operaciones.
- La automatización de pruebas ayuda a mantener la consistencia con la misma configuración para todas las pruebas relevantes.
- Enfatiza las expectativas empresariales para mitigar riesgos.
- Proporciona acceso ubicuo al entorno de pruebas con la virtualización de servicios.

4.3.6 Desafíos del Continuous Testing

- El proceso tradicional limita el cambio cultural entre desarrolladores y profesionales de QA.
- Falta de habilidades de DevOps y herramientas adecuadas para pruebas en entornos ágiles y DevOps.
- Entornos de prueba heterogéneos que no reflejan el entorno de producción.
- Procesos de prueba convencionales y gestión de datos de prueba poco definidos.
- Ciclos de integración de código más largos que generan problemas de integración y corrección tardía de defectos.
- Recursos y entornos de prueba insuficientes e ineficaces.
- Arquitectura de aplicaciones compleja y lógica empresarial que restringe la adopción de DevOps.

4.3.7 Conclusión sobre Continous Testing

- En la ingeniería de software, el Continuous Testing es el proceso de probar temprano, probar con frecuencia, en todos lados y automatizar.
- El método antiguo de pruebas se basaba en un enfoque escalonado, pasando el software de un equipo a otro.
- El Continuous Testing proporciona retroalimentación útil en cada etapa del pipeline de entrega.
- Ayuda a mejorar la calidad del código.
- El proceso tradicional limita el cambio cultural entre desarrolladores y profesionales de QA.
- Los ciclos de integración de código más largos generan problemas de integración y corrección tardía de defectos.

4.4 Microsoft Azure DevOps

4.4.1 ¿Qué es Microsoft Azure DevOps?

Microsoft Azure DevOps es una plataforma que ofrece varias herramientas para mejorar la colaboración en equipo. También cuenta con herramientas para procesos de construcción automatizada, pruebas, control de versiones y gestión de paquetes.

4.4.2 Herramientas Ágiles para la Colaboración

Azure DevOps incluye herramientas ágiles que nos ayudan a planificar, rastrear y discutir nuestro trabajo, incluso con otros equipos. Estas herramientas modernas y simples, como los tableros Kanban, los backlogs, los tableros de scrum y los dashboards, se adaptan a las necesidades de tu equipo y pueden escalar fácilmente.

4.4.3 Integración con Git y Control de Versiones

Puedes conectar y enviar tu código de forma segura a tus repositorios de Git desde cualquier IDE, editor o cliente de Git. Realiza revisiones de código más efectivas con discusiones en

hilos y la integración continua para cada cambio. Utiliza forks para fomentar la colaboración con flujos de trabajo internos.

4.4.4 Automatización de Construcciones y Despliegues con CI/CD

Con Azure DevOps, puedes construir, probar y desplegar usando CI/CD (Integración Continua/Despliegue Continuo) que funciona con cualquier lenguaje, plataforma y nube. Automatiza tus construcciones y despliegues con Pipelines para que puedas dedicar más tiempo a ser creativo y menos a los detalles técnicos.

4.4.5 Gestión de Paquetes Integrada

Crea y comparte paquetes Maven, npm, NuGet y Python desde fuentes públicas y privadas. Añade gestión de paquetes totalmente integrada a tus pipelines de CI/CD con un solo clic, y comparte feeds de paquetes con equipos de cualquier tamaño.

4.4.6 Mejora de la Calidad del Código con Pruebas Planificadas y Exploratorias

Azure DevOps permite mejorar la calidad del código mediante servicios de pruebas planificadas y exploratorias. Captura datos ricos en escenarios mientras ejecutas pruebas para hacer que los defectos descubiertos sean más manejables. Aprovecha la trazabilidad de extremo a extremo y la calidad para tus historias y características.

4.4.7 Aplicación en el Sector de la Automoción

En el sector de la automoción, donde la precisión y la seguridad son cruciales, Azure DevOps puede ser especialmente útil. Permite realizar pruebas continuas durante todo el ciclo de desarrollo del software automotriz. Desde el diseño hasta la producción, se pueden automatizar las pruebas para cada componente del software del vehículo.

1. **Automatización de Pruebas de Software Embarcado**
    - En el desarrollo de software para automóviles, se pueden automatizar las pruebas para validar el rendimiento y la consistencia del software embarcado. Esto asegura que cada pieza del software cumpla con los requisitos del cliente.
2. **Integración Continua**
    - Cada vez que un desarrollador introduce cambios en el código, se puede configurar una pipeline que ejecute automáticamente una serie de pruebas. Si las pruebas fallan, se notifica al desarrollador de inmediato, permitiendo correcciones rápidas y manteniendo la calidad del software.
3. **Gestión de Versiones y Colaboración**
    - Los equipos pueden trabajar juntos de manera más eficiente, gestionando versiones de software y revisiones de código de manera colaborativa. Esto es especialmente importante en proyectos grandes y complejos como los de la industria automotriz.
4. **Mejora de la Calidad y Seguridad**
    - Con las pruebas continuas, se pueden identificar y corregir defectos a lo largo de todo el proceso de desarrollo, mejorando la calidad y la seguridad del software que se integrará en los vehículos.
5. **Trazabilidad y Cumplimiento**

- Mantener un registro detallado de cada cambio y prueba permite asegurar el cumplimiento de los estándares de la industria y las regulaciones de seguridad.

5 Arquitectura del proyecto

Para realizar los diferentes tests de los diferentes proyectos, se necesita tener un conjunto de dispositivos, instrumentos y cargas conectadas a la placa que se quiere testear.

Por ello para cada proyecto se monta un armario con los diferentes dispositivos conectados a la placa y también conectados a un ordenador (Test host). También se tendrá un conjunto de cargas eléctricas que simulan diferentes partes de un automóvil o diferentes situaciones requeridas, pero para este trabajo no es de interés conocer en profundidad como esta conformado esto, por lo que se tratará como "caja negra".

Será **en este ordenador (Test host) donde el testing framework se ejecutará** e interactuará con los diferentes dispositivos con tal de poder llevar a cabo las diferentes pruebas.

Con tal de ilustrar mejor lo anteriormente comentado, se muestra el caso real de un proyecto en el que se ha probado el testing framework.

Por la forma de funcionar el testing framework y teniendo en cuenta los principales objetivos, se ha conformado el siguiente esquema del proyecto.

La arquitectura tiene tres grandes partes diferenciables:

- **Librerías**
  - Son ficheros independientes entre si que tienen como objetivo que cada una de ellas controle o interactúe con cada uno de los dispositivos que son parte del sistema de testeo.
- **TestBench**
  - Es la parte que une las funciones de los tests con las librerías, además también es quien gestiona todo proceso de inicialización de los dispositivos y control de todos los errores que pueden ocurrir durante la ejecución.

6 Desearrollo del proyecto

En esta sección se expondrá todo el desarrollo técnico llevado a cabo durante el proyecto en cada una de las partes. El orden en el que se exponen cada una de las partes es por orden cronológico a como se desarrollaron.

6.1 Librerías

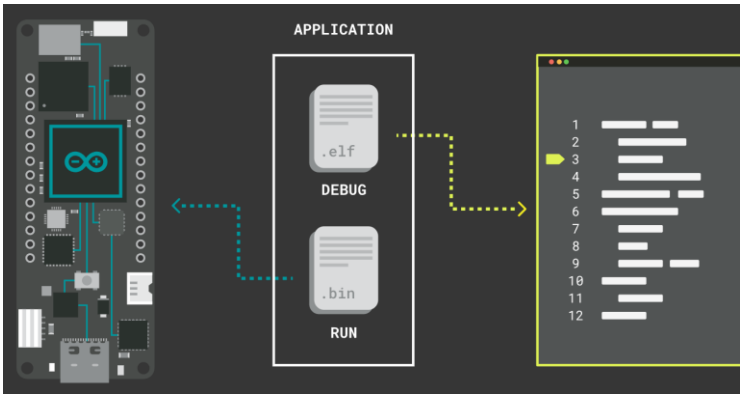
Para el desarrollo del Framework Test Environment se han desarrollado 5 librerías que son las encargadas de realizar toda la parte de bajo nivel para interactuar y controlar los diferentes dispositivos o software con el objetivo de poder realizar las acciones necesarias para hacer los tests.

Todas las librerías constan de al menos dos partes y algunas con tres. Primero tenemos la parte principal, que es el código principal de la librería con todas las funciones y métodos necesarios para poder interactuar y controlar los diferentes dispositivos. Luego esta la parte de Test que es un proyecto, que tienen todas las librerías, que constan de un grupo tests para comprobar que la librería funciona correctamente y según lo esperado.

6.1.1 Trace 32

En el mundo del desarrollo de sistemas embebidos, uno de los mayores desafíos que enfrentan los ingenieros es encontrar y corregir errores en el código que corre en microcontroladores y otros dispositivos electrónicos. Aquí es donde entran en juego los debuggers. Un **debugger** es una herramienta esencial que permite a los desarrolladores observar y controlar la ejecución del programa en tiempo real, ayudando a identificar y solucionar problemas de manera efectiva.

Imagina que estás escribiendo un libro y tienes la posibilidad de detenerte en cualquier página para revisar cada palabra, línea por línea, asegurándote de que todo está en su lugar antes de continuar. Así funcionan los debuggers, pero con el código. Se conectan a las placas electrónicas a través de interfaces estándar como JTAG o SWD, permitiendo una comunicación directa con el procesador. Esta conexión es como tener una lupa y un control remoto para tu programa.



Uno de los superpoderes de los debuggers es la capacidad de establecer **breakpoints**. Estos son puntos en el código donde la ejecución se detiene automáticamente, permitiendo al ingeniero inspeccionar el estado del sistema en ese momento específico. Es como presionar pausa en una película para analizar una escena en detalle. Además, los debuggers permiten ejecutar el código paso a paso, una función conocida como stepping, que permite observar cómo se comporta el programa en cada línea de código, lo cual es invaluable para entender qué está pasando en cada momento.

Otra característica genial es la capacidad de monitorear variables específicas mediante **watchpoints**. Esto es útil para ver cuándo y cómo cambian ciertos valores en el programa, como si tuvieras un sensor que te avisa cada vez que alguien toca tu libro favorito en una biblioteca.

Pero no todo se trata de observar. Los debuggers también permiten interactuar con el sistema. Puedes leer y escribir en la memoria, modificar el contenido de los registros del procesador y hasta interactuar con los periféricos del microcontrolador. Es como ser un cirujano que puede ajustar los órganos de un paciente en tiempo real mientras observa sus efectos inmediatos.

Ahora, hablando de herramientas de alta gama en este campo, Lauterbach TRACE32 es uno de los nombres más destacados. Lauterbach, una empresa alemana conocida por sus soluciones avanzadas, ha desarrollado TRACE32, una familia de herramientas de depuración y trazado que es altamente valorada en la industria.

Figura 5. PowerDebug E40 with an IDC20A Debug Probe

TRACE32 es como el equivalente a un equipo de laboratorio de última tecnología para ingenieros de sistemas embebidos. No solo te permite hacer todo lo que hace un debugger tradicional, sino que lleva las capacidades al siguiente nivel. Por ejemplo, su capacidad de trazado es impresionante. Imagina poder grabar y analizar la ejecución de tu programa a lo largo del tiempo, viendo cada evento y cada instrucción ejecutada como si pudieras revisar cada movimiento en una partida de ajedrez después de que ha terminado. Esto es invaluable para diagnosticar problemas complejos y entender el rendimiento del sistema.

Otra gran ventaja de TRACE32 es su soporte para sistemas **multicore**, es decir, procesadores con múltiples núcleos de ejecución. En un mundo donde los dispositivos se están volviendo cada vez más complejos, esta capacidad es crucial. Además, TRACE32 es compatible con una amplia gama de interfaces de depuración, lo que significa que puedes usarlo con casi cualquier plataforma de hardware.

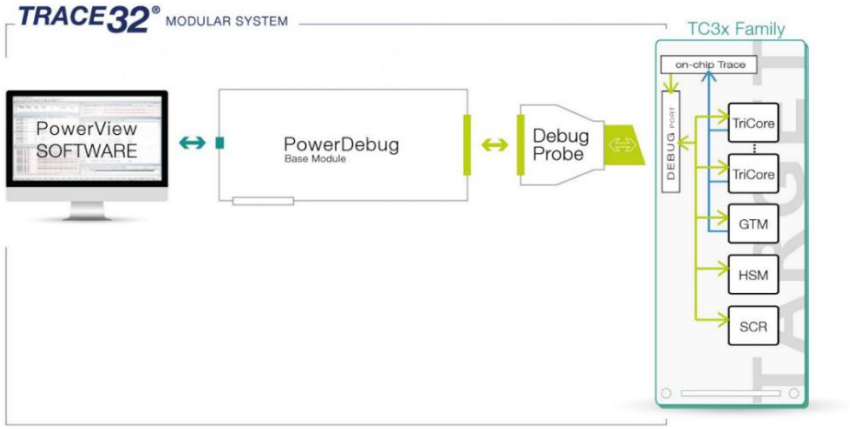


Figura 6. TRACE32 Modular System

El entorno de desarrollo integrado de TRACE32 es otro punto fuerte. Ofrece herramientas para escribir, compilar, depurar y trazar el código, todo en un solo lugar. Su interfaz de usuario es altamente configurable y proporciona vistas detalladas del estado del sistema, la memoria, los registros y más. Es como tener un panel de control completo para tu proyecto.

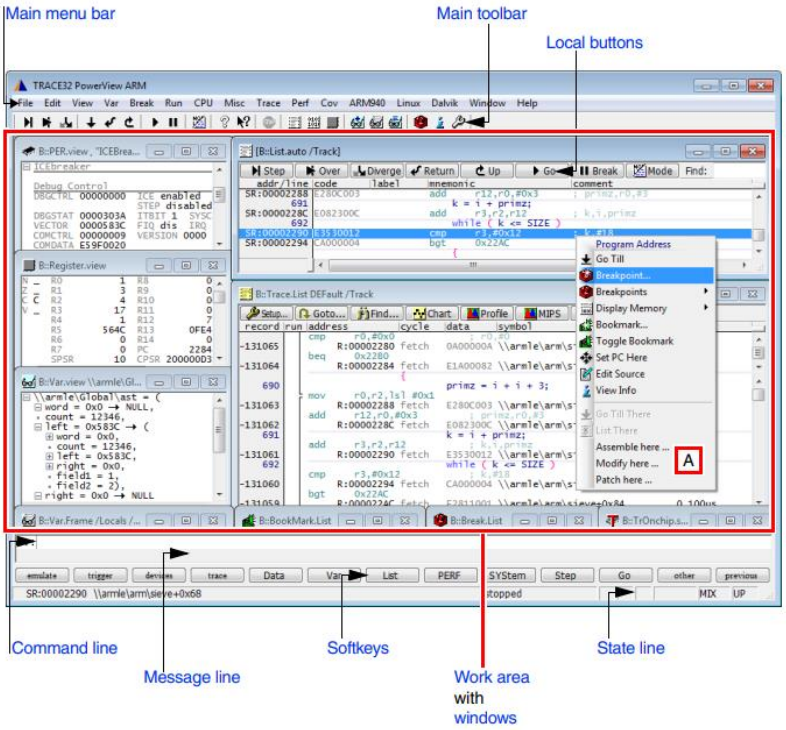


Figura 7. TRACE32 PowerView

Además, TRACE32 permite la automatización de tareas de depuración y prueba mediante scripts. Esto es particularmente útil cuando necesitas ejecutar pruebas repetitivas y complejas, mejorando la eficiencia del desarrollo y ahorrando tiempo. Lauterbach también proporciona un excelente soporte técnico y actualizaciones regulares, asegurando que las herramientas se mantengan al día con las últimas tecnologías y procesadores.

6.1.1.1 Trace 32 Main Part

La parte principal de la biblioteca TRACE32 consta de dos proyectos: t32apinet (desarrollado en C) y T32Ctrl (desarrollado en C#). El funcionamiento esencial de la biblioteca se basa en el envío y la recepción de comandos específicos de TRACE32, detallados en la documentación técnica de Lauterbach.

t32apinet

El proyecto t32apinet es de bajo nivel y se encarga de gestionar la comunicación directa con TRACE32. Gran parte del código de este proyecto es proporcionado directamente por Lauterbach, la empresa proveedora del debugger. Este proyecto se enfoca en la transmisión eficiente y confiable de comandos y respuestas, garantizando la integración precisa y el control con TRACE32.

6.1.1.2 T32Ctrl Test

Para poder testear que funciona la librería T322Ctrl se ha seguido la estrategia de realizar un test para cada función, de esa forma se cubre el 100% de funcionalidades de la librería.

6.1.2 PowerSupply

La fuente de alimentación es un dispositivo imprescindible para realizar el Software Testing de las placas en los proyectos deseados. Su uso principal es alimentar las placas, pero también pueden controlar ciertas cargas y adaptarse a diferentes escenarios específicos de un test. La información proporcionada por una fuente de alimentación permite determinar si el comportamiento de la placa es el esperado, verificando así su correcto funcionamiento. Por esta razón, es fundamental poder extraer estos datos de manera precisa.

6.1.2.1 Importancia de las Fuentes de Alimentación en el Framework de Pruebas

Las fuentes de alimentación son un componente crítico dentro del Framework Test Environment. El objetivo es controlar estos dispositivos de forma automatizada y sin supervisión humana. Un manejo incorrecto, como dejar una fuente encendida cuando no es necesario, puede provocar situaciones de peligro o incluso accidentes. Por este motivo, durante el desarrollo de esta librería, se ha prestado especial atención al correcto funcionamiento de las fuentes de alimentación, probándolas exhaustivamente en todas las situaciones posibles.

6.1.2.2 Control de las Fuentes de Alimentación con SCPI

Para controlar las fuentes de alimentación en esta librería, se utilizan comandos SCPI (Standard Commands for Programmable Instruments) enviados a través de un puerto USB. A continuación, se explica brevemente qué es SCPI y cómo funciona.

6.1.2.3 ¿Qué es SCPI?

SCPI, o Standard Commands for Programmable Instruments, es un estándar para comunicar y controlar instrumentos electrónicos. Fue desarrollado en la década de 1980 para unificar y simplificar la forma en que los dispositivos de prueba y medición interactúan con controladores y computadoras. SCPI define un conjunto de comandos textuales que permiten configurar, controlar y obtener datos de los dispositivos de manera consistente.

6.1.2.4 Funcionamiento de SCPI

Comandos Estructurados: SCPI utiliza una estructura jerárquica de comandos, organizada en subsistemas. Cada comando tiene una estructura clara y predecible, lo que facilita su comprensión y uso.

Ejemplo de un comando SCPI para configurar un voltaje:

Comunicación: Los comandos SCPI se envían a través de interfaces estándar como USB, GPIB (General Purpose Interface Bus), Ethernet o RS-232. En este caso, se utiliza el puerto USB para la comunicación.

Consistencia: SCPI proporciona una interfaz uniforme para una amplia variedad de instrumentos, permitiendo que los mismos comandos se utilicen en diferentes dispositivos.

Respuesta: Los dispositivos que reciben comandos SCPI responden con información relevante o confirman la ejecución de los comandos. Por ejemplo, se puede enviar un comando para medir el voltaje y recibir el valor medido en respuesta.

Ejemplo de Uso de SCPI

Para controlar una fuente de alimentación, se podrían enviar los siguientes comandos SCPI:

Configurar el voltaje de salida a 5V:

```
SOUR:VOLT 5.0
```

Activar la salida:

```
OUTP ON
```

Leer el valor actual del voltaje:

```
SOUR:VOLT?
```

6.1.2.5 Implementación en la Librería

La librería desarrollada utiliza estos comandos SCPI para interactuar con las fuentes de alimentación. Por ejemplo, al iniciar una prueba, la librería puede configurar la fuente de alimentación con los parámetros deseados, encenderla, y monitorear continuamente el voltaje y la corriente para asegurar que el dispositivo bajo prueba se comporte como se espera. Además, se han implementado medidas de seguridad para asegurarse de que las fuentes se apaguen correctamente cuando no se están utilizando, minimizando el riesgo de accidentes.

6.1.2.6 PS9000

Esta librería se ha diseñado de tal forma que simplemente con especificar el puerto en el que esta conectada la fuente de alimentación esta ya se configura automáticamente. Esto lo puede hacer gracias a que la librería identifica y detecta el modelo de la fuente de alimentación, con lo que con esa información para puede configurar todo lo que es necesario para esa fuente en específico.

La función para detectar el modelo se basa en uno de los comandos mas universales que hay dentro de los SCPI como es el comando **"*IDN?"** que sirve para que el instrumento de identifique. Con el nombre del dispositivo se puede obtener el modelo, así como obtener los valores máximos de tensión, corriente y potencia, esto la librería lo utiliza para setear los valores máximos y de esta forma controlar que el usuario que utilice la librería no trate de poner un valor superior al que la fuente de alimentación puede otorgar.

En caso de que en el proceso de detección del modelo de la fuente de alimentación ocurra un error que no permita configurar los valores máximos, por defecto se tendrá los valores del modelo de menor potencia a modo de proteger en el peor caso posible.

La estructura general de ficheros está constituida por el fichero de mas alto nivel en la arquitectura que es el fichero **PowerSupplyController**.cs, es el encargado de gestionar en alto nivel la situación de la fuente de alimentación, por ejemplo si se quiere hacer una cierta acción este se encarga de comprobar que se ha realizado correctamente y de leer el estado de la fuente de alimentación. Son las funciones del fichero **PowerSuppluController** las que se ven desde fuera de la librería.

PowerSupplyController utiliza el fichero de mas bajo nivel, el fichero Source.cs, para hacer todas las acciones. Es este fichero de más bajo nivel (**Source.cs**) es el encargado de enviar los comandos SCPI, todo el fichero esta formado por un seguido de funciones que lo único que hace es enviar el correspondiente comando para la función deseada.

Para evitar problemas con las comas o los puntos, que puede ocurrir si donde se ejecuta la librería es en un idioma u otro que utiliza diferente carácter para separar los decimales, se introduce esta clase para poner un único formato de separación de los decimales. (Sin esta clase puede ocurrir que en un ordenador con diferente idioma pueda enviar en vez de un valor de 1 V este lo interprete como 1000 V).

6.1.2.7 PS9000 Test

Para asegurar el correcto funcionamiento de la librería de la fuente de alimentación, se han diseñado y ejecutado una serie de tests exhaustivos para cada función. Estos tests no solo verifican la funcionalidad básica, sino que también exploran diversos escenarios y parámetros para evaluar el comportamiento de la librería en situaciones variadas.

6.1.2.8 PS9000 GUI

Para la librería de la fuente de alimentación se ha diseñado una interfaz grafica que permite de una forma rápida y visual poder controlar la fuente de alimentación.

En esta ocasión para la fuente de alimentación se han diseñado dos GUI (Graphic user interface), la primera interfaz esta basada en la apariencia real que tiene en panel de control físico que tiene la fuente de alimentación. Por una parte, esto permite que alguien familizarizado con la fuente de alimentación en el laboratorio sepa perfectamente como interactuar con ella mediante esta interfaz grafica puesto que es exactamente igual. Por otro lado esto también aporta una apariencia bonita y natural que puede ser de agradecer para el usuario que la quiera utilizar.

La segunda interfaz desarrollada es una interfaz mas simple, menos cargada y que muestra todos los datos de una forma mas practica e inmediata que la interfaz anterior.

Es por ello que para la versión por defecto se utiliza esta interfaz que permite a los developers poder analizar y controlar la fuente de una forma mas practica y rápida.

6.1.3 Kmtronic

Los dispositivos Kmtronic son herramientas esenciales en algunos proyectos y tests, especialmente cuando se requiere activar o desactivar señales específicas. A continuación, se explica en detalle qué es un dispositivo KMtronic, cómo funciona y cuál es su papel en el contexto de los tests.

Figura 8. Dispositivo kmtronic RS485 Relay.

Un KMtronic es un dispositivo que incluye un conjunto de relés controlables a través de una conexión USB. Estos relés permiten activar o desactivar salidas específicas, lo que es fundamental para la automatización de ciertas tareas dentro de los tests y para la configuración inicial de algunos proyectos.

El dispositivo KMtronic funciona mediante la activación o desactivación de sus relés controlables. Cada relé del dispositivo tiene tres conexiones:

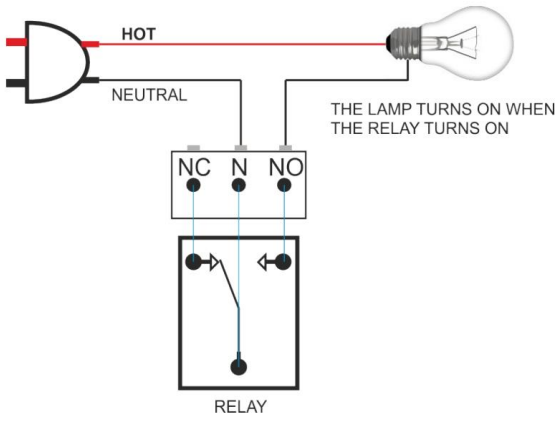


Figura 9. Esquema de conexión de cada relé.

- **NC (Normally Closed)**: Esta es una conexión que está cerrada por defecto. Cuando el relé se activa, esta conexión se desactiva. Es útil para circuitos que necesitan estar cerrados hasta que se indique lo contrario.
- **NO (Normally Open)**: Esta es una conexión que está abierta por defecto. Cuando el relé se activa, esta conexión se cierra. Se utiliza en circuitos que deben permanecer abiertos hasta que se active el relé.
- **N (Neutral)**: Esta es la conexión neutral que puede ser conectada o desconectada de las otras dos conexiones (NC y NO) según las necesidades del proyecto.

6.1.3.1 Función de los KMtronic en los Tests

Dentro de los tests y la configuración de proyectos, los dispositivos KMtronic tienen varias funciones cruciales:

1. **Activación y Desactivación de Señales**: Permiten controlar señales específicas necesarias para la realización de pruebas precisas. Por ejemplo, en la configuración inicial de un proyecto, puede ser necesario activar ciertas salidas para preparar el entorno de pruebas.
2. **Automatización de Tareas**: Facilitan la automatización de tareas repetitivas, lo que reduce el tiempo de configuración manual y minimiza el riesgo de errores humanos.
3. **Control Remoto**: Al ser controlables vía USB, los KMtronic permiten el control remoto de los relés, lo que es particularmente útil en escenarios de pruebas donde los dispositivos no están físicamente accesibles.
4. **Versatilidad en la Configuración**: Los diferentes modos de conexión (NC, NO y N) proporcionan flexibilidad en cómo se pueden configurar y utilizar los relés para cumplir con las necesidades específicas de cada proyecto.

6.1.3.2 KMTRONIC Main part

Esta es una de las librerías mas fáciles de implementar porque para poder controlar cada uno de los relees solo hay que enviar 3 bytes por el serial port.

El primer byte siempre es **FF** por la documentación del dispositivo.

El segundo byte es el numero del relee que se quiere controlar.

El tercer y último byte es el estado que se quiere poner la salida.

A continuación, se muestra una tabla con todos los posibles comandos que se pueden enviar.

No.	Command	Format	Description
1	FF0101	ASCII	Turn ON relay 1
2	FF0201	ASCII	Turn ON relay 2
3	FF0301	ASCII	Turn ON relay 3
4	FF0401	ASCII	Turn ON relay 4
5	FF0501	ASCII	Turn ON relay 5
6	FF0601	ASCII	Turn ON relay 6
7	FF0701	ASCII	Turn ON relay 7
8	FF0801	ASCII	Turn ON relay 8
9	FF0100	ASCII	Turn OFF relay 1
10	FF0200	ASCII	Turn OFF relay 2
11	FF0300	ASCII	Turn OFF relay 3
12	FF0400	ASCII	Turn OFF relay 4
13	FF0500	ASCII	Turn OFF relay 5
14	FF0600	ASCII	Turn OFF relay 6
15	FF0700	ASCII	Turn OFF relay 7
16	FF0800	ASCII	Turn OFF relay 8
17	FFE0xx	ASCII	xx - HEX representation of relays state MSB - relay8 LSB - relay1 **Example:** xx = A6; //0xA6 HEX 0xA6 as binary: 10100110 => MSB LSB 1 0 1 0 0 1 1 0 r8 r7 r6 r5 r4 r3 r2 r1 on off on off off on on off
18	FF0000	ASCII	Return Status of Relays – xxxxxxxx First digit(MSB) is status of Relay1, last(LSB) – status of Relay8 **Example:** Board returns in ASCII 10100110 => MSB LSB 1 0 1 0 0 1 1 0 r8 r7 r6 r5 r4 r3 r2 r1 on off on off off on on off

Figura 10. Tabla de comandos kmtronic.

Por último, lo que se hace en la librería para poder asegurarse que la salida esta en el estado esperado es enviar un segundo comando para poder obtener el estado actual de la salida, de esta forma se sabe si se había enviado correctamente el primer comando.

6.1.3.3 KMTRONIC Test

Para garantizar la fiabilidad y robustez de la librería del KMTRONIC, se ha diseñado un plan de testeo exhaustivo que se divide en dos partes principales. Esta estructura asegura que abarquemos tanto las pruebas individuales de cada relé como las situaciones más complejas que podrían surgir en el controlador del KMTRONIC.

Por una parte están las pruebas específicas para cada relé del dispositivo KMTRONIC, verificando su comportamiento en todos los estados posibles.

- **Pruebas por Relé:**
    - **Activación y Desactivación:** Comprobamos que cada relé pueda activarse y desactivarse correctamente.
    - **Respuesta a Comandos:** Aseguramos que cada relé responde adecuadamente a los comandos enviados a través de la conexión USB.
    - **Estabilidad del Estado:** Verificamos que, una vez cambiado el estado del relé, este mantiene su estado hasta que reciba un nuevo comando.

La segunda fase se enfoca en el KmtronicController, que gestiona el grupo de relés del Kmtronic. Aquí se evaluan todas las posibles situaciones para asegurar que la librería esté preparada para manejar cualquier parámetro o comportamiento, tanto correctos como incorrectos.

1. **Verificación de Situaciones:**
    - **Configuración Inicial:** Validamos que el controlador puede inicializarse correctamente con diferentes configuraciones.
    - **Secuencias de Comandos:** Probamos secuencias de activación y desactivación de múltiples relés para verificar que el controlador puede gestionar varias operaciones simultáneamente.
    - **Gestión de Errores:** Introducimos parámetros incorrectos y simulamos fallos para asegurar que el controlador maneja los errores adecuadamente sin comprometer la estabilidad del sistema.

6.1.3.4 KMTRONIC GUI

Para esta librería se ha diseñado una interfaz visual que sirva a los desarrolladores poder activar y desactivar salidas de una forma rápida y ágil.

Esta interfaz visual esta ideado para poder configurar varios kmtronics y poderles asignar nombre a todos los parámetros para poder saber cada cosa a que corresponde solo echando un vistazo a la pantalla.

6.1.4 CANoe

6.1.4.1 ¿Qué es Vector CANoe?

Vector CANoe es una herramienta de desarrollo y prueba altamente sofisticada utilizada en el sector de la automoción para la simulación, análisis, prueba y diagnóstico de redes de comunicación en vehículos. Estas redes suelen incluir protocolos como CAN (Controller Area Network), LIN (Local Interconnect Network), FlexRay, Ethernet, y otros. CANoe, desarrollado por Vector Informatik, se ha convertido en un estándar en la industria automotriz para el desarrollo y la validación de sistemas electrónicos en vehículos.

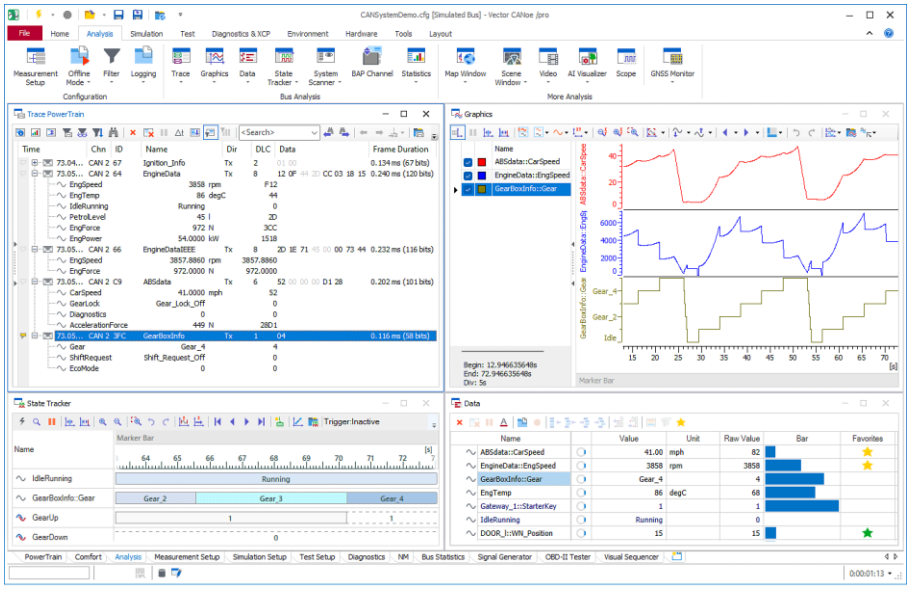


Figura 11. Ventana del programa Vector CANoe.

6.1.4.2 ¿Para Qué se Utiliza CANoe?

1. **Simulación de Redes**:
   - CANoe permite a los ingenieros simular redes de comunicación vehicular completas, facilitando el desarrollo y la validación de unidades de control electrónico (ECUs) y otros componentes antes de que el hardware físico esté disponible.
2. **Análisis y Diagnóstico**:
   - La herramienta proporciona capacidades avanzadas para capturar, visualizar y analizar el tráfico de la red en tiempo real. Esto es esencial para identificar y solucionar problemas en la comunicación entre diferentes ECUs.
3. **Pruebas y Validación**:
   - CANoe permite la realización de pruebas automatizadas y manuales de sistemas electrónicos. Los ingenieros pueden definir casos de prueba específicos para verificar que los componentes cumplen con los requisitos funcionales y de comunicación.
4. **Integración de Sistemas**:
   - La herramienta facilita la integración de múltiples componentes y sistemas, asegurando que funcionen correctamente en conjunto dentro del vehículo.

En el contexto de un proyecto específico, CANoe facilita el envío de tramas con parámetros definidos, lo que es crucial para testear y validar el comportamiento de placas electrónicas y asegurar su correcto funcionamiento dentro del sistema del vehículo. Esta capacidad de configurar, enviar y analizar tramas detalladamente hace de CANoe una herramienta indispensable para el sector del automóvil.

6.1.4.3 CANoe Test

Los tests de esta librería consisten en un test por cada una de las funciones que contiene la librería, por lo que se ha podido comprobar el correcto funcionamiento de todas las funciones de la librería.

Para poder ejecutar los tests se necesita un proyecto que abrir dentro del programa CANoe, es por ello que ha hecho un proyecto de ejemplo que contiene la propia librería para poder ejecutar los tests sin tener que copiar el proyecto de un ordenador a otro.

6.1.5 CAN

CAN (Controller Area Network) es un protocolo de comunicación en serie diseñado para que los microcontroladores y dispositivos puedan comunicarse entre sí sin necesidad de una computadora central. Fue creado por Bosch en la década de 1980 y hoy es un estándar internacional (ISO 11898). Es esencial en la automoción, ya que permite que diferentes componentes de un vehículo se comuniquen de manera eficiente y fiable.

6.1.5.1 ¿Cómo Funciona el CAN?

El CAN utiliza dos cables, CAN_H (alta) y CAN_L (baja), para transmitir datos. Los dispositivos conectados al bus pueden enviar y recibir mensajes. Cada mensaje incluye un identificador que establece la prioridad del mismo, lo que es crucial en situaciones donde dos dispositivos intentan comunicarse al mismo tiempo.

1. **Transmisión y Recepción**: Los nodos en el bus CAN pueden tanto enviar como recibir mensajes. Al transmitir, un nodo envía un mensaje con un identificador único.
2. **Arbitraje**: Si dos nodos transmiten al mismo tiempo, el bus CAN usa un método de arbitraje basado en la prioridad del identificador del mensaje para decidir cuál se transmitirá primero.
3. **Detección de Errores**: CAN tiene mecanismos integrados para detectar errores durante la transmisión y recepción de datos, asegurando la integridad de la comunicación.

6.1.5.2 Importancia del CAN en la Automoción

En la industria automotriz, el bus CAN es vital debido a su robustez y eficiencia. Permite que diversos módulos electrónicos del vehículo, como el motor, la transmisión, los frenos antibloqueo (ABS), airbags y sistemas de entretenimiento, se comuniquen de manera rápida y fiable. Esto asegura que todas las partes del vehículo trabajen juntas de manera armoniosa y segura.

6.1.5.3 Uso del CAN para Testing en Proyectos de Automoción

Para probar las placas electrónicas en el sector automotriz, se utiliza el bus CAN para verificar que las unidades de control electrónico (ECUs) y otros componentes funcionen correctamente. Durante estas pruebas se realizan las siguientes actividades:

1. **Envío y Recepción de Tramas**: Se envían mensajes CAN a las ECUs y se verifica su respuesta para asegurar que los comandos y datos se procesen correctamente.
2. **Simulación de Condiciones**: Se simulan diferentes condiciones de operación del vehículo y se observan las respuestas del sistema.
3. **Detección de Errores**: Se monitorizan los mensajes en el bus CAN para identificar posibles errores de comunicación.

6.1.5.4 Uso de CANoe y Librerías CAN

CANoe es una herramienta desarrollada por Vector Informatik que permite simular, analizar y probar redes CAN. Aunque es muy potente, requiere una licencia para su uso completo.

Sin embargo, existen librerías de CAN que permiten enviar y recibir tramas CAN sin necesidad de herramientas costosas. Para utilizarlas, solo necesitas un dispositivo compatible. Ese es el objetivo de esta librería, poder enviar tramas sin tener una licencia de CANoe.

6.1.5.5 Dispositivo de Testing: Vector VN1604

En los proyectos de automoción, un dispositivo común para realizar pruebas es el **Vector VN1630**. Este hardware permite interactuar directamente con el bus CAN, facilitando el envío y recepción de tramas para verificar el funcionamiento correcto de los sistemas.

Figura 12. VN1630A CAN/LIN Interface.

6.1.5.6 CANCase

La librería encargada de la comunicación CAN ha sido llamada CANCase porque su principal funcionamiento es interactuando mediante una CANCase de Vector, por lo que de esta forma el nombre ayuda a recordar para lo que sirve y no con el nombre de CAN que se puede confundir con una librería mas abstracta y con diferente funcionalidad.

Esta librería esta se plantea para poder ser utilizada en cualquier proyecto que se quiera testear mediante dispositivos CANCase, por ello se ha desarrollado para que con esta se pueda configurar y usar diversos dispositivos de forma simultánea.

Este proceso de inicializar y configurar los canales se realiza utilizando la librería propia que proporciona vector para poder utilizar las interfaces del bus Vector de manera eficaz y potente en aplicaciones personalizadas.

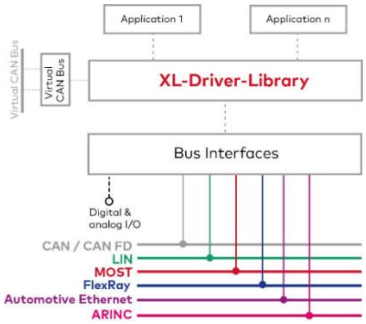


Figura 13. Interacción XL-Driver-Library a nivel aplicación.

6.1.5.7 CANCase GUI

Para esta librería se ha desarrollado una interfaz grafica que utiliza la parte principal, anteriormente descrita, para poder configurar, enviar y recibir tramas CAN de una forma muy sencilla y visual.

Es una forma muy útil para los desarrolladores de poder hacer pruebas de una forma muy ágil y que me ha servido mucho para el desarrollo de la librería poder comprobar su correcto funcionamiento.

En la interfaz visual se ha buscado un aspecto atractivo y agradable a la vista, poder tener botones fácilmente distinguibles y que sean totalmente intuitivo de utilizar.

6.2 TestBench

El testbench es una de las partes más importantes del framework de testing, funcionando como el cerebro del sistema. Actúa como intermediario entre las funciones de los tests y las librerías, gestionando todas las acciones, errores y situaciones para asegurar que los tests se ejecuten correctamente y evitar problemas no deseados.

El testbench también se encarga de preparar el entorno del proyecto. A partir de la configuración proporcionada, realiza todas las acciones necesarias para inicializar los dispositivos y asegurar que todo esté listo para ejecutar los tests con éxito. Esto incluye:

- **Configuración del Proyecto**: Ajustar y aplicar la configuración específica del proyecto para que los dispositivos funcionen correctamente.
- **Inicialización de Dispositivos**: Encender y preparar todos los dispositivos necesarios para los tests.
- **Gestión de Ejecución**: Controlar la ejecución de los tests, asegurando que cada uno se realice en el orden correcto y que los resultados se registren adecuadamente.
- **Manejo de Errores**: Detectar y gestionar cualquier error o problema que surja durante la ejecución de los tests, garantizando que el proceso continúe de manera estable.

6.2.1 TestBench Main Part

6.2.1.1 TestBench Controllers

Uno de los objetivos principales del proyecto es que pueda ser utilizado para cualquier proyecto, por lo que el testbench debe estar preparado para tener diferentes configuraciones de diferentes dispositivos, los encargados de esto son los denominados **controllers**, hay uno por cada una de las librerías desarrolladas.

La estrategia de los controllers es que puedan soportar un gran numero de dispositivos configurados y debe poder gestionar su correcto funcionamiento, a continuación se muestra un esquema del caso general que se pretende conseguir.

6.2.2 TestBench CLI

Se ha desarrollado una versión CLI (Interfaz de línea de comandos) por tal de poder realizar las funciones del TestBench utilizando comandos, esto puede permitir a muchos desarrolladores poder hacer algunas acciones del TestBench de una forma rápida utilizando una linea de comandos.

Otra cosa que permite el TestBench CLI es el poder utilizar el testBench desde otro programa totalmente diferente o poder utilizar Scripts (secuencia de comandos) y automatizar algunos procesos que deseen.

6.2.3 TestBench GUI

El TestBench tiene una interfaz visual que permite tener desde una única ventana o panel todos los dispositivos que se utilizan en un proyecto para hacer los tests, de esta forma se puede controlar todo el sistema y permite que un ingeniero pueda montar o probar de una forma más fácil y sencilla.

Esta interfaz visual se ha desarrollado a partir de las interfaces visuales de las librerías, agrupándolas en una única venta y aplicación.

Una de las grandes funcionalidades que tiene este programa, es que **la interfaz visual es generada a partir del fichero de configuración,** lo que permite cambiar la interfaz visual de un proyecto a otro solo cambiando los parámetros del fichero de configuración.

Esto permite a un ingeniero que cambiar rápidamente de un proyecto a otro solo cambiando el fichero que se utiliza para ser generado, por lo que no se tiene que programar una interfaz visual para cada proyecto.

6.3 AssemblyMethods

El componente assemblyMethods es crucial dentro del framework de pruebas, ya que define el flujo de acciones necesarias para preparar y ejecutar los tests. A continuación, se detalla todo lo que hace este componente, junto con el orden en que se ejecutan las tareas.

Este flujo de trabajo detallado asegura que todos los componentes necesarios están configurados y funcionando correctamente antes de proceder con las pruebas específicas, proporcionando un entorno de pruebas robusto y fiable.

6.3.1 Runsettings

El runsettings es el fichero en que se especifica que instrumentos o dispositivos se utilizan y donde están conectados o a que están asignados. Con este fichero el testing framework ya se encarga de hacer o ejecutar lo necesario para poner todo lo que se especifica listo para poder ejecutar los tests.

Para que este fichero tenga un único formato estándar y que todo se especifique de una misma forma se ha detallado la siguiente forma de escribir los parámetros necesarios.

6.4 Documentación

La documentación es una parte imprescindible de cualquier proyecto, pero en este caso aun mas pues para que los usuarios sepan como utilizarlo, las funcionalidades, que partes poder tocar y que otras partes no, los posibles errores, la enumeración de clases y para que sirven etc.

Es por ello que en las fases iniciales del proyecto se experimentó que a la hora de documentar la parte de funciones y métodos que tenia una librería, era una tarea muy repetitiva y que en cierta parte ya se había realizado cuando en el código ya había comentado el funcionamiento. De esta forma se propuso y se desarrolló una herramienta que permite, a partir de los ficheros de un proyecto realizado en Visual Studio, generar la documentación de dicho proyecto

Esta herramienta llamada **DocXLEARGen** permite explotar una funcionalidad que tiene Visual Studio que a la hora de compllicar este genera un fichero XML con todas las funciones, clases y tipos que tiene ese proyecto con todos los comentarios. **DocXLEARGen** lo que hace es ir leyendo ese fichero XML, cuando encuentra un elemento del tipo que le interesa, este lo trata y lo copia en un nuevo fichero de tipo **Markdown** (es un tipo de fichero utilizado para documentar en repositorios), finalmente a partir de este **Markdown** genera un documento **PDF** con un formato totalmente entregable y con las señas de lear con todo lo que contenía el documento de Markdown anteriormente mencionado.

A Continuación se muestran algunos ejemplos generados.

7 Conclusiones

El proyecto de Máster en Ingeniería Industrial realizado en LEAR Corporation ha sido una experiencia sumamente enriquecedora. Durante este tiempo, he adquirido un conocimiento profundo sobre el funcionamiento de los proyectos en el sector de la automoción y he tenido la oportunidad de familiarizarme con los métodos y técnicas de prueba utilizando diversos instrumentos.

A lo largo del proyecto, se cumplieron casi todos los objetivos planteados al inicio. Sin embargo, la falta de tiempo en la última etapa impidió una documentación tan exhaustiva como hubiese sido deseable. También me hubiera gustado tener más tiempo para probar el conjunto del framework de pruebas en más proyectos, lo que habría permitido obtener más datos sobre su rendimiento y eficacia.

La planificación inicial se siguió casi a la perfección, especialmente en las primeras etapas del proyecto. El desarrollo de las librerías se completó dentro de los tiempos estimados, e incluso algunas librerías se desarrollaron antes de lo previsto. No obstante, en la última fase del trabajo surgieron algunos cambios sustanciales y pequeños errores que debían corregirse, lo que provocó desviaciones importantes. Estas desviaciones impidieron comprobar completamente el funcionamiento del sistema y dejaron poco tiempo para realizar toda la documentación necesaria.

Uno de los grandes retos fue anticipar todas las posibles situaciones que podrían ocurrir, incluso aquellas que parecían irreales. Diseñar un framework que maneje cada una de estas situaciones de manera segura y controlada fue una tarea compleja. Aunque puede parecer sencillo pensar en posibles escenarios, la realidad de las pruebas reveló muchas más posibilidades que inicialmente no se consideraron. Plantear cómo debería actuar el sistema ante cada posibilidad fue uno de los grandes desafíos de este trabajo.

Se pudieron probar cada una de las partes del sistema por separado, así como el framework de pruebas completo en diferentes situaciones y en dos proyectos de la empresa. Este proceso permitió verificar el cumplimiento y la eficacia de los objetivos iniciales del proyecto.

El proyecto ha sido extremadamente valioso para mí, proporcionándome un aprendizaje significativo sobre los aspectos técnicos y operativos del sector automotriz. Además, ha facilitado mi inserción en el mundo laboral. Aunque hubo algunas dificultades y áreas que no se pudieron explorar completamente debido a limitaciones de tiempo, el proyecto en general cumplió con los objetivos iniciales y proporcionó resultados positivos tanto para mí como para la empresa.

Este trabajo no solo ha mejorado mis habilidades técnicas, sino que también ha reforzado mi capacidad para planificar, ejecutar y documentar proyectos complejos en un entorno industrial real. La experiencia en LEAR Corporation ha sido, sin duda, un paso crucial en mi desarrollo profesional y me ha preparado para enfrentar futuros desafíos en mi carrera en la ingeniería industrial.

8 Referencias

[1] Lauterbach, «PowerView User's Guide,» Febrero 2024. [En línea]. Available: https://www2.lauterbach.com/pdf/ide_user.pdf.

[2] Lauterbach GmbH, «TriCore™ Debugger & Trace,» [En línea]. Available: https://www.lauterbach.com/supported-platforms/architectures/tricore. [Último acceso: 2024].

[3] © Microsoft 2024, «What is Azure Devops,» [En línea]. Available: https://learn.microsoft.com/es-es/azure/devops/user-guide/what-is-azure-devops?view=azure-devops. [Último acceso: 2024].

[4] Vector Informatik GmbH, «XL-Driver-Library,» [En línea]. Available: https://www.vector.com/int/en/products/products-a-z/libraries-drivers/xl-driver-library/#c75439. [Último acceso: 2024].

[5] Lauterbach GmbH, «PowerDebug E40,» [En línea]. Available: https://www.lauterbach.com/products/debugger/powerdebug-system/powerdebug-e40.

[6] EA Elektro-Automatik GmbH, «Operating Guide PS 9000 2U DC Laboratory Power Supply,» [En línea]. Available: https://divisoft.se/media/manuals/06230204_EN.pdf.

[7] Lear Corportation, «Pàgina web de Lear Corportation,» [En línea]. Available: http://www.lear.com/.

[8] SmartBear Software, «Gherkin Reference,» [En línea]. Available: https://cucumber.io/docs/gherkin/reference/.

[9] Automotive Vehicle TestingAutomotive Vehicle Testing, «ISO 15765-2 Protocol or CAN-TP or DoCAN Explained,» [En línea]. Available: https://automotivevehicletesting.com/vehicle-diagnostics/uds-protocol/iso-15765-2-protocol/.

[10] CSS Electronics, «CAN DBC File Explained,» [En línea]. Available: https://www.csselectronics.com/pages/can-dbc-file-database-intro.

[11] Keysight Technologies, «SCPI Basics,» [En línea]. Available: https://helpfiles.keysight.com/csg/n5106a/scpi_basics.htm.

[12] EA Elektro-Automatik, «an005 SCPI Command list overview,» [En línea]. Available: https://elektroautomatik.com/wp-content/uploads/an005_scpi_command_list_overview_en.pdf.