Escuela Politécnica Superior de Jaén
Grado en Ingeniería Informática

Universidad de Jaén
Escuela Politécnica Superior
de Jaén

TRABAJO FIN DE GRADO

SOFTWARE DE PRUEBAS PARA CAMA DE PINCHOS

Alumno

Carlos Borrás Rubio

Tutor

Jose María Serrano Chica
(Departamento de Informática)

11, 2021

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Don Jose María Serrano Chica, tutor del Trabajo Fin de Grado titulado: '**Software de pruebas Para Cama de Pinchos**', que presenta Don Carlos Borrás Rubio, otorga el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, [mes] de [año]

El alumno: El tutor:

Carlos Borrás Rubio Jose María Serrano Chica

(Página intencionalmente en blanco)

Agradecimientos

Este trabajo se lo dedico a mi familia que siempre me apoya en lo que hago, sobretodo mis padres y mi hermano, que vigilan que no me descalabre. A mis compañeros de carrera Pedro Luis y Jose Ángel, que me dieron una vida universitaria plena. A mis compañeros de trabajo que me ayudaron a integrarme en el espacio laboral y me dieron muchísimas ideas mientras me guiaban en el mundo laboral de la ingeniería. A mi amigo Fran que siempre está encima de mí para que termine este trabajo y sobretodo este parrafo para ponerme en serio. Ah bueno, no olvidar a mi tutor que me permitió una gran libertad en este trabajo y a todos los profesores de la carrera que me dieron la mayor parte de los conocimientos que hoy día pongo en práctica.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Especialidad (solo TFG)	Sin especialidad
Mención (solo TFG)	Sin mención
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	Carlos Borrás Rubio
Fecha de asignación	15/10/2021
Descripción corta	Las placas de prueba para circuitos integrados (PCB test fixture), o “camas de pinchos”, como se suelen denominar en el ámbito industrial, constituyen una herramienta esencial para comprobar el buen funcionamiento de microcontroladores durante el proceso de su fabricación. Generalmente, este proceso de prueba puede automatizarse por medio de diferentes soluciones software. En este trabajo fin de grado, se propone el diseño e implementación de una herramienta software para la programación de un microcontrolador y la automatización de los procesos de pruebas semifuncionales sobre el mismo, optimizando dicho software para una máxima usabilidad y eficiencia en su aplicación, medida como el número circuitos impresos testados en un determinado espacio de tiempo.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Contenido

Índice de ilustraciones

Índice de tablas

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en el apartado 6 (Definiciones y abreviaturas).

1.1 Introducción

A su atención como lector de este trabajo de fin de grado se hablará sin nombres de clientes ni de la empresa que gestiona el proyecto, tampoco es posible incluir la documentación de la oferta ni la guía de pruebas que más adelante será mencionada por confidencialidad.

Hoy día la electrónica y los microcontroladores inundan el mercado y la industria, de hecho, en el tiempo en el que se escribe este trabajo hay una crisis en el silicio que provoca grandes retrasos en distintas industrias que hoy día instalan estos microcontroladores como la automovilística, que se ve obligada a cerrar algunas de sus fábricas debido a esta escasez.

Por otro lado, la "calidad" como departamento y como fase, en las líneas de producción aumenta su importancia cada vez más, debido a la automatización de estos puestos, se han de efectuar distintas utilerías para encomiar estas tareas, como no puede ser de otra manera, para abaratar costes a la larga y no requerir de usuarios expertos o trabajadores experimentados y que requieren un coste adicional mes a mes a la empresa, estas optan por soluciones que faciliten la tarea lo máximo hasta el punto de requerir 2 o 3 pasos clave para el usuario/trabajador.

Estas soluciones como veníamos explicando constan de automatismos ya bien sean por parte de dispositivos, de software o de una combinación de ambos, como es la mayoría de los casos. De este trabajo se podrá sacar en claro al final del mismo, la clara ventaja que tiene invertir un dinero en una solución específica a invertirlo en trabajadores experimentados, que para una tarea como esta puede ser automática y que termina requiriendo de un conocimiento suficiente. Para ello se

tomará de referencia un proyecto de la empresa para la que he trabajado, con ello resumiré los costos tanto de diseño, como de investigación (que fueron necesarios para construir el software), en especial la parte software a la que debo atenerme por el objetivo del trabajo.

1.2 Objetivos del trabajo

El objetivo del trabajo propiamente dicho será que el software programe un microcontrolador y que efectúe pruebas semifuncionales sobre este, con el aliciente de que el software sea lo más simple posible de cara al usuario final, el trabajador de una línea de producción en la fase de calidad, que su instrucción no dure más de 1 hora hasta que el usuario sepa cómo llevar cualquiera de las situaciones que se pueden dar, además el software, tiene que llevar a cabo las tareas en un corto periodo de tiempo ya que prima que se puedan comprobar el máximo número de tarjetas de circuito impreso (*"PCBs"* como nos referiremos a partir de ahora) en la jornada.

Esta es la tarea principal encomendada al proyecto por parte del cliente que contrató los servicios, no más no menos. Con la oferta de este trabajo además se incluye una guía para efectuar las pruebas anteriormente expuestas. Las pruebas semifuncionales constan de aplicar voltajes en ciertas zonas del circuito de una PCB y medir voltajes en otras zonas comprobando que se cumplen unos umbrales.

1.3 Antecedentes y estado del arte

En otras ocasiones en esta empresa ya se hicieron proyectos de esta índole, aunque estaban separadas las tareas de programación del microcontrolador y la de las pruebas semifuncionales/funcionales, por lo que se tiene el conocimiento de que el trabajo que se desempeña en la fase de calidad de las cadenas de producción ya se efectuaba anteriormente con distintos tipos de “camas de pinchos” como son llamadas popularmente en este ámbito.

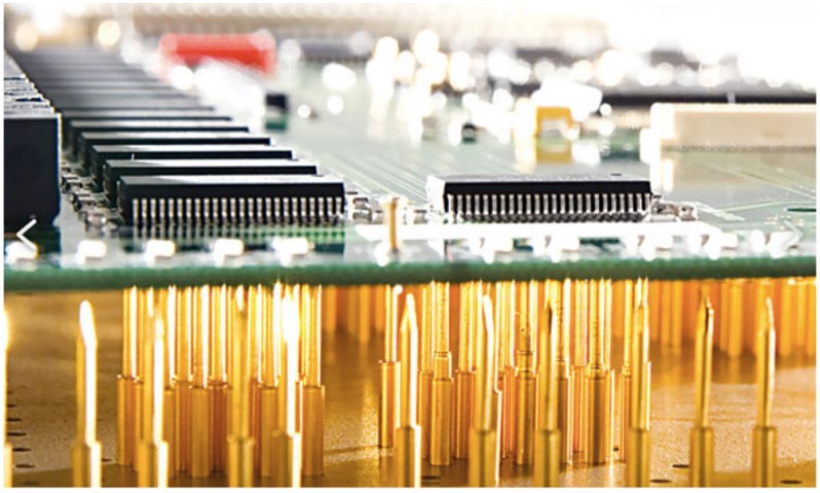


Ilustración 1 .- Cama de pinchos de cerca

Estas camas de pinchos, son utilizadas para acceder a distintos puntos de las PCBs eléctricamente, como si de las agujas de un polímetro se tratara (como puede verse en la imagen de arriba), para sacar distintas mediciones o, como anteriormente se explicaba, para inducir algún voltaje y llevar a cabo pruebas. Además, estos puntos también son utilizados para programar los microcontroladores que se instalan sobre las PCBs. Pueden encontrarse distintos tipos de estas camas de pinchos ya que no siempre cumplen una función meramente electrónica, a veces también se requieren comprobaciones a nivel lumínico. Las camas de pinchos más simples sólo son usadas para programar los microcontroladores de las PCBs mientras que las más complejas son usadas para

Además de esta solución para la comprobación semifuncional/funcional de PCBs, ahora se encuentran nuevos puestos mucho más costosos pero con muchísimas más funcionalidades, dedicadas a encontrar desperfectos en la manufacturación de los dispositivos que se producen en las cadenas de producción de PCBs, estos son los puestos de test de sondas móviles. Unos equipos

semejantes a armarios en los que unas agujas semejantes a las usadas en las camas de pinchos tradicionales, pero que por el contrario se mueven a través de las obleas, pudiendo comprobar una oblea de PCBs en pocos segundos. Sin embargo estos equipos pueden llegar a costar decenas de miles de euros y por ello las empresas que manufacturan PCBs suelen tener alguna de estas máquinas y pedir camas de pinchos para otras, dependiendo de los volúmenes de producción en los que se trabaje según la PCB se rentabiliza más la cama de pinchos tradicional o la maquinaria de sondas móviles en estos trabajos.

Las ventajas de una máquina de sondas móviles sobre una cama de pinchos tradicional son varias, entre ellas la automatización es un punto fuerte ya que estas máquinas es fácil incluirlas en líneas de producción, por lo que finalmente en placas PCB simples, cuando aún están en un formato de oblea, si el volumen de producción es suficiente es mejor usar este tipo de máquinas. Además, aunque anteriormente no se ha mencionado, estas máquinas tan costosas tienen como principal funcionalidad su versatilidad, puesto que las pruebas pueden implementarse para distintos tipos de PCB (sobre todo en su formato de oblea), es decir, una misma máquina puede ser usada en distintas líneas de producción. El principal inconveniente de estas como veníamos diciendo es que son muy costosas, además de que hay tareas que no son capaces de llevar a cabo, como son pruebas semifuncionales/funcionales (algunas de las pruebas semifuncionales serían capaces de llevarlas a cabo, pero no es muy recomendable debido a la limitación en el número de sondas, algunas de las pruebas pueden necesitar de 5 a 6 puntos de test). (Universidad de Jaén, 2004)[1]

Por otro lado, las camas de pinchos tradicionales, pueden tomar múltiples formas debido a las pruebas que se pueden realizar en estas, podemos encontrar camas de pinchos en las que sólo haga falta la interfaz de “pinchos” a los puntos de test, pero a partir de ahí existe variedad de sensores que se pueden usar, desde cámaras y sensores de color a útiles de fibra óptica para observar color e intensidad de luces en campo visual, también pueden ser necesarias tarjetas de comunicaciones si lo que se va a probar está relacionado con ello, como puertos serie o comunicaciones TCP/IP por ejemplo. Estas últimas funciones serían para

[1] SISTEMA ROBÓTICO DE INSPECCIÓN AUTOMÁTICA PARA PCBs BASADO EN VISIÓN POR COMPUTADOR Y SONDAS MÓVILES (14/10/2021) - https://intranet.ceautomatica.es/old/actividades/jornadas/XXV/documentos/146-uanesjaezo.pdf

hacer pruebas funcionales, pero en este trabajo nos ceñiremos en pruebas semifuncionales.

Para dar contexto, una prueba semifuncional es una prueba que se basa en la prueba fuera del contexto completo del dispositivo, es decir, fuera de su ecosistema. En este contexto, sería coger una PCB como la nuestra y hacer pruebas de funcionalidad eléctrica sin tener el equipo completo, con la cama de pinchos inducimos corrientes eléctricas y hacemos pequeños cortocircuitos donde se supone que interactúa con otros dispositivos simulando estos mismos. Si esta PCB fuera a probar un pequeño motor eléctrico simularíamos las señales eléctricas que actúan sobre el conector que utiliza como nexo.

1.4 Descripción de la situación de partida

Como punto de partida hay una oferta de proyecto en la que se demanda una cama de pinchos la cual sea capaz de programar un microcontrolador y a posteriori hacer unas pruebas semifuncionales sobre este, las cuales vienen dadas con la oferta.

Se decide tomar una decisión y usar dispositivos de “bajo coste” así como una aplicación JAVA y así producir una cama de pinchos como sistema embebido que lleve este programa.

1.4.1 Descripción del entorno actual

Antes de efectuar este trabajo nos encontramos con la necesidad de suplir una fase en la producción de un nuevo dispositivo, su calidad. Para que un dispositivo llegue a manos del cliente final, primero ha de pasar unas pruebas de funcionamiento. Sin el equipo que debe usarse para este cometido, la fase de calidad podría realizarse manualmente, pero es muchísimo más ineficiente y la empresa no podría beneficiarse de ningún modo. Aunque sea la fase de calidad hay que tener en cuenta que es tan importante como todas las anteriores, es el culmen y el punto en el que se da el visto bueno a un producto. Sin esta, el producto podría venderse, pero ante cualquier problema con el dispositivo manufacturado, el productor del mismo podría tener que indemnizar a su cliente lo que volvería a producir costes por encima del beneficio sin contar por supuesto la pérdida de confianza con el mismo.

Por lo tanto, partimos de la base de que este dispositivo es necesario y da un valor añadido al producto que se manufactura, un sello de calidad que indica que el producto ha sido probado y funciona una vez sale de la fábrica, una tarea que lleva años existiendo en el entorno industrial.

1.4.2 Resumen de las deficiencias y carencias identificadas

Como se ha mencionado en el apartado anterior, es necesaria la inclusión de un dispositivo de pruebas que, además, pueda llevar un registro de las placas que han sido testadas y del resultado de la prueba, en definición, dar soporte a la trazabilidad de las PCBs testadas.

1.5 Requisitos iniciales

El software final debe ser capaz de:

- Programar un microcontrolador mediante el dispositivo "*JLink BASE", dispositivo del que hablaremos más adelante.*
- Corroborar que se ha programado correctamente.
- Corroborar que el microcontrolador tiene asignado un ID interno y es legible por el dispositivo.
- Efectuar pruebas semifuncionales de la PCB a testar. Las 10 pruebas incluidas en la guía anteriormente mencionada.
- Ofrecer un sistema de trazabilidad para las PCBs testadas.

El software final debería ser capaz de:

- Ofrecer la posibilidad de programar solamente o, pasar una de las pruebas (sin necesidad de trazabilidad).

1.6 Alcance

El trabajo engloba la parte de investigación, implementación e instrucción del proyecto, es decir, desde se comienza a trabajar en el proyecto hasta que el mismo se pone en marcha con el cliente, verificando así que el punto de que el dispositivo sea "amigable" y sencillo para el usuario final se lleva a cabo.

Los entregables por lo tanto serán:

- Este mismo documento.

- El código de la aplicación JAVA.
- El manual de usuario proporcionado al cliente.
- Códigos anexos a la aplicación.

1.7 Hipótesis y restricciones

Una de las exigencias es que cada iteración de la máquina, cada test que se lleve a cabo, sea lo más rápido posible, por lo que cualquier parte del dispositivo, bien sea hardware o software, ha de ser cambiado por algún medio que beneficie esta restricción. Otra de las restricciones como vimos anteriormente, es abaratar los costes de un dispositivo hecho con LabVIEW y elementos de National Instruments, para lo que llevaremos un registro del costo del dispositivo final y licencias de cada modelo.

El TFT se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

1.8 Estudio de alternativas y viabilidad

Existen diversos lenguajes que se pueden utilizar para proyectos de ésta índole entre ellos: JAVA, C++, Python, LabVIEW.

A día de hoy JAVA es utilizado en multitud de dispositivos, uno de los lenguajes de programación más populares. Hace uso de una máquina virtual en la que corren sus programas el conocido JVM, lo cual lo hace algo más lento que otros lenguajes de programación conocidos. Además este lenguaje permite facilidad en la creación de prototipos.

C++ es un lenguaje de programación bastante extendido en la comunidad, debido a ello, al igual que JAVA, hay una gran cantidad de documentación disponible y de librerías que permiten implementaciones de gran envergadura. Ofrece herramientas para manejo de memoria que permite al programador mejorar la eficiencia en el uso de la memoria

Por otro lado, Python es uno de los lenguajes más populares a día de hoy, un lenguaje de código abierto, orientado a objetos, simple y fácil de entender el cuál se

ha convertido en una poderosa alternativa ante C++. Además de ser un lenguaje ideal en muchos campos de programación que a día de hoy están más que nunca en auge, por ejemplo: Big Data, Data Science, Inteligencia Artificial, e incluso en Frameworks de Pruebas. (Soloaga, 2018)[2]

Por último, LabVIEW es una herramienta bastante viable de desarrollo en el entorno en el que se trabaja ya que LabVIEW proporciona un potente entorno de programación que es usado sobre todo en entornos industriales y electrónicos, las principales características que lo definen: su intuitividad como lenguaje de programación y la gran cantidad de herramientas de alto nivel preparadas para el desarrollo de las aplicaciones. (Universidad de Cantabria, 2021)[3]

1.9 Descripción de la solución propuesta

La solución aportada para este proyecto será una aplicación que gestione los dispositivos que serán utilizados para comunicarnos con la PCB y que sea capaz de llevar a cabo las pruebas señalizadas en la guía de pruebas.

Para esto se realizará un estudio de los requisitos y la implementación de la aplicación en un sistema embebido que se incorporará dentro de la cama de pinchos.

1.10 Tecnologías utilizadas

Las tecnologías que principalmente serán usadas son:

- Un lenguaje de programación orientado a objetos.
- Librería de comunicación con PLCs.
- Librería de comunicación serie.
- Librería de comunicación TCP/IP.
- Librerías de hilos y procesos.

[2] Usos de Python (14/10/2021) - https://www.akademus.es/blog/programacion/principales-usos-python/

[3] Software LabVIEW (14/10/2021) - https://sdei.unican.es/Paginas/servicios/software/Labview.aspx

1.11 Metodología de desarrollo de software

[INDICADO CUANDO SE DESARROLLA Y/O SE IMPLANTA UN SOFTWARE O SISTEMA]

La metodología usada en este proyecto es basada en la metodología en cascada, debido a la cantidad de tecnologías que intervienen se vio una metodología de este tipo como la idónea, dando así la posibilidad de proponer unos objetivos muy definidos.

Teniendo en cuenta las distintas metodologías y el tiempo límite del proyecto, sin contar las posibles peticiones del cliente en mitad del mismo, nos da la posibilidad de probar cada una delas tecnologías que intervienen e investigarlas previo desarrollo.

Por una parte, teníamos que probar que todas las partes del dispositivo funcionasen, por lo que, las primeras dos semanas, el objetivo sería probar que todas las tecnologías son válidas dentro del programa y no había ningún problema dentro de estas.

1.12 Análisis de riesgos

Dentro de los riesgos que se podían dar dentro del proyecto, estaba la incompatibilidad de alguna de las tecnologías que intervienen. Si el PLC no era capaz de comunicarse con nuestro sistema la acción llevada a cabo habría sido buscar un PLC que fuera específico para estas tareas, debido a esto podría haberse retrasado el proyecto puesto que habría que invertir tiempo en investigar el nuevo dispositivo. Otro problema podría ser la incompatibilidad de la comunicación TCP/IP con nuestro objetivo de que las pruebas se tenían que llevar a cabo en el menor tiempo posible ya que, en ocasiones, los dispositivos que se controlan a partir de esta tecnología no tienen por qué procesar las peticiones en un tiempo crítico (cuando hablamos de dispositivos comerciales), hablaremos de esto más adelante.

Otro posible punto crítico del proyecto podría ser la equivocación al comprar el dispositivo para la instalación del firmware JLink ya que de estos existen varios tipos, como mencionamos con anterioridad, dentro de la empresa ya se habían hecho otros programas que efectivamente usaban estos dispositivos y cómo veremos más adelante un error de este tipo es posible pero, se puede solventar mediante software.

Hay otro punto crítico a tener en cuenta y es que este tipo de programas están pensados para trabajar como una máquina de estados, un error en esta evidencia puede retrasar el desarrollo, ya que si alguno de estos estados está mal planteado el programa sería propenso a fallar.

Otro tipo de problemas pueden ser los errores de manual de pruebas, estos últimos son los más difíciles de evitar puesto que corresponde al cliente final dar una respuesta rápida y efectiva que solucione estos problemas, como se verá más adelante estos problemas son más habituales de lo que pueda parecer y tienen repercusión en la duración final del trabajo o incluso pueden provocar daños materiales.

1.13 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFT, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados el tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (la persona autora del trabajo).

Para este trabajo se tuvo que tomar dos meses de trabajo, que en horas totales de trabajo se dieron 93 horas, dado que no se trabajaba sólo en este proyecto. Además del tiempo implicado en este documento y el manual de usuario final.

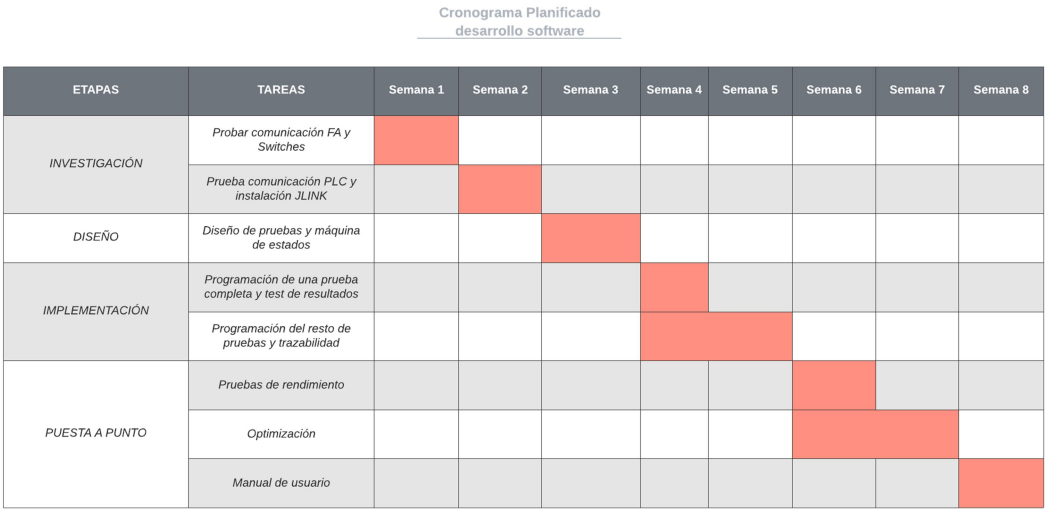
1.14 Planificación temporal

La planificación del proyecto es dividida en 4 etapas que serán divididas en tareas:

- **INVESTIGACIÓN**: Es la etapa inicial del proyecto y se caracteriza por ser la parte más “teórica” de la implementación, tras recoger las distintas tecnologías que se emplearán en el proyecto, se hace la respectiva investigación sobre cómo se pueden implementar y combinar con las herramientas con las que contamos. Se hacen pruebas en un entorno ajeno al final dónde se comprueba el correcto funcionamiento de cada parte por separado.

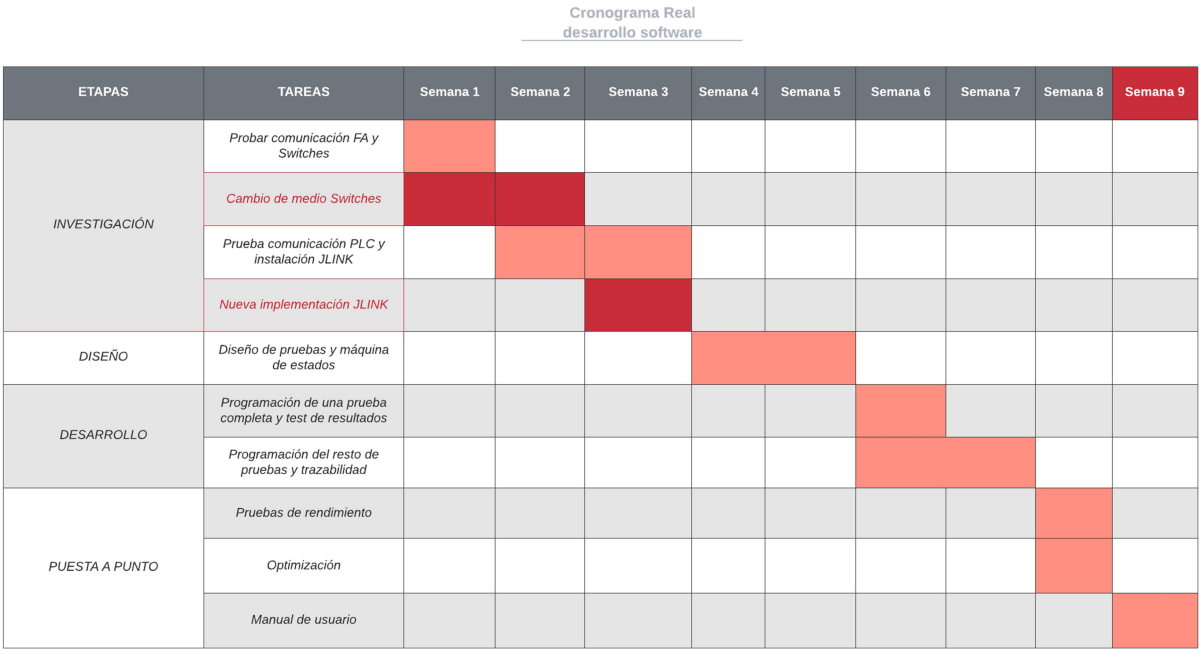
- **DISEÑO**: Una vez la etapa de investigación se ha terminado podemos comenzar con la implementación del "conjunto" de las funcionalidades anteriormente comentadas y se comienza diseñar el "esqueleto" de la aplicación, en principio el diseño de funcionamiento.
- **IMPLEMENTACIÓN**: La etapa de implementación como su nombre indica lleva el grueso del desarrollo de la aplicación en cuanto a las pruebas (el objetivo principal del proyecto) se trata.
- **PUESTA A PUNTO**: En esta última etapa se implementan los últimos rasgos de la aplicación, alguna optimización y los respectivos mecanismos de seguridad. Por último, el manual de usuario queda incluido dentro de esta etapa.

Tabla 1 - Planificación del desarrollo software



En general el objetivo de la planificación es que cada semana hubiera un objetivo al menos que fuera resuelto, pero esto es sólo una planificación y la realidad es bien distinta puesto que como en cualquier trabajo de este tipo ocurren imprevistos que solventar, a continuación se verá el cronograma real del proyecto y se explicará cada uno de los retrasos y cambios de planificación.

Tabla 2 - Cronograma final del proyecto



Como podemos ver en este cronograma aparecen tareas ajenas a las originales, problemas arrastrados al no contar con una perspectiva del software al comienzo del proyecto.

Por último, queda explicar que como podemos ver en el cronograma final, algunas de las tareas como optimización tuvieron que retrasarse en pos de sacar el proyecto a tiempo para el cliente. Dentro de esta tarea no se perdió mucho rendimiento de la aplicación puesto que en su mayoría las tareas más críticas pertenecen a las pruebas en las que se mantuvo la planificación y una estructura previa.

1.15 Presupuesto

El presupuesto de este proyecto está desglosado en los siguientes apartados, siendo los únicos datos cedidos las horas ofertadas de los apartados correspondientes y un total del presupuesto en materiales. Al final de este, se incluirá lo finalmente consumido de la oferta.

Tabla 3 - Presupuestos

Tipo de gasto	Ofertado	Consumido
Materiales (hardware y estructura)	4000€	3961.54€
Horas de ingeniería (Diseño mecánico y eléctrico)	48 horas	60 horas
Horas de software (Diseño e implementación)	80 horas	93 horas
Horas de producción (Montaje)	48 horas	61.25 horas
Horas de gestión (Reuniones con cliente)	12 horas	14 horas
Horas de instalación (Instalación e instrucción sobre el software)	10 horas	9.5 horas

Como podemos ver en algunas categorías se consumieron más horas de las ofertadas, lo que influye negativamente en el coste final del proyecto. Basando las horas en un precio ficticio de 10€/h, se podría decir que este proyecto tuvo un sobrecosto de aproximadamente 402€, que dentro de los márgenes de beneficio del mismo no está del todo mal si basamos un precio final ficticio del producto terminado de 16000€ + (materiales ofertados).

1.15.1 Mecanismos de control de calidad

El aseguramiento de la calidad en la redacción del trabajo implica la verificación de la completitud (falta de omisiones), la integridad de la documentación, así como una redacción clara, concisa y entendible por todos los participantes e interesados en dicho trabajo.

Al estar el trabajo desarrollado por una sola persona y verificado por un/a tutor/a, no es necesario llevar un documento de control de edición y revisión de la documentación. De esta forma, se han utilizado mecanismos básicos para la verificación de la integridad y completitud, que incluyen un control de versiones a nivel de documento y un control sencillo de la trazabilidad de especificaciones y requisitos.

Para el proyecto en sí los controles de calidad son claros, el cliente por su parte nos deja parte de una remesa de su producto a probar y de los recursos necesarios para la elaboración del mismo. Además se cuenta con el contacto a tres bandas, entre nuestra empresa, la empresa cliente y la empresa cliente de nuestro cliente, así, se puede conseguir un mayor consenso en los objetivos que se deben cumplir con el proyecto. Por último, con un número de placas producidas por nuestro cliente se hacen las respectivas pruebas finales, de la cama de pinchos, por ejemplo, que todo funcione perfectamente y puntos críticos en el software, para ello se genera un documento por nuestra parte, pensando en todos los casos de fallo, tanto hardware como software, véase que la Fuente de alimentación falle en mitad del proceso, en cuyo caso se hace una prueba pertinente.

1.15.2 Otras referencias

- Moka7: https://sourceforge.net/projects/snap7/files/Moka7/1.0.1/ (De donde se descargó la versión utilizada de Moka7 para comunicar con el PLC)
- Moka7 tutorial: https://www.youtube.com/watch?v=bryLbi-UigM&ab_channel=ChunzPS (Un video de una implementación simple usando esta librería, la cual ayudó a usarla en este proyecto)
- jSerialComm: https://fazecast.github.io/jSerialComm/ (Librería usada para comunicar con un Arduino MEGA del que más tarde hablaremos)

1.16 Resumen

En este capítulo hemos introducido el punto del que partimos, una oferta de un cliente sobre una cama de pinchos que sea asequible sin sacrificar la eficiencia ni

la eficacia de las pruebas, para ello, el uso de dispositivos asequibles y compatibles es importante así como llevar una planificación estructurada.

También hemos podido echar un vistazo a la utilidad de las camas de pinchos y sus "competidoras" las máquinas de sondas móviles, dando así un contexto y un sentido al origen de este proyecto.

2 DISEÑO INICIAL

En este capítulo trataremos los detalles iniciales del proyecto: dispositivos usados, requisitos y el porqué de algunas cuestiones anteriormente mencionadas.

2.1 Especificaciones del sistema

2.1.1 Hardware usado

Como programador en este proyecto, no se influye en los dispositivos que se usarán en principio, estos dispositivos son:

- Un PLC de Siemens: Un autómata programable que nos hará las veces de voltímetro para efectuar las mediciones de las que hablamos.
- Un PC embebido de Siemens: Debido a que el sistema tiene como objetivo estar dentro de la cama de pinchos. (PC SIMATIC IPC127E).
- Una Fuente de alimentación de laboratorio programable: La oferta exige distintos voltajes por ello se utiliza una fuente de alimentación programable, la cual puede ser programada vía Ethernet. (CPX400DP)
- Dos tarjetas de relés controladas vía Ethernet: Dado que se exige usar distintos puntos de la tarjeta se utilizan relés para decidir dónde se miden los voltajes o por donde se cortocircuita mediante estas tarjetas. (Bewinner 16 Canales Módulo de Relé, Módulo de Control Ethernet)

- JLink BASE: Para programar el microcontrolador se hará uso del dispositivo JLink BASE de SEGGER, este dispositivo es exigencia de la oferta.
- Un Ethernet Switch: Un pequeño dispositivo Ethernet Switch que servirá de nexo entre los distintos dispositivos conectados por Ethernet (PLC, PC, Tarjetas de relés y fuente de alimentación programable).

Aunque estos son los dispositivos con los que comenzamos el proyecto veremos cómo algunos cambian conforme se avanza en el trabajo dadas algunas incompatibilidades con el objetivo del proyecto.

2.1.2 Requisitos del sistema

Por otro lado, los requisitos del software a nivel de usuario están definidos, estos son:

- Poder instalar el firmware en el microcontrolador de la PCB (y que compruebe internamente que efectivamente se ha instalado).
- Poder hacer pruebas de manera automática, de manera que todas las pruebas que requiere una de estas placas se lleven a cabo en una sola ejecución. (Estas pruebas finalmente dejan guardados los datos para la posterior trazabilidad)
- Poder hacer las pruebas de manera manual, de esta forma el usuario puede probar placas que, una vez han dado un fallo en el proceso, han sido en teoría reparadas y pueden probarse sólo la función que se falló.

Para estas funcionalidades se escogió diseñar una aplicación que internamente funcionara como una máquina de estados, por lo que una vez hay un error en alguna de las pruebas o en la misma instalación, se puede reintentar la función o continuar según lo decida el usuario, dando la posibilidad de parar el proceso si falla. Además de ello hay que tener en cuenta en todo momento la disponibilidad de las demás tecnologías como la Fuente de alimentación o el PLC, que en cuyo caso si se perdiese la comunicación el programa pararía inmediatamente su normal proceso.

Para poder llevar a cabo las respectivas pruebas en el contexto en el que estamos normalmente ya sean pruebas semifuncionales o funcionales, es más

natural encontrar aplicaciones programadas en entornos como LabVIEW o similar, que ofrecen un gran apoyo al desarrollador de estas pruebas y que facilita la introducción de nuevas pruebas si hiciera falta. LabVIEW es un entorno de programación basado en diagramas de bloques que sobretodo ayuda a ingenieros no necesariamente informáticos a desarrollar aplicaciones con objetivos similares a los de este proyecto, de una manera más gráfica y sencilla a usuarios no acostumbrados a programar (National Instruments, 2021)[4].

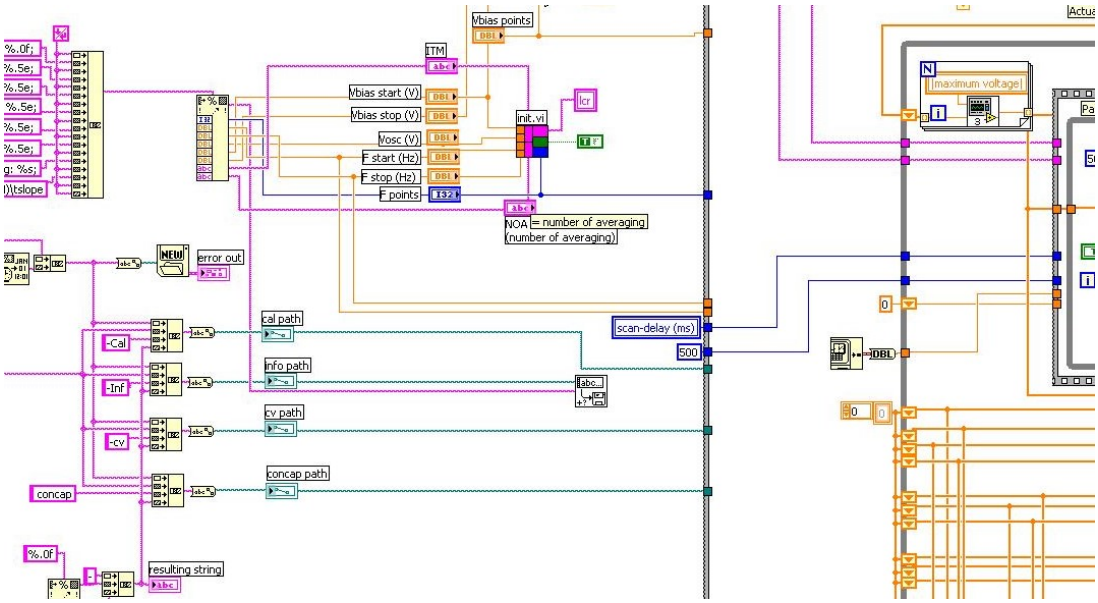


Ilustración 2 .- Diagramas de bloques de LabView (ejemplo)

Si bien LabView es una buena herramienta para estas pruebas requiere de un soporte hardware propietario, es decir, hay que utilizar un hardware por detrás perteneciente a la empresa que lleva LabView, NI "National Instruments". Además de tener que pagar licencias tanto de LabView como de alguna librería en específico (además de la licencia Runtime) y de todo ese caro hardware.

Debido a que el proyecto requiere secundariamente que sea una solución "Low Cost", se opta por usar el lenguaje JAVA, por lo que se deberá encontrar distintas librerías que ayuden a la implementación de las distintas tecnologías incluidas en el proyecto en la etapa de "investigación".

Para las distintas tecnologías usadas en el proyecto encontramos librerías que nos permiten implementarlas, estas son:

[4] National Instruments - https://www.ni.com/es-es.html

- Para la comunicación con el autómata programable de SIEMENS se utiliza la librería "*MOKA7*" la cuál fue mencionada anteriormente en el apartado "*otras referencias*". Como se explica en la página del creador de Moka7, Moka7 es la adaptación a Java del cliente Snap7, no consta de toda la librería dinámica de la que proviene, pero es más que suficiente para el proyecto que tratamos. (Nardella, 2021)[5]
- La librería Socket de Java usada para comunicarnos con la Fuente de Alimentación programable usada en el proyecto. Esta fuente de alimentación cuenta con un sistema "servidor" al cual podemos hacer peticiones que no son más que instrucciones que sirven para controlar esta mediante el protocolo TCP/IP.
- Las librerías "Thread" y "ProcessBuilder" de Java las cuáles se usarán para poder llevar a cabo la concurrencia de distintos procesos, en particular la instalación del firmware y las pruebas semifuncionales.
- La librería "jSerialComm", una librería usada en comunicación serie es ofrecida como alternativa a la librería "RxTx" y a la librería obsoleta de JAVA, "*Java Communications API*". (Hedgecock, 2021)[6](Más adelante se explicará la necesidad de esta librería).

2.1.3 Desglose final

Tal y como se mencionaba en el apartado anterior las principales funcionalidades de la aplicación han sido definidas, aparte de esas funcionalidades hay que crear unas funcionalidades "secundarias" unas que serán transparentes para el usuario (cómo se efectúan las pruebas en sí) y otras funciones que no están implementadas como tales dentro de la aplicación para el usuario "encargado" del funcionamiento del puesto. Además se incluyen funcionalidades de "recuperación" del sistema. Como explicábamos si alguna parte del sistema falla debería de recuperarse desde el software si es posible.

Funcionalidades "primarias" (definidas en el apartado anterior):

- Instalar el firmware

[5] Moka7 (14/10/2021) - http://snap7.sourceforge.net/moka7.html
[6] Fazecast jSerialComm (14/10/2021) - https://fazecast.github.io/jSerialComm/

- Pruebas automáticas
- Pruebas manuales

Las funcionalidades secundarias como decíamos son cada una de las pruebas que se efectúan en un test, por lo que su explicación queda dentro de este mismo diagrama como parte de él. Las pruebas son diferentes entre sí pero, todas guardan ciertas similitudes. Entre estas similitudes:

- Control de algunos de los puntos de la cama de pinchos mediante relés.
- Toma de mediciones mediante el PLC.
- Control de los voltajes de entrada mediante la fuente de alimentación.

Por otro lado, hay funciones que caen fuera de la implementación como tal del programa como por ejemplo cambiar las propiedades de la aplicación. Para esta tarea se utiliza un archivo "*.properties*".

Por lo tanto, se pueden destacar 2 casos de uso:

- El usuario "trabajador" cuyos cometidos son: Instalar el firmware, hacer tests completos o hacer una prueba manual. (Incluyendo funcionalidades de recuperación del sistema).
- El usuario "administrador" cuyo cometido es tener acceso a las propiedades editables del programa.

Como ya se mencionaba anteriormente las funcionalidades secundarias son las propias herramientas de las que hacen uso las funciones "prueba" para llevar a cabo cada una de ellas.

2.2 Análisis y diseño del sistema

2.2.1 Diagrama del sistema embebido

Como veíamos en un principio tenemos un sistema previamente diseñado, para entrar en contexto de cómo funciona se incluye un pequeño diagrama de dispositivos y cómo están conectados a groso modo.

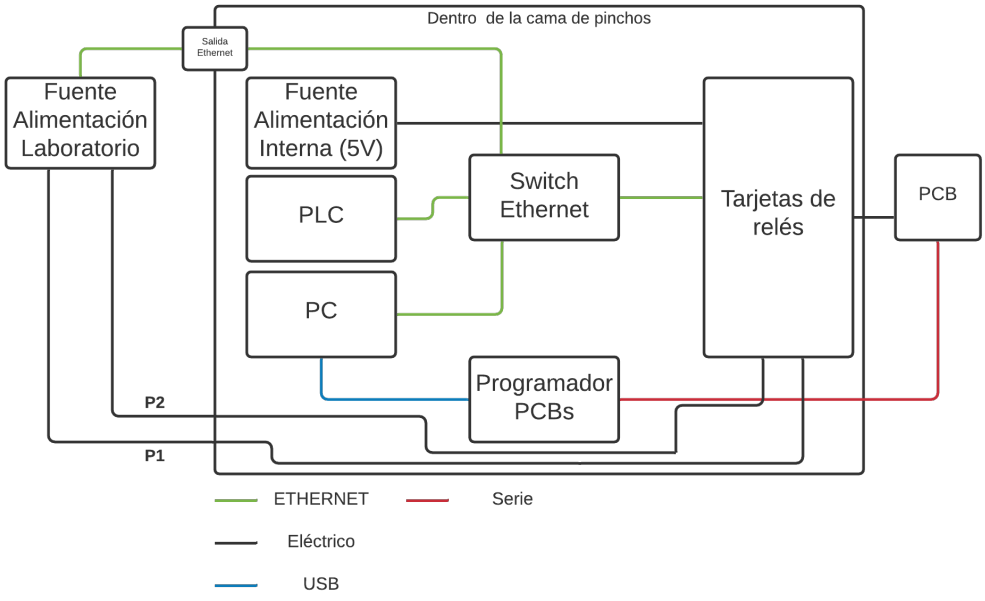


Ilustración 3 - Diagrama de sistema

Tras el desarrollo de este hubo un cambio mayor, puesto que el funcionamiento de la tarjeta de relés no era óptimo para el sistema del modo en que lo hacía, por ello se cambió ligeramente el diseño quedando finalmente así:

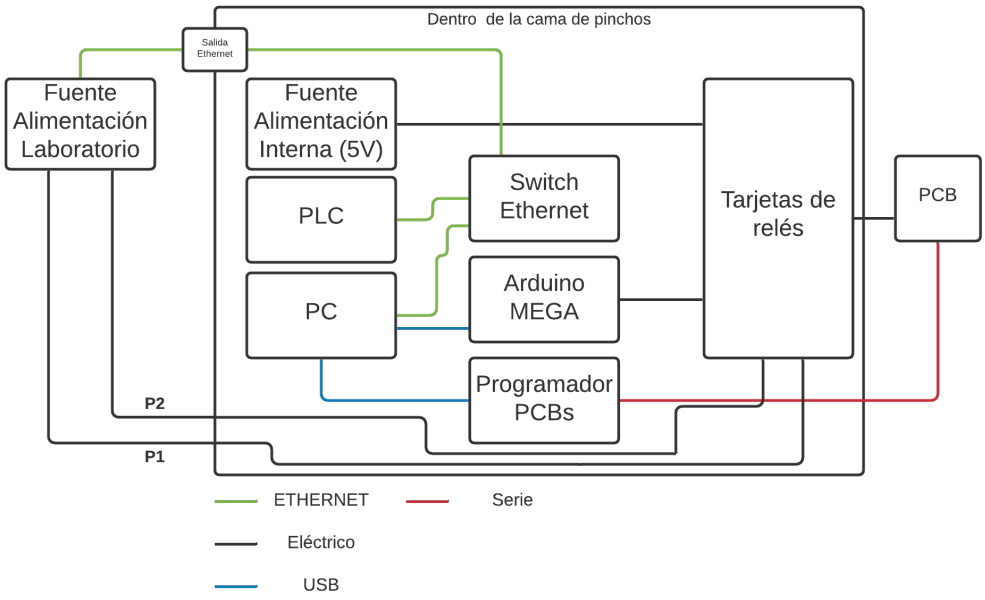


Ilustración 4 - Diagrama de sistema post-cambio

Como vemos se incluye un microcontrolador Arduino MEGA, esto es como veníamos explicando, tras hacer pruebas de rendimiento conectando las tarjetas de relés mediante Ethernet, puesto que se conseguía controlar los relés pero con un retraso de respuesta importante.

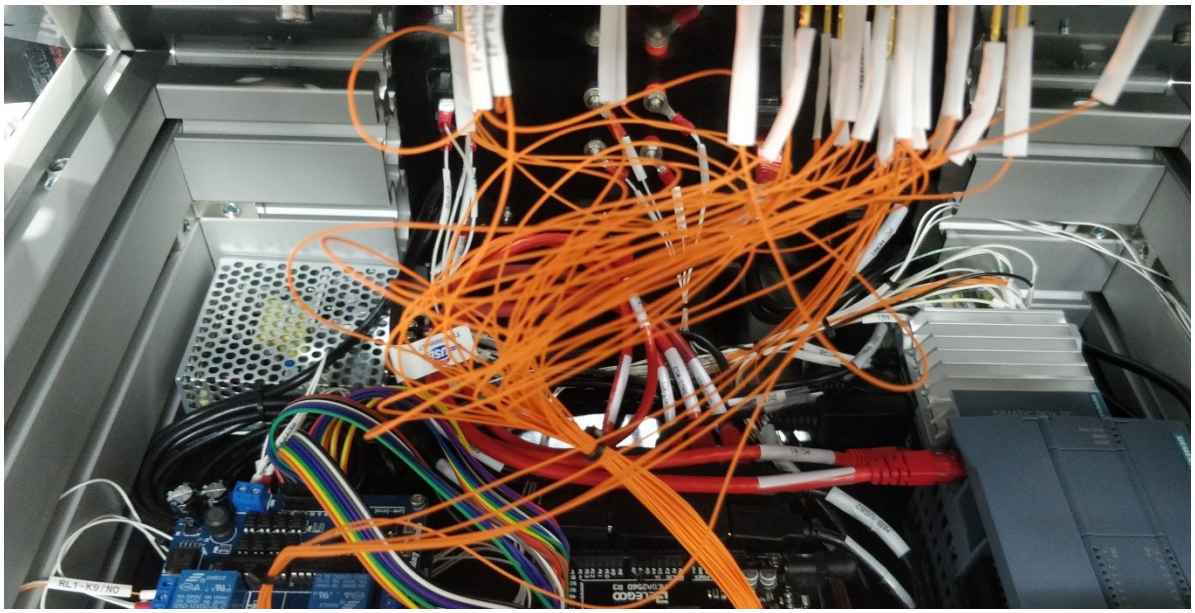


Ilustración 5 - Imagen del interior de la cama de pinchos

Como podemos apreciar en la imagen, todos los dispositivos que hemos visto previamente quedan dentro de la cama, para que sea un dispositivo único como vemos a continuación:

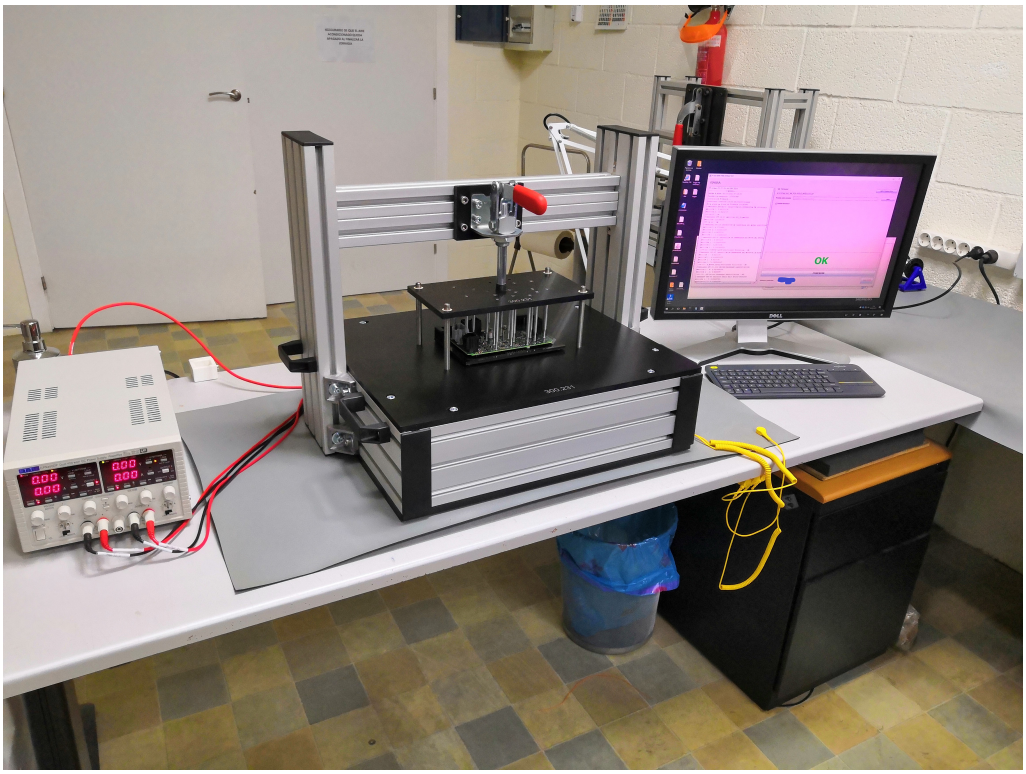


Ilustración 6 - Cama de pinchos en funcionamiento

2.2.2 Casos de Uso

Por otro lado, como mencionamos en el apartado anterior se intuyen dos actores principales cuya representación se muestra en la siguiente gráfica de manera simplificada en forma de casos de uso.

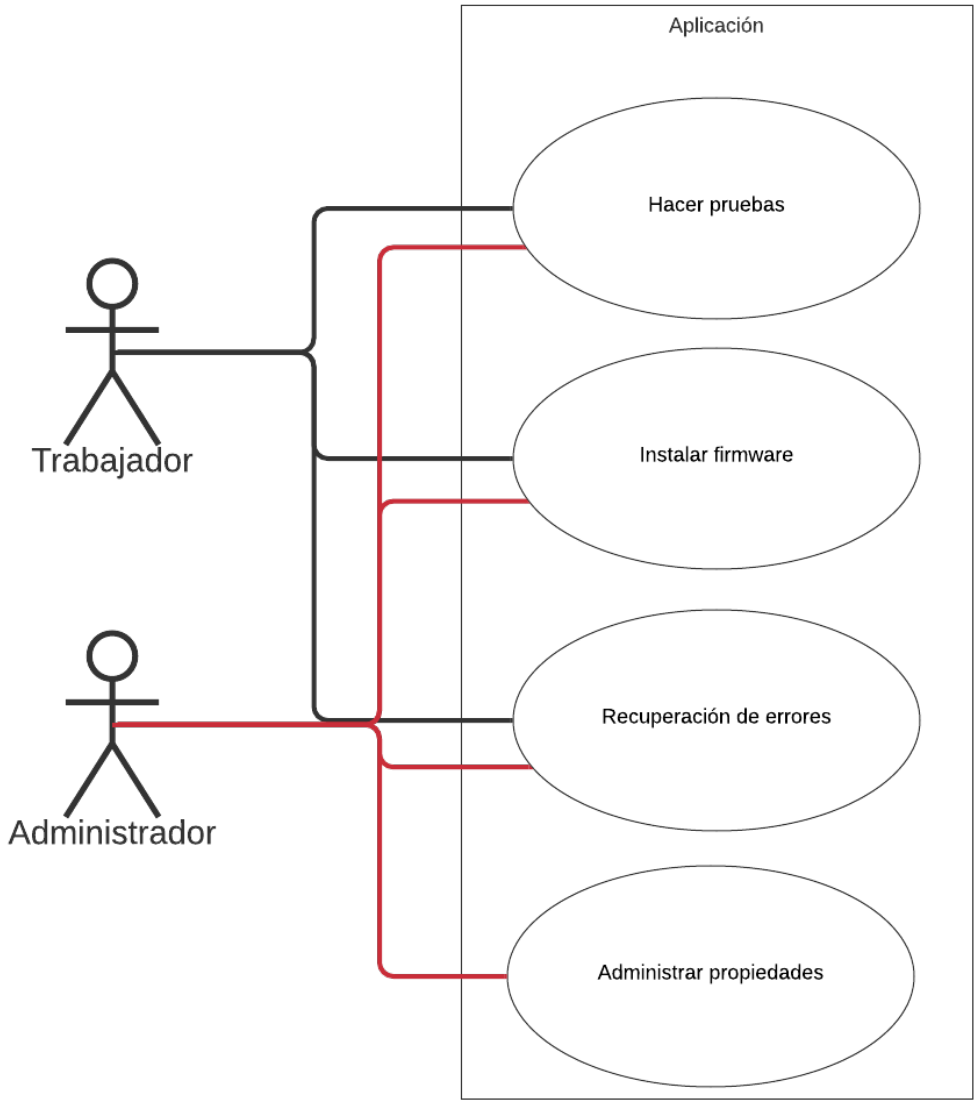


Ilustración 7 - Diagrama de caso de uso

Dentro de algunos casos de uso hay que hacer algunas distinciones:

- **“Hacer pruebas”**: si es la prueba completa de una PCB tenemos que llevar el seguimiento de la PCB por lo que tendremos que hacer la lectura de un Datamatrix 2D, parecido a un código QR, para ello se usará un

lector de Datamatrix cedido por el cliente y que no necesita drivers, puesto que funciona como un “teclado”.

- **“Instalar firmware”**: Como veremos más adelante es una tarea que puede tanto ser independiente como complementar a “Hacer pruebas”.
- **“Recuperación de errores”**: Es una función que sirve tanto para que la aplicación no quede “congelada” en ningún momento así como para proteger la PCB y el equipo.
- **“Administrar propiedades”**: Es una tarea que lleva en exclusiva el administrador o “encargado de línea”, en esta tarea el encargado debería ser capaz de cambiar algunos accesos a ficheros, cambiar datos relevantes sobre la lectura del datamatrix, etc.

Debido a la petición del cliente como se verá más adelante la interfaz tiene que ser lo más simple posible, no dejar lugar a dudas al operario es prioritario para el cliente, cualquier duda del operario trabajando se traduce en pérdida de rendimiento en la línea, por ello, el diseño de una interfaz simple y familiar es un punto a tener en cuenta. Como veremos más adelante se sigue un diseño de interfaz parecido al de interfaces anteriores.

2.2.3 Diagramas de secuencia

Como tratábamos en la introducción, el sistema fue diseñado como una máquina de estados por lo que es importante tener en consideración de qué manera se desarrollan los hechos y cuál deben ser tanto los estados como el flujo de la aplicación, por ello, antes de diseñar todo el sistema se estudia el caso y se dibujan unos diagramas para comprender el funcionamiento de las funciones a groso modo y así tener una guía que seguir en el posterior desarrollo.

Así como nos encontramos con tres requisitos principales, se hacen tres diagramas según estos:

1. Diagrama de pruebas automáticas:

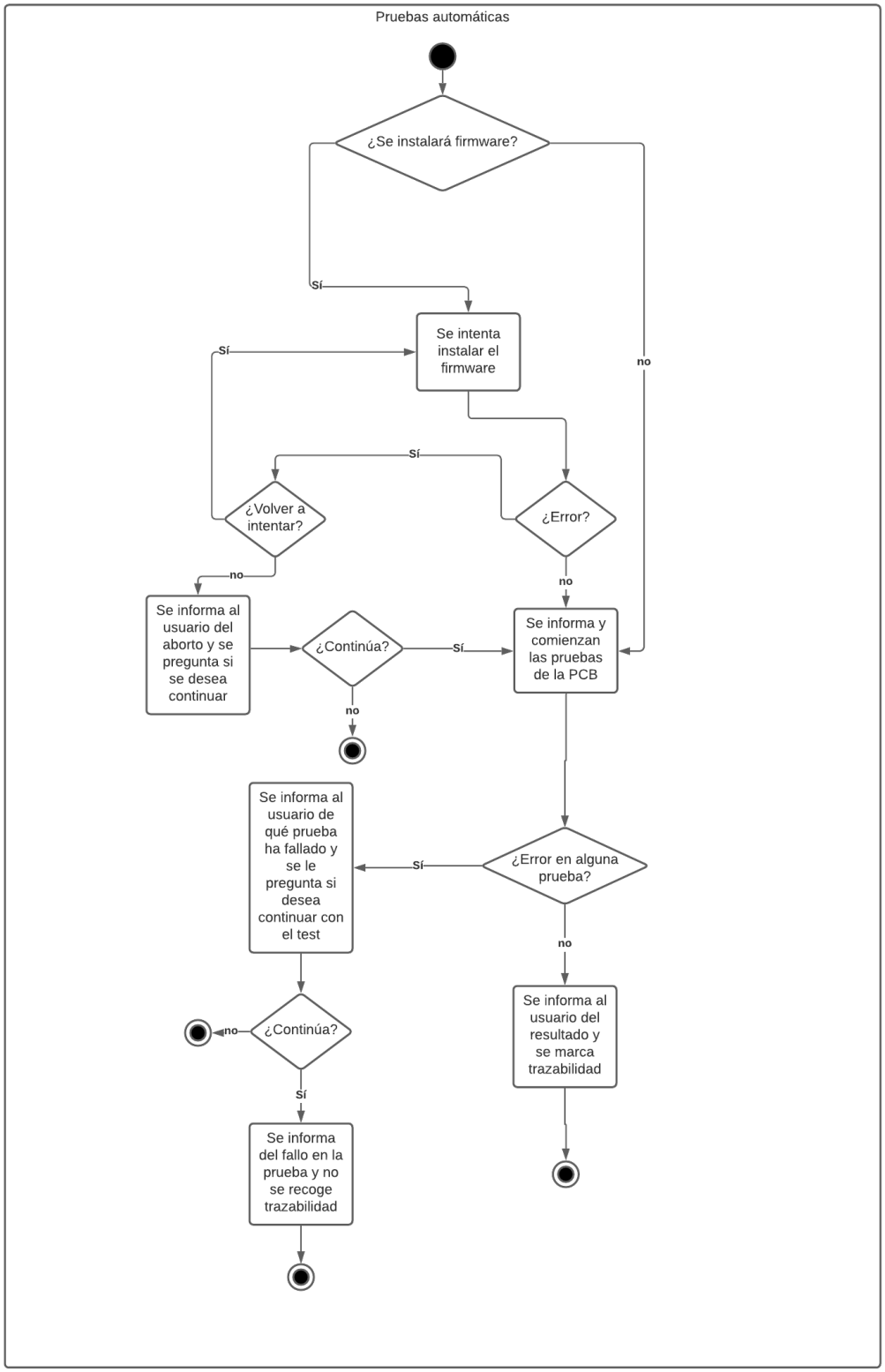


Ilustración 8 - Diagrama de secuencia de pruebas automáticas

2. Diagrama de pruebas “manuales”:

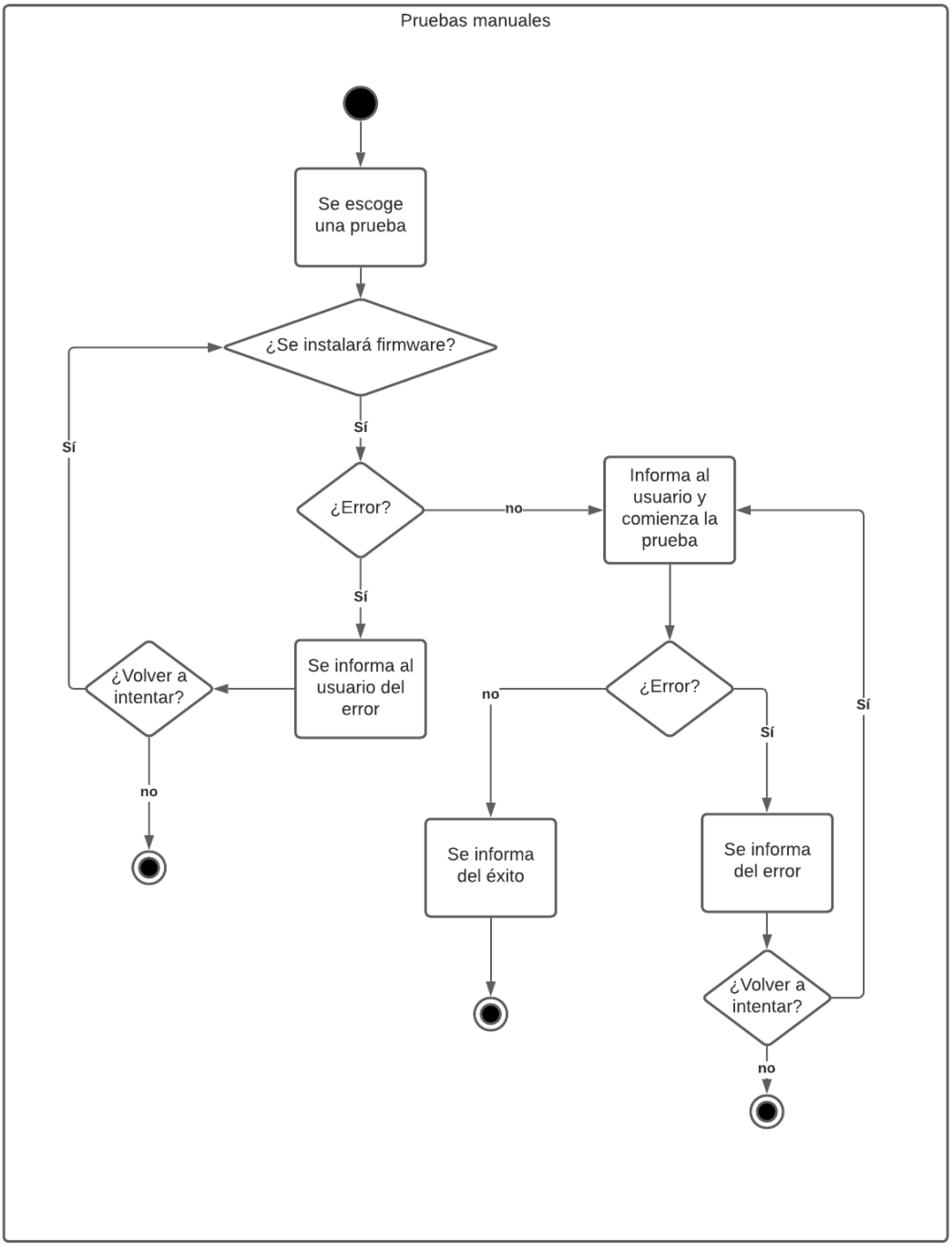


Ilustración 9 - Diagrama de pruebas manuales

3. Diagrama de instalación de firmware:

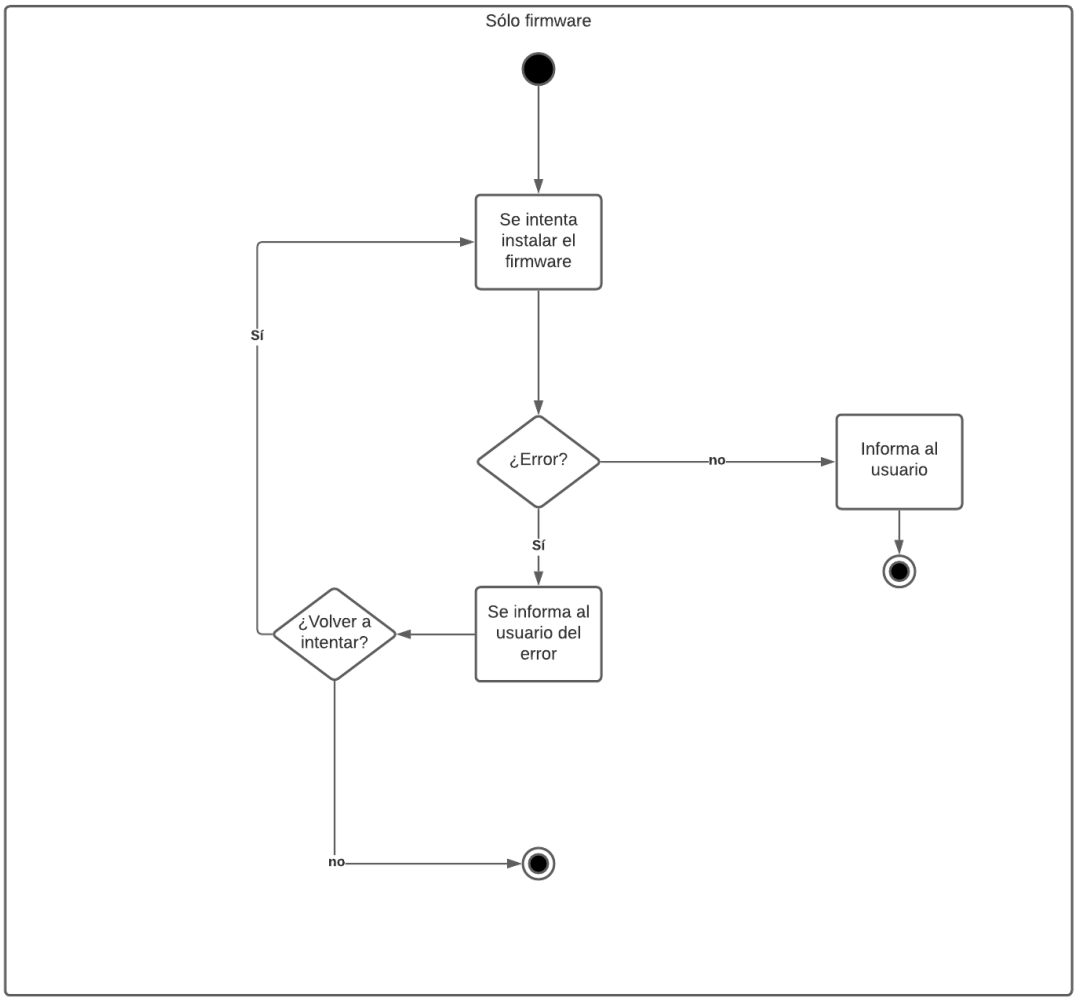


Ilustración 10 - Diagrama de secuencia de instalación de firmware

1. **Diagrama de pruebas automáticas**: Como vemos en la (Ilustración 6), en el funcionamiento del requisito para las pruebas automáticas, es decir, todas las pruebas de un test seguidas, encontramos dos fases principales, la instalación del firmware en el microcontrolador de la PCB y las pruebas para comprobar el correcto funcionamiento de esta.

Como se puede observar, la instalación del firmware es un paso opcional puesto que en ocasiones el operario podría ya haber instalado el firmware en un test previo (por ejemplo con resultado erróneo en las pruebas) y sólo está haciendo un chequeo post reparación de la PCB. Tras la instalación del firmware, o no, la segunda fase, las pruebas del test donde se comprueba el

correcto funcionamiento de la PCB y de sus funciones (fuera del ecosistema propio). Por último se recoge registro en el fichero de resultados de las pruebas para llevar una trazabilidad.

2. **Diagrama de pruebas "manuales"**: En este caso (Ilustración 7) la función principal es que el operario compruebe si un defecto en una PCB anteriormente probada ha sido reparado, este requisito/función existe con el propósito de que el operario invierta el menor tiempo posible en estas tareas y para ello se le da la opción de hacer una única prueba de las opciones existentes. Así como en el primer diagrama podemos observar que se ofrece el instalar el firmware como una opción y una vez pasada esta fase se efectúa la prueba seleccionada.

3. **Diagrama de instalación del firmware**: Esta es una fase independiente y corresponde con instalar un firmware de manera independiente, así pues, se puede actualizar el firmware sin tener que llevar a cabo pruebas.

Estos diagramas tienen un punto en común, cada paso que se da en estas funciones tiene que estar correspondientemente comunicado al operario para que en el momento de algún error, se pueda recoger el tipo y el encargado pueda decidir que opciones tomar.

2.2.4 Diseño de la interfaz

Para realizar algunas de estas funciones a nivel interfaz con el usuario, primero se prepara un pequeño boceto para el cliente, para que haga sus peticiones respecto al diseño para que le sea lo más cómodo posible al operario. Al haber contado con otros trabajos para este cliente encontramos otros proyectos de los cuales podemos sacar inspiración para la interfaz.

Ejemplo:

Ilustración 11 - Aplicación Programadora Anterior

Como se puede ver es una interfaz extremadamente sencilla para el operario, esta aplicación es usada para programar el microcontrolador de la PCB. En la interfaz podemos ver varios campos de salida entre los que se encuentra:

- La consola: encargada de representar todos los pasos del procedimiento.
- "Tipo PCB": que se encarga de representar qué PCB están programando.
- "Ver. Firmware": representa la versión firmware que se va a instalar.
- "Campo Resultado": campo que indica el resultado de la operación.
- Botón "COMENZAR": botón que inicia la operación.
- Firma de creación: dónde va el logo de la empresa desarrolladora del software.

Algunas de estas partes las recrearemos por petición del cliente, cómo se puede ver en el siguiente mockup hecho con la herramienta de interfaces de NetBeans JAVA:

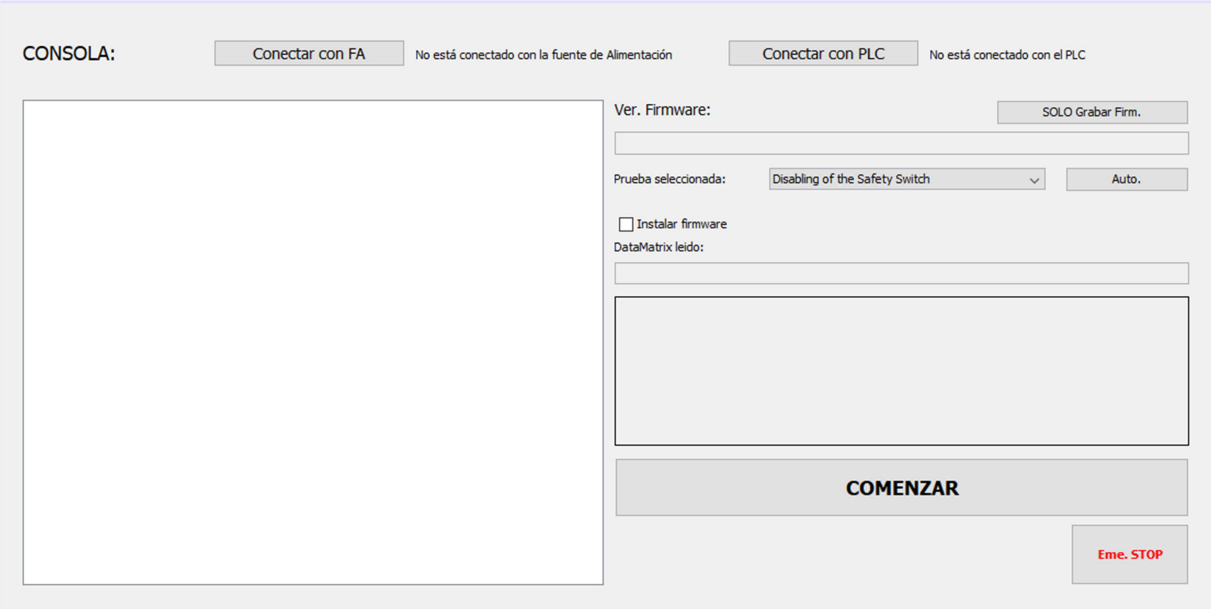


Ilustración 12 - Interfaz programa final (prototipo)

Como veíamos en el caso anterior algunas de las partes del anterior software hecho para el cliente se puede reutilizar, la consistencia en el software que se entrega al cliente es importante puesto que ayuda al operario a adquirir familiaridad con el producto y reutilizar sus conocimientos y destrezas de una aplicación a otra.

Para conseguir esta consistencia colocamos los elementos reutilizados en los mismos espacios, en el mismo orden. Sin embargo, podemos observar cómo hay distintos apartados nuevos dentro de nuestra aplicación, por ejemplo: en la parte superior del prototipo observamos un par de botones para conectar tanto con el PLC como con la FA (Fuente de Alimentación) estos botones y avisos sólo aparecerían cuando el programa lo requiriese para recuperar el control de este si este se pierde.

Por otro lado, podemos observar en el mockup un botón que serviría para grabar el firmware sólo y no hacer las pruebas, uno de los requerimientos que nos pedían, este es un botón, es decir, una vez clicamos este la instalación del firmware comienza.

Por último, vemos debajo del campo de versión de firmware, el “selector de pruebas” que sería accesible si el interruptor Auto./Manual está en modo “Manual”, es decir, si el operario sólo quisiera hacer una prueba. Tras esto un selector, por si el operario no quiere instalar el firmware en el microcontrolador (por ejemplo, el microcontrolador ya tiene firmware y se está volviendo a probar). Al final, hay un

botón que simula una “Seta de emergencia” un botón que detiene de manera segura todo el procedimiento.

2.2.5 Resumen

En este capítulo hemos echado un vistazo al desglose de requisitos, al hardware que se usará y hemos hecho una pequeña planificación de cómo queremos que funcione y se vea el sistema en términos generales con una perspectiva del lado del cliente. Este capítulo da un contexto e información necesaria para el siguiente capítulo.

3 DESARROLLO

Este capítulo se dividirá en 4 subapartados, uno por cada etapa vista en la planificación temporal. El objetivo de este es entender las decisiones tomadas y las soluciones implementadas en el transcurso del proyecto, así como dar una visión del proceso de desarrollo de una aplicación específica en el contexto industrial electrónico.

Los distintos subapartados se tratarán de manera cronológica al igual que se pueden observar las etapas y tareas en la planificación temporal mostrada anteriormente (Tabla 1 y 2).

3.1 ETAPA DE INVESTIGACIÓN

Como explicábamos en el subapartado de planificación temporal este proyecto se divide en cuatro etapas diferenciadas, empezando por el apartado de investigación, localizado cronológicamente como las 3 primeras semanas del proyecto.

La principal característica de este trata en base a buscar las herramientas necesarias y probarlas para hacer este proyecto funcional.

3.1.1 Fuente de alimentación y tarjeta de relés

Así pues, la primera semana del proyecto se propuso como tarea probar las comunicaciones tanto con la Fuente de alimentación como con las tarjetas de relés, ambas controladas mediante comunicación TCP/IP.

Al usar el mismo tipo de comunicación para ambos se puede usar la misma librería, en este caso, la librería Socket es una librería idónea. Para saber cómo funcionan ambos dispositivos se usan los manuales de estos. En el caso de la fuente de alimentación es posible encontrar fácilmente el manual de la misma, dónde se explica el formato de los mensajes. Procedemos a explicarlo:

La fuente de alimentación de laboratorio usada es una CPX400DP de la marca TTi, es una fuente de alimentación de laboratorio de dos salidas y al igual que las demás es regulable. Por otro lado, esta fuente de alimentación cuenta con un control basado en un servidor web al que podemos mandar peticiones en el formato que nos indican en su manual de usuario.

Ilustración 13 - Fuente de alimentación de laboratorio CPX400DP

Con la librería Socket el procedimiento queda así:

```
1. //DECLARACION DE VARIABLES
2. Socket socketFuente;
3. OutputStream fuenteOutput;
4. InputStream fuenteInput;
5.
6. //INICIALIZAR VARIABLES FUENTE
7. socketFuente = new Socket(config);
8. fuenteOutput = new OutputStream(socketFuente);
9. fuenteInput = new InputStream(socketFuente);
10.
11.     //INICIAR LA FUENTE A 0
12.     Enviar Mensaje A Fuente Alimentacion("V1 a 0");
13.     Enviar Mensaje A Fuente Alimentacion("I1 a 0");
14.     Enviar Mensaje A Fuente Alimentacion("V2 a 0");
15.     Enviar Mensaje A Fuente Alimentacion("I2 a 0");
16.
17.     Si fallo Dialogo a Usuario de Fallo
```

Primero, declaramos una variable *Socket* (socketFuente), la cual mantendrá la comunicación como cliente con la fuente de alimentación. Además para poder comunicarnos por este "canal" crearemos adicionalmente dos variables de tipo

PrintWriter (fuenteOutput) y InputStream (fuenteInput). Para declarar estas variables primero declararemos la variable de tipo “Socket” con el contructor de la clase “*Socket(String IP, Int puerto)*” que en esta ocasión lo provee el fabricante por defecto en “*192.168.0.100:9221*”.

Una vez declarado socketFuente nos comunicaremos mediante las variables fuenteOutput (para enviar las peticiones a la fuente) y fuenteInput (para recibir las respuestas de la fuente). En las líneas 6 y 7 podemos ver la inicialización de los streams.

Más adelante veremos como funciona la función “*enviarMensajeAFuenteAlimentación(String)*” para facilitar la implementación de las pruebas, por ahora sólo tenemos que saber que las instrucciones que aquí aparecen se utilizan para inicializar la fuente de alimentación desde 0.

Aprovechando esta última podemos explicar qué formato tienen las distintas instrucciones que mandamos a la fuente de alimentación de forma resumida:

Para cambiar el voltaje de salida de una de las dos salidas tal y como aparece en el código:

V<número de salida> <numero en decimales de Voltios>

Ejemplo:

V1 0; o V1 0.0;

Cambia el valor de Voltios de la salida número 1 a 0 Voltios.

El resto de comandos se pueden encontrar en el apartado “Command List” que aparece en el manual “*CPX400D &DP Instruction Manual – English.pdf*” adjuntado en el proyecto.

Una vez comprobado el correcto funcionamiento de la comunicación con la funete de alimentación, se procede a comprobar el funcionamiento de las tarjetas de relés.

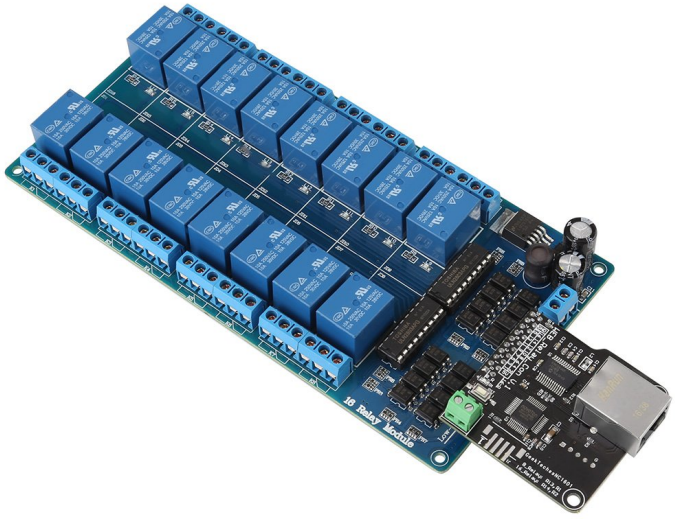


Ilustración 14 . - Módulo de relés

El funcionamiento de estas tarjetas de relés es similar al de la fuente de alimentación pero ligeramente distinto puesto el formato de los mensajes es mucho más sencillo al tratarse del control de 16 relés. En la página donde se encontró el artículo (https://www.amazon.es/Control-Ethernet-Servidor-Tarjeta-controladora/dp/B08DK2XJRT/ref=sr_1_43?dchild=1&keywords=tarjeta+reles+controlador+ethernet&qid=1634571436&sr=8-43 comprobado a día 18/10/2021) se pueden encontrar unas escuetas instrucciones de uso:

"IP predeterminada: 192.168.1.4 prot: 30000 http://192.168.1.4/30000

(puede cambiar la dirección IP predeterminada)

Comentario HTTP:

http://192.168.1.4/30000/00: Relé-01 APAGADO

http://192.168.1.4/30000/01: Relay-01 ON

http://192.168.1.4/30000/02: Relay-02 OFF

http://192.168.1.4/30000/03: Relay-02 ON

http://192.168.1.4/30000/04: Relay-03 OFF

http://192.168.1.4/30000/05: Relay-03 ON

...

http://192.168.1.4/30000/30: Relay-16 OFF

http://192.168.1.4/30000/31: Relay-16 ON

http://192.168.1.4/30000/41: Ingrese

http://192.168.1.4/30000/40: Salir

http://192.168.1.4/30000/42: Página siguiente

http://192.168.1.4/30000/43: Página siguiente”

El funcionamiento es sencillo consta de una petición mediante la dirección IP/Puerto/ seguido de un número consecutivo a la instrucción que queramos llevar a cabo.

Así pues de la misma forma que la fuente de alimentación fue probada, tras hacer pruebas activando y desactivando los distintos relés mediante este tipo de comunicación, se llega a la conclusión de que el tiempo de respuesta por parte de los relés es insuficiente y variable.

Tras este problema se comienza en el proyecto un proceso de “Incidencia” interno en la empresa, un proceso por el cual se busca una solución al problema y se trata de fundar un método para evitar en futuros casos.

En este caso particular y tras hacer las pruebas, se comprobó que todos y cada uno de los relés funcionaba de manera correcta, por lo que el problema no erradicaba en la tarjeta de relés si no en el módulo Ethernet que podemos apreciar en la “Ilustración 9” como una placa negra con puerto Ethernet. Se propusieron dos soluciones o bien se compraba una nueva tarjeta de relés que pudiera ser mejor en este tipo de comunicación o se cambia el propio módulo de Ethernet de esta tarjeta.

Tras una búsqueda infructífera de otra tarjeta de relés se decide investigar un poco más a fondo el funcionamiento de esta con resultados positivos, se llega a la siguiente conclusión:

- El módulo Ethernet solo trata como intermediario para las peticiones traduciéndolas en una salida de corriente que activa/desactiva un relé según el número de la petición a exención de los códigos especiales de navegación que veíamos anteriormente.

- El módulo Ethernet por lo tanto es un microcontrolador muy simple.

Con esta información se decide poner a prueba una posible solución propuesta, cablear un Arduino MEGA a ambas tarjetas de relés y controlar ambas desde un solo microcontrolador con una comunicación Serie.

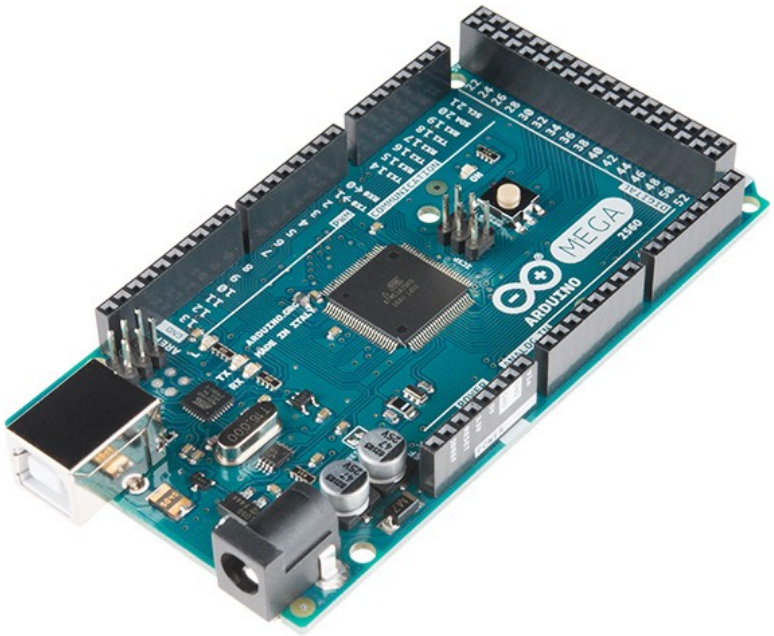


Ilustración 15 - Arduino MEGA

A partir de aquí se comienza una implementación sencilla para este Arduino, el objetivo de éste es sustituir el anterior módulo Ethernet de dos tarjetas de relés de 16 relés cada una usando el puerto serie de Arduino. Para ello, eléctricamente serán usados las salidas digitales de la placa MEGA de la 22 a la 54 y de alguna manera Arduino tendrá que traducir un “código” que mandemos desde la aplicación para que maneje los relés, para ello usaremos la trama de un byte de la siguiente manera:

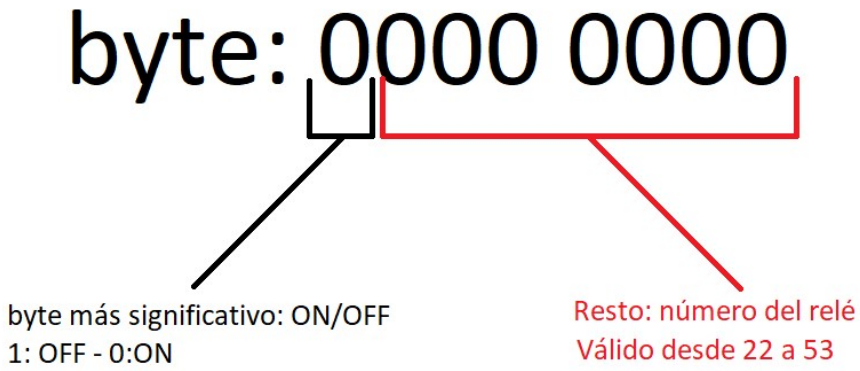


Ilustración 16 - Trama byte Arduino MEGA

Como se aprecia en la imagen explicativa con 1 byte de información es más que suficiente para controlar 32 relés (y más si se diera el caso), $2^7 = 128$ posibles relés controlables en total pero, como van a ser controlados directamente por las salidas digitales del Arduino MEGA las direcciones escogidas y válidas serán desde la número 22 (0001 0110) a la 53 (0011 0101). Así pues, las tramas válidas para el programa que usaremos serán:

0001 0110 ⟶ Cerrar relé de salida 22 (tarjeta 1 relé 1)

.

.

.

0011 0101 ⟶ Cerrar relé de salida 53 (tarjeta 2 relé 16)

...

1001 0110 ⟶ Abrir relé de salida 22 (tarjeta 1 relé 1)

.

.

.

1001 0101 ⟶ Abrir relé de salida 53 (tarjeta 2 relé 16)

0000 0001 ⟶ Señal para comprobar conexión con Arduino (sólo pruebas)

Para llevar a cabo esta operación se hace un pequeño script/programa para el Arduino MEGA, comprobar *apartado 5.2.* . Para comprobar el correcto

funcionamiento de este se crea una pequeña aplicación de control mediante botones que se dedica a mandar las tramas (cabe destacar que no lleva ningún control sobre el estado de los relés en tiempo real, importante detalle a tener en cuenta cuando tratemos las pruebas de manera individual).

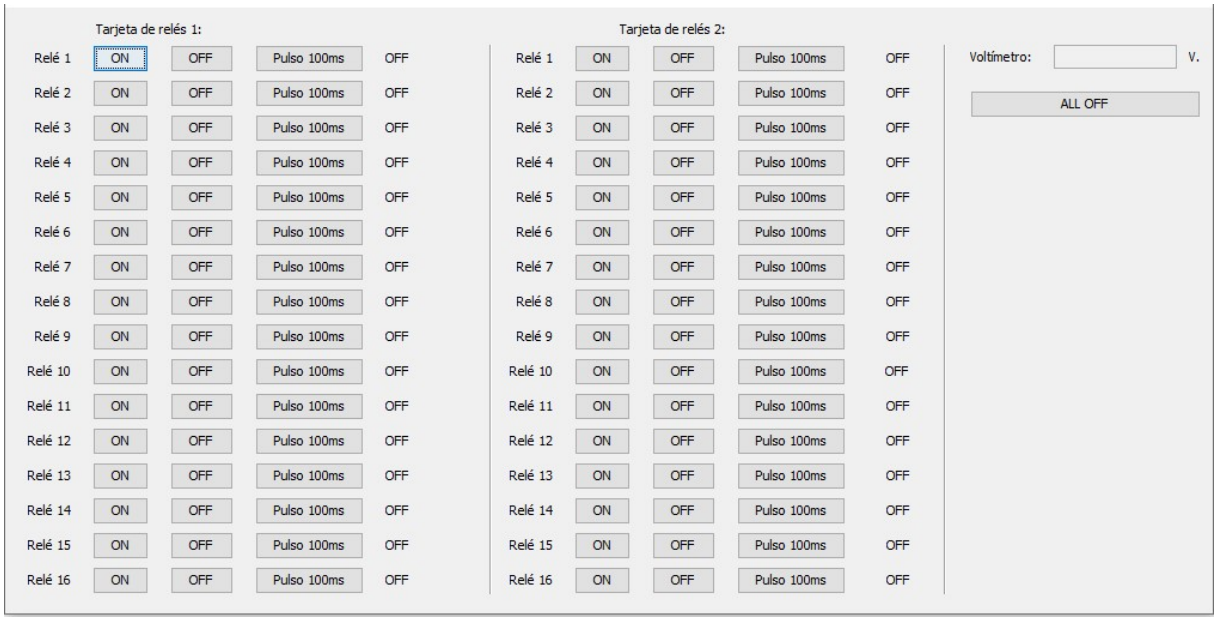


Ilustración 17 - Aplicación pruebas tarjetas de relés.

Como se ve esta aplicación sólo lleva el registro desde que se toma comunicación con Arduino, momento en el que se mandan las tramas para apagarlos todos. A continuación veremos cómo se efectúa la comunicación con Arduino gracias a la librería jSerialComm de fazecast:

```
1. //DECLARACIÓN DE VARIABLES
2. SerialPort puerto_ser = null
3. SerialPort[] puertos = SerialPort.getPuertos()
4. //BÚSQUEDA DEL PUERTO QUE CONTESTE
5.     ForEach puerto En puertos Hacer
6.         Si puerto.nombre != "COM1" Entonces
7.               Configura puerto
8.               EscribeMensajeAPuerto(1)
9.          Si hay respuesta Entonces
10.                  Si ComprobarRespuesta Entonces
11.                       Puerto_ser = puerto
12.                       BreakForEach
13.                  FinSi
14.              FinSi
15.          FinSi
16.     FinForEach
17.
18.     InputStream inSerie = new InputStream(puerto_ser)
19.     OutputStream outSerie = new OutputStream(puerto_ser)
```

Como vemos es una manera muy sencilla de tener comunicación serie con un Arduino desde una aplicación JAVA, una vez hemos comprobado que tenemos esta comunicación abierta el control de los relés se parece a esto otro:

```
1. outSerie.write(22);
2. estado22.setText("ON");
```

Además se su respectiva try-catch clause. En este caso activaríamos el relé 1 de la tarjeta 1.

3.1.2 PLC

Una vez solventado el anterior problema, se empieza a investigar la comunicación con el PLC de marca SIEMENS.

Ilustración 18 - PLC SIEMENS

Para comunicar con el PLC de SIEMENS desde JAVA encontramos la librería Moka7 la cual permitiría acceder a los registros de memoria del PLC y conseguir la información de él. Antes de seguir con la implementación de JAVA, hay que explicar cómo funciona este PLC y cómo funciona su memoria.

El PLC funciona casi como un equipo PC, tiene **CPU**, **memoria**, **módulo de entradas/salidas**, **fuente de alimentación** y la **unidad de programación**, la principal diferencia es que su función dista mucho de la de un computador personal.

Se trata de un autómata industrial, usado para detectar diversos tipos de señales de proceso y actuar a estos de alguna manera previamente programada. (SRC, 2021)[7]

Normalmente estos PLCs usan software específico para ser programados y esta no es una excepción, en este caso usaremos "**SIMATIC TIA Portal**" el software propio de los PLCs de SIEMENS que sirven tanto para programar los HMI (Human-Machine Interfaces) y los PLCs. Aunque sólo nos ocuparemos en este caso de la programación del PLC:

La aplicación del PLC de Siemens tiene como objetivo recoger en una dirección de memoria el dato que lee de una entrada analógica, que nos dará un valor en Voltios de lo que se mide en la PCB que queremos probar. Para ello esta breve aplicación será escrita con el lenguaje SCL (Structured Control Language), un lenguaje común a todos los controladores.

```
1  #Aux := NORM_X(MIN:=0, VALUE:=#Valor, MAX:=27648);
2  "DB".Valor1 := SCALE_X(MIN:=0.0, VALUE:=#Aux, MAX:=15.0);
3
```

Ilustración 19 - Función escalado de señal

Como vemos es un programa muy breve que como veremos a continuación se "llama" desde la función "main" que para entenderlo mejor es cómo la función "loop" de Arduino, una función que no deja de ejecutarse de manera cíclica.

Esta función es utilizada para escalar el valor que recoje la entrada analógica para ello se emplean dos funciones: una de normalización de la variable (NORM_X) y una función de escalado (SCALE_X). La primera se utiliza para que independientemente de los valores que entren, estos queden dentro de un rango, en este caso 0 y 27648, cuyo sentido es el rango de valor para una señal analógica. Tras esto el valor que queda comprendido entre 0 y 27648 se traduce en un voltaje mediante el escalado, comprendiendo así entre los valores 0.0 V y 15.0 V. Por último, en la segunda línea podemos ver como el valor es guardado en una variable perteneciente a una "DB".

Las "DB" son Bloques de Datos, estructuras que guardan datos en la memoria del PLC, datos/variables a las que se les puede aplicar distintas propiedades como

[7] ¿Cómo funciona un PLC? (15/10/2021) - https://srcsl.com/que-es-un-plc/

la “remanencia”, que un dato quede guardado incluso cuando se reinicie el PLC, entre otros.

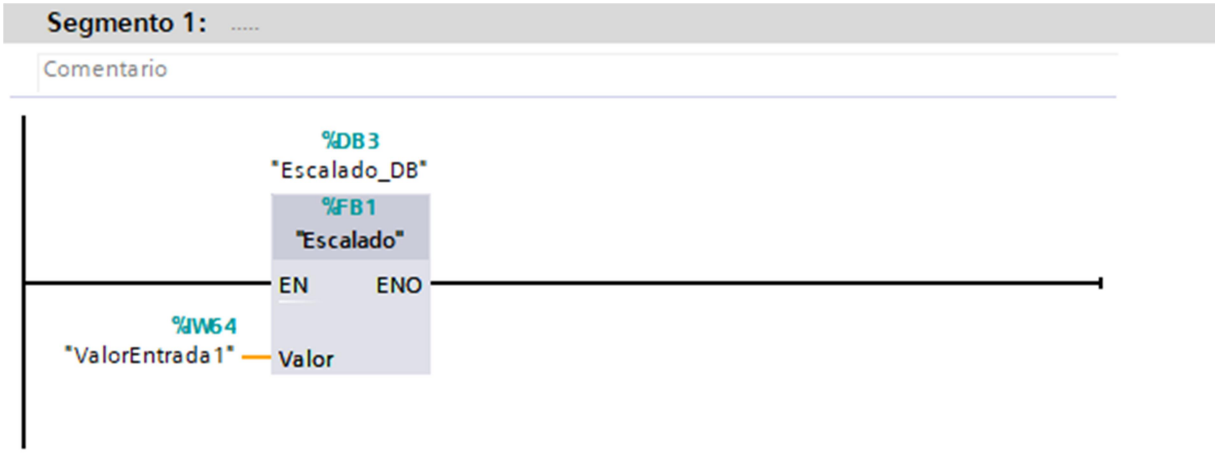


Ilustración 20 - Función MAIN del PLC

Como decíamos la función MAIN sólo llamara a la función encargada de escalar el valor de la entrada analógica.

Una vez explicado el contexto de la aplicación del PLC, proseguimos explicando cómo se comunica este con la aplicación JAVA mediante la librería Moka7 anteriormente mencionada, para ello veremos de manera sencilla cómo se establece la conexión entre la aplicación y el PLC así como, cómo acceder a los datos almacenados en un Data Block.

Para conectar el PLC SIEMENS con la aplicación en Java es pertinente usar la librería Moka7 la cual contiene múltiples clases, en concreto usaremos la clase “*S7Client*” que nos permite acceder al PLC como si este fuera un server.

```
1. //DECLARACION DE VARIABLES
2. S7Client cliente = new S7Client()
3.
4. //INICIALIZAR CONFIGURACIÓN
5. cliente.setConfiguracion(config)
6.
7. //COMPROBACION DE CONEXIÓN
8.      SI !cliente.estaConectado() ENTONCES
9.          Dialogo a Usuario de Error
10.     FINSI
```

En este breve ejemplo podemos ver cómo se conecta fácilmente con un PLC, como decíamos usaremos la clase S7Client, crearemos una variable cliente (*client*). Una vez creada e inicializada con la función new, usaremos la función

SetConnectionType(Int) con la que definiremos el tipo de comunicación con el PLC, en nuestro caso usaremos **S7.OP** la cual se usa para efectuar operaciones como la lectura/escritura entre otros.

Por otro lado *ConnectTo(String, Integer, Integer)* nos permite establecer la conexión con el PLC propiamente dicho. Los parámetros usados son:

- **String (IP):** IPv4 en la que está localizado en la red el PLC. *(en nuestro caso la 192.168.0.1)*
- **Integer (Rack):** Este parámetro es usado cuando hay varios PLCs con la misma IP en una red. *(0)*
- **Integer (Slot):** Al igual que el parámetro anterior es un identificador más. *(1)*

Una vez se usa la función *ConnectTo* comprobamos que se ha llegado a efectuar de manera correcta la conexión mediante la variable pública de la clase S7Cliente "*Connected*" que avisará si en algún momento perdemos esta o si no se estableció la conexión en primera instancia.

Hecho esto, pasamos a ver cómo podemos obtener los datos que van a almacenarse en el DB anteriormente mencionado, las medidas de Voltaje que toma el PLC, mediante la función ReadArea.

```
1. client.ReadArea(S7.S7AreaDB, DBNumber, StartAddress, Amount,
   Buffer);
```

Esta función toma como parámetros:

- **Integer Area**: el área de memoria a la que accedemos en nuestro caso los DataBlocks (*S7.s7AreaDB*).
- **Integer DBNumber**: El numero identificativo del DataBlock.
- **Integer StartAddress**: La dirección de comienzo del valor que queremos tomar.
- **Integer Amount**: Cantidad de datos en número de bytes, en nuestro caso 8 bytes, un Float.
- **byte[] Buffer**: un buffer de bytes donde guardaremos la lectura (temporalmente).

Esto no es todo puesto que para la aplicación lo que hay es solo un vector de bytes. Para transformarlo en un Float que pueda leer la aplicación se usa de nuevo otra función de la librería Moka7.

```
1. Float resultado_consulta = S7.GetFloatAt(Buffer /*BUFFER DE
   DATOS*/, 0/*NUMERO DE PALABRA*/ );
```

S7.GetFloatAt convierte la palabra que hay dentro del buffer en la posición que se le indica en un Float "legible" por Java. Hecho esto, *resultado_consulta* ya guarda la lectura de voltaje.

Una vez probadas que las lecturas son correctas y se corresponden a experimentos que se hacen con la PCB de manera manual, se procede al siguiente paso. Instalar el firmware mediante el uso del programador jLink BASE Compact.

3.1.3 Instalar firmware mediante jLink BASE Compact

El dispositivo jLink es un dispositivo usado para instalar el firmware deseado a los microcontroladores con los que este es compatible.

Ilustración 21 - Programador J-LINK BASE Compact

Normalmente este dispositivo es usado con alguno de sus software específicos para la tarea.

- **JLINK.exe**: Es la herramienta tipo consola de comandos que sirve para controlar los dispositivos de la familia jLink.

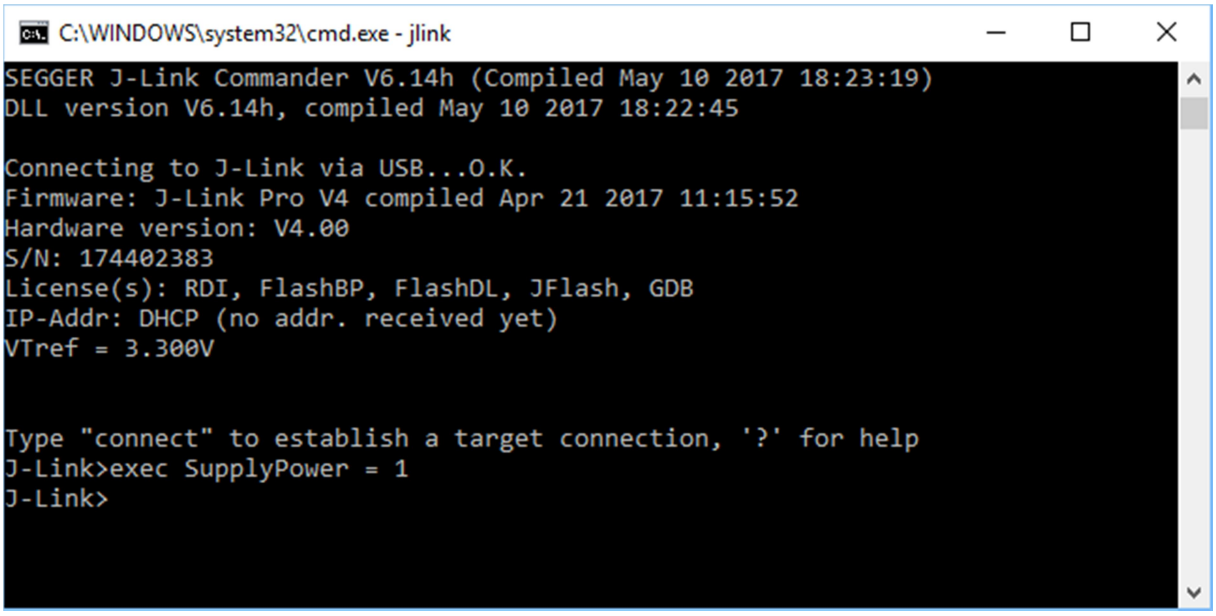


Ilustración 22 - jlink commander

- **JFlash.exe**: Esta herramienta al igual que la anterior es usada para las mismas funciones con la diferencia de que esta cuenta con una interfaz gráfica y otras facilidades para los desarrolladores.

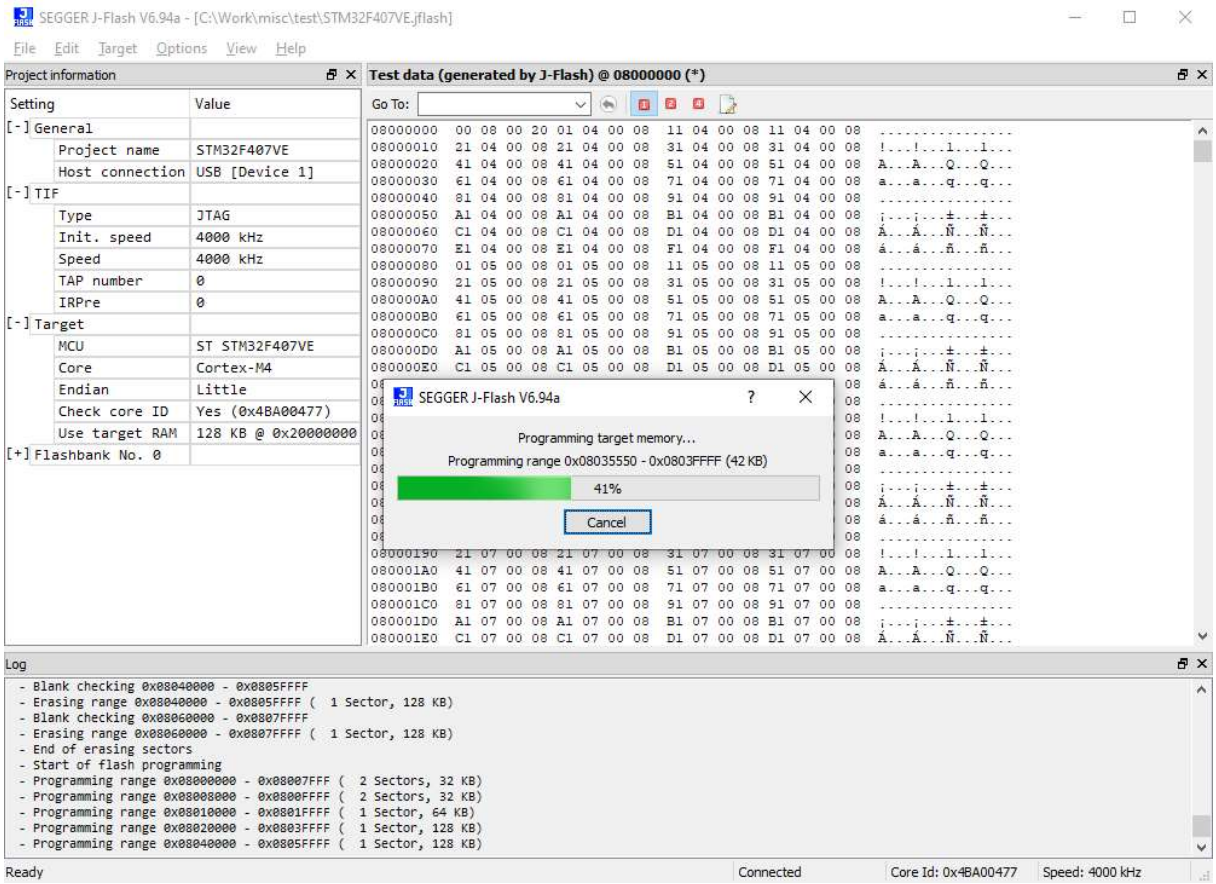


Ilustración 23 - Herramienta JFlash

Al usar este software de terceros, para ser propiamente usado desde la aplicación, se tienen que preparar ciertas herramientas previas. Para esta tarea hay que tener en consideración que, JLink BASE Compact no puede usar la herramienta JFlash anteriormente mencionada, por lo que usaremos "*jlink.exe*", este hecho produjo un retraso en la implementación ya que, con anterioridad se había usado el dispositivo jLink PRO Compact, que es parecido pero es capaz de usar JFlash, al tener que rehacer esta parte se perdió media semana entre pruebas (y un error por parte del cliente final). Para usar herramientas de tipo consola de comandos con argumentos es normal usar la librería ProcessBuilder, con la que podemos crear un proceso y ejecutar un ejecutable aparte en nuestro caso JLINK.EXE.

```
1. ProcessBuilder pb = new ProcessBuilder("PATH\\TO\\JLINK",
2.  "-device", "S32K146",
3.  "-If", "JTAG",
4.  "-speed", "4000",
5.  "-jtagconf", "-1,-1",
6.  "-autoconnect", "1",
7.  "-commandFile", "Config\\JLINK\\PATH",
8.  "-Log", "PATH\\TO\\LOG",
9.  "-ExitOnError", "1");
```

Usar ProcessBuilder es como usar la consola de comandos y teclear un comando en ella con la ventaja de que podemos monitorizar en parte el comando mediante la aplicación. Para ello vamos a desglosar qué es necesario para usar "JLINK.EXE":

- **-device <device_id>:** En la segunda línea podemos ver la opción "device" en la que definiremos el microcontrolador a programar en este caso es un S32K146.
- **-If <interface_type>:** En la tercera línea se configura el tipo de interfaz a usar, esto depende en gran medida de que conectores se usan o son compatibles con el microcontrolador en este caso usamos JTAG ("Joint Test Action Group" es normalmente usado para la depuración, ofrece una "puerta trasera" al microcontrolador) pero también es común usar SWD ("Serial Write Debug" una interfaz de depuración que ha ido sustituyendo a JTAG).
- **-speed <SPEED>:** Se utiliza para definir la velocidad de comunicación de la interfaz, parecido al baudrate que se utiliza en la comunicación serie.

- **-jtagconf <IR/DR>:** configura la localización de unos bits de prueba, en este caso en "-1,-1" para no usarlos.
- **-autoconnect <0/1>:** Usado para conectar automáticamente una vez se lance el comando. (0 -> NO, 1 -> SÍ)
- **-commandFile** <PathToCommandFile>: Si se quiere hacer algo más además de conectar con el microcontrolador es necesario usar un pequeño "script" llamado CommandFile, del cual hablaremos en breve un poco más.
- **-Log <PathToLog>:** Para llevar un control de lo que hizo la aplicación crearemos un Log donde quedara registrado si efectuó la operación de manera correcta o por el contrario ocurrió algún error inesperado.
- **-ExitOnError <0/1>:** Un flag que si es activado detendrá la ejecución del proceso cuando ocurra algún error (SEGGER, 2021)[8].

En el archivo CommandFile pueden efectuarse distintas operaciones como:

- Lecturas de memoria (mem <posMemoria>): Si fuera necesario, podríamos llevar a cabo una inspección de la memoria, por ejemplo si requiriéramos comprobar un ID localizado en una posición específica de la memoria del microcontrolador.
- Instalar un firmware (loadfile <pathToFirmware>): Para instalar el firmware se pondría en una línea del CommandFile.
- Para terminar la ejecución del proceso (qc "QuitConnection").

  A continuación se incluye un pantallazo de un archivo que conecta con el microcontrolador y comprueba una serie de localizaciones de memoria.

[8] Comandos jlink(13/10/2021) - https://wiki.segger.com/J-Link_Commander

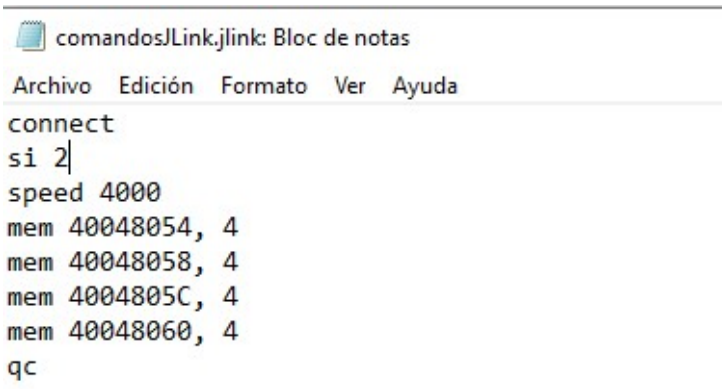


Ilustración 24 - Command File 1

En ocasiones por el contrario no hay por qué conectar con este método y se puede conectar tal y como vimos anteriormente desde la línea de comandos. Por lo que en el siguiente ejemplo se verá como los comandos sólo son usados para la instalación del firmware.

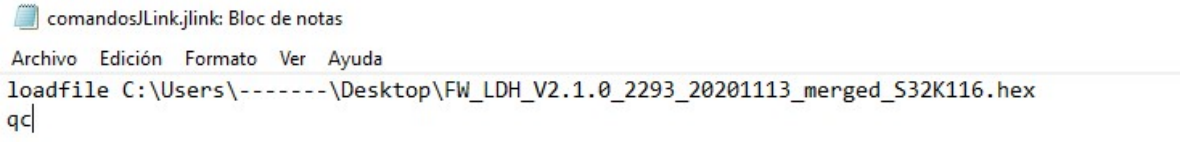


Ilustración 25 - Command File 2

En este subapartado hemos visto la investigación sobre cómo usar el software de terceros JLink.exe para instalar el firmware desde una aplicación de JAVA.

3.2 ETAPA DE DISEÑO

En esta etapa del proyecto veremos cómo se diseña una prueba, desde los datos que partimos (datos artificiales) y cómo lo entramamos para que funcione como una máquina de estados dentro de nuestra aplicación.

Para explicar cómo funciona una prueba de esta índole usaremos un ejemplo artificial:

1. El cliente nos da un documento en el que se señalan las pruebas.

2. Las pruebas viene subdivididas en unas etapas (acción, espera y lectura)

Nombre de la Interfaz	Acción	Cantidad o resultado esperado
RC10 y RC13	acción	
	espera	100 ms
RC10 y RC13	lectura	5.0V ± 0.5

En esta tabla podemos ver un ejemplo en el que activamos dos de los pinchos para hacer una lectura, en este caso activaríamos 2 de los relés de las tarjetas, uno por cada interfaz (pincho), una espera de 100ms desde la activación y por siguiente la lectura.

Cuando activamos un relé el efecto que tiene es el mismo que si colocáramos las agujas de un polímetro en la PCB para hacer la lectura.

Teniendo esto en cuenta ahora podemos ir a un ejemplo práctico:

```
1. private void prueba1() throws IOException, InterruptedExce
   ption{
2.     //Comprobar la funcionalidad de los interruptores de
   seguridad
3.
4.     boolean pruebaCorrecta = true;
5.
6.     enviarMensajeArduino(1, 2, true);
7.
8.     Thread.sleep(100);
9.
10.    enviarMensajeAFuenteAlimentacion("V1 13.5;");
11.    Thread.sleep(200);
12.
13.    enviarMensajeAFuenteAlimentacion("I1 0.5;");
14.    Thread.sleep(200);
15.    enviarMensajeAFuenteAlimentacion("I2
   0.1;");
16.    Thread.sleep(200);
17.
18.    enviarMensajeAFuenteAlimentacion("OP2 1;");
19.
20.    Thread.sleep(TIEMPO_DE_ESPERA_ESTABILIZACION);
21.
22.    enviarMensajeArduino(1, 1, true);
23.    Thread.sleep(TIEMPO_DE_ESPERA_ESTABILIZACION);
24.    actualizacionConsola.setText("COMENZANDO PRUEBA 1");
25.
26.    //Activamos relé 1 de la tarjeta 2
27.
```

```
28.    //Medimos la tension entre los puntos anteriormente
   mencionados
29.    Thread.sleep(TIEMPO_DE_ESPERA_ESTABILIZACION);
30.    client.ReadArea(S7.S7AreaDB, 1, 0, 8, Buffer);
31.    Float query_result = S7.GetFloatAt(Buffer /*BUFFER DE
   DATOS*/, 0/*NUMERO DE PALABRA*/ );//HAY QUE PASARLO A
   Voltios CUANDO SE TENGA
32.    //resultado_consulta real esperado 13.5V +/- 0.3V
33.
34.    Thread.sleep(100);
35.    if(query_result > 1.25 || query_result < 0.75){
36.        pruebaCorrecta = false;
37.        actualizacionConsola.setText("PRUEBA 1.1 FAILED");
38.    }
39.    Thread.sleep(TIEMPO_DE_ESPERA_ESTABILIZACION);
40.    enviarMensajeArduino(4, 1, true);
41.    //
42.    //YA ESTÄ ACTIVADO EL RELÉ DE ANTES
43.    //Medimos la tensión entre los puntos anteriormente
   mencionados
44.    Thread.sleep(TIEMPO_DE_ESPERA_ESTABILIZACION);
45.    client.ReadArea(S7.S7AreaDB, 1, 0, 8, Buffer);
46.    query_result= S7.GetFloatAt(Buffer /*BUFFER DE
   DATOS*/, 0/*NUMERO DE PALABRA*/ );//HAY QUE PASARLO A
   Voltios
47.    //resultado_consulta real esperado 13.5V +/- 0.3V
48.    actualizacionConsola.setText("MEDICION 2:
   " + query_result + "V.");
49.    Thread.sleep(100);
50.//Comprobación del rango de la lectura
51.
   if(query_result > 13.8 || query_result < 13.2){
52.        pruebaCorrecta = false;
53.        actualizacionConsola.setText("PRUEBA 1.2 FAILED");
54.}
55.
56.    enviarMensajeArduino(4, 1, false);
57.
58.    //PRUEBA 1 COMPLETA
59.    if(pruebaCorrecta){
60.        estadoPruebas[0] = pruebaCorrecta;
61.        actualizacionConsola.setText("PRUEBA 1 - OK");
62.    }else{
63.        actualizacionConsola.setText("PRUEBA 1 - NO OK");
64.        falloEnPruebaFuncional();
65.    }
66.
67.    //Desactivamos relé 1 de la tarjeta 2, SE HA TERMINADO
   DE MEDIR EN ESE PUNTO
68.    enviarMensajeArduino(1, 2, false);
69.    Thread.sleep(100);
70.}
```

Como vemos en este fragmento de código de la aplicación las partes que vimos anteriormente en la etapa de investigación intervienen en las pruebas.

- ***enviarMensajeArduino(Integer NTarjeta, Integer NRelé, boolean ON/OFF)*:** Es la función encargada de activar y desactivar los relés de manera transparente para facilitar la implementación de las pruebas.
- ***enviarMensajeAFuenteAlimentación(String comando)*:** Comanda la fuente de alimentación de manera que sea transparente a la hora de implementar la prueba.
- ***ReadArea() y GetFloatAt()*:** Vistos en el apartado del PLC para efectuar la lectura.
- ***Thread.sleep(ms)*:** Función para efectuar las esperas pertinentes entre etapas de una misma prueba.

Para que la aplicación pueda actualizarse a la par que se efectúan las pruebas, será necesario dividir la aplicación en el proceso de la interfaz propiamente dicha y los procesos de instalación del firmware y de las pruebas, de ahí que podamos observar el uso de Thread.sleep(ms) en las pruebas que además nos sirven para hacer las esperas pertinentes a la hora de recoger lecturas.

Hay que tener en cuenta además como funciona una máquina de estados, casi todo programa funciona casi como una máquina de estados, cada vez que tenemos una entrada en una función, podemos hacer que esta se comporte de una manera u otra, y eso propiamente dicho ya sería una “máquina de estados”. A lo que nos referimos en este contexto es a algo de más nivel, a ver las pruebas como un conjunto de estados propiamente dicho. Queremos que el programa sea capaz de parar de manera segura en cualquier punto de una prueba, de un estado. Para ello, cada ciertas operaciones haremos una comprobación de seguridad:

```
1. //COMPROBACIÓN DE HILO
2. SI !HiloPruebas.isRunning() ENTONCES
3.      FinalizarPruebas()
4.      RETURN 0
5. FIN SI
```

Aunque parece que no tiene sentido ya que las pruebas ocurren en el mismo hilo que esta operación, esta secuencia es activada cuando, por ejemplo, se pierde la conexión con la fuente de alimentación, que detiene el hiloPruebas. La función “finPruebas()” lleva a cabo una secuencia en la que abre todos los relés dejando sin

corriente a la PCB y (si hay conexión con la fuente) apaga ambas salidas de corriente.

Propiamente dicho así se conforma una prueba, como vemos, el esquema de una prueba es muy sencillo: configuramos la fuente de alimentación, activamos el/los relé/s que corresponda/n y por último se toman las oportunas mediciones, si todo está dentro de los resultados correspondientes se continúa.

En este apartado hemos visto el diseño de un simulacro de prueba, este pequeño diseño nos ayuda a ver cómo debemos maquetar el resto, seguir una pauta repetitiva que ayude a entender cómo funcionará e implementará el resto de pruebas.

3.3 ETAPA DE IMPLEMENTACIÓN

Tras terminar el diseño de las pruebas veremos cómo se unirá todo el conjunto de pruebas, junto con las tecnologías que vimos anteriormente. Después de investigar las tecnologías, probarlas por separado y diseñar una prueba como queremos que sea, quedan algunos objetivos que serán resueltos en la implementación:

- Gestión de propiedades/opciones.
- Trazabilidad.
- Gestión del proceso de pruebas.
- Gestión de diversos errores.

Para efectuar la implementación finalmente se decide hacer una aplicación muy básica, con una estructura que gira alrededor de la ventana principal, la cual lleva el grueso de la interacción y desde la que se ejecutarán los distintos hilos que llevarán al correcto funcionamiento de la aplicación.

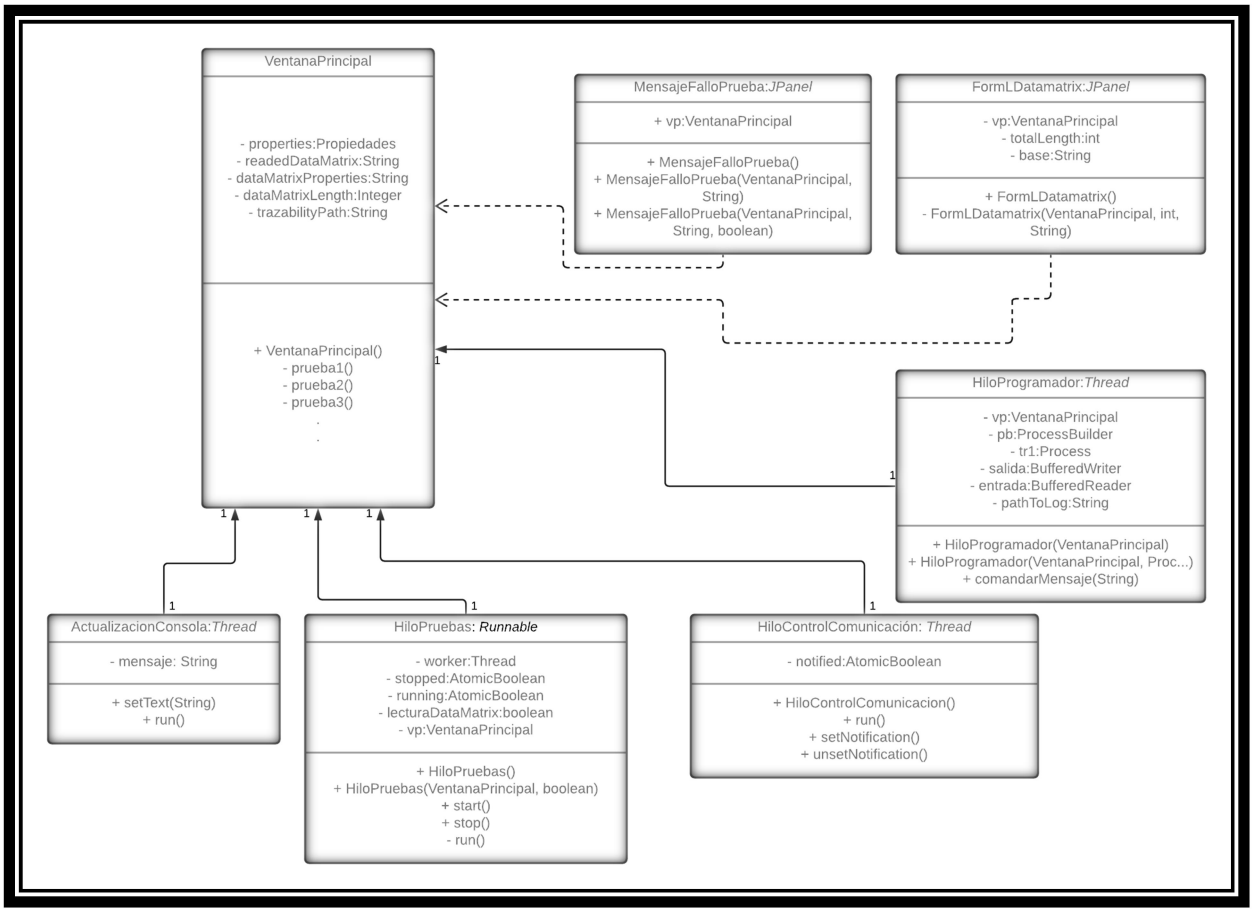


Ilustración 26 - Diagrama de Clases simplificado

3.3.1 Gestión de propiedades/opciones

Si en algún momento el software cambiara de equipo o el firmware fuera actualizado, el encargado debe de ser capaz de cambiar alguna de las configuraciones. En este caso, las opciones relevantes son:

- **Opciones de DataMatrix**: El encargado debe ser capaz de cambiar la "base" del Datamatrix, para ello esto se determina mediante dos propiedades: Longitud del Datamatrix y una plantilla del DataMatrix.
- **Direcciones respecto de JLink**: Tanto el ejecutable de JLink como el del script generado por la aplicación, deben estar escritos en este.
- **Dirección de trazabilidad**: Por otro lado, la dirección dónde se encontrará el fichero txt de la trazabilidad de las PCBs.
- **Dirección del firmware**: Por último, la dirección del firmware que se instalará en el microcontrolador.

Para llevar el registro y control de estas propiedades, se usa un fichero "*properties*", el cuál contendrá las siguientes propiedades:

- ***JLinkPath***: Ruta al ejecutable de JLink.exe.
- ***ProgramPath***: Ruta al firmware para instalar.
- ***JLinkConfigPath***: Ruta del fichero "script" para la instalación.
- ***DataMatrix***: Plantilla para comprobar la correspondencia del DataMatrix.
- ***DataMatrixL***: Longitud a comprobar del Datamatrix.
- ***TrazabilidadPath***: Ruta para el fichero de la trazabilidad.

Además del manual de usuario que se entregará al cliente, estas propiedades van acompañadas en el fichero de unas breves descripciones que ayudarán al responsable para no tener que acudir al manual de usuario.

3.3.2 Conjunto de pruebas

Sin contar la instalación del firmware, el número de pruebas semifuncionales de las que consta un test es 10. Para que todo funcione como la máquina de estados de la que hablábamos antes, cada una de las pruebas es un estado, un estado peculiar llamado "Prueba" que se reitera como tantas haya. Sin embargo, también hay que tener en cuenta que dentro de una prueba pueden ocurrir imprevistos que puedan poner en riesgo la PCB que está siendo probada, por ello es importante que las pruebas puedan pararse de un momento a otro de manera segura como se mencionó anteriormente. Todo esto tiene que ser hecho sin arriesgar el conjunto de pruebas en sí.

Para afrontar este problema las pruebas serán implementadas en una clase que heredará de la clase Thread, es decir, esta clase será ejecutada aparte de la interfaz/aplicación-principal. Además de esta particularidad, el control de comunicaciones es también un hilo que funciona de manera independiente a la interfaz y proceso.

Por otro lado, debemos llevar constancia del resultado de las pruebas, para lo que usaremos un simple vector (de valores booleanos) que lleve la cuenta de las pruebas superadas. Así al final de las pruebas mostrar el resultado si todo ha ido bien.

Las clases implementadas además de la Ventana Principal, dentro de la misma, son:

- ***HiloPruebas*** *(hereda Runnable)*: Este hilo se iniciará una vez se inicie la secuencia deseada, este hilo lleva la secuencia de pruebas tal y como vimos en los diagramas de máquina de estados.
- ***ActualizacionConsola*** *(hereda Thread)*: Este hilo por otro lado lleva a cabo la actualización de la consola, donde el operario podrá ver cómo avanzan las pruebas y las mediciones tomadas. Además también se incluye el DataMatrix que se lee y la fecha en la que se inicia el test.
- ***HiloControlComunicacion*** *(hereda Thread)*: Como dijimos previamente es el encargado de comprobar que las comunicaciones con los distintos dispositivos se llevan a cabo.

3.3.3 Trazabilidad

La trazabilidad de las pruebas es una de las funciones más importantes de las que se tienen que implementar en este tipo de software, además el cliente hizo el siguiente apunte sobre este; sólo quedarán guardadas las PCBs que tengan un OK de todas las pruebas, dado que las que no las tengan serán reparadas o desechadas y en ningún caso saldrán al mercado.

El cómo se recoge esta trazabilidad aparte de su formato, es libre, siendo la opción preferida del cliente un simple fichero formato "*csv*".

El formato de una línea de trazabilidad es el siguiente:

dd/MM/AAAA-00:00:00;Datamatrix;[FirmwareInstalado];OK

El campo de Firmware Instalado es opcional ya que, el operario puede decidir si instalar de nuevo el firmware o no en la aplicación.

La dirección al fichero de trazabilidad, como vimos es una de las propiedades, por lo que el administrador puede cambiar el nombre de este si quiere (ya que si no existe el fichero, se crea uno nuevo con el nombre).

3.4 Pruebas finales

Las pruebas finales, en este caso, se llevan a cabo con el cliente, para ello el sistema debe pasar 3 requisitos clave:

- Debe llevar a cabo todo el repertorio de pruebas de manera correcta.
- Debe instalar el firmware sin problema.
- Tiene que comprobarse que el tiempo medio de test no supere un límite.

3.4.1 Pruebas de verificación del sistema

Antes de la llegada del cliente, es necesario llevar a cabo una serie de pruebas internamente dentro de la empresa, esto es:

1. El desarrollador, yo, debo hacer sendas pruebas con PCBs cedidas por el cliente, en teoría, todas tienen el visto bueno de la producción. Ninguna debe tener un fallo en ejecución.
2. Coger una de las PCBs y hacer fallos efectivos en el transcurso de las pruebas para comprobar que los sistemas de comunicación no interfieren con la seguridad, es decir, desconectar por ejemplo la conexión Ethernet con la Fuente de alimentación para ver si todos los relés reaccionan y se apagan.
3. Una prueba con algún compañero de producción, explicarle cómo funciona el software en un máximo de 10 minutos y ver si es capaz de llevar a cabo unos test sin ninguna duda.

3.4.2 Pruebas de validación del sistema

Las pruebas de validación son llevadas a cabo por el cliente, una vez llevadas a cabo las anteriores pruebas, llevamos la cama de pinchos al cliente y junto con el se hace una demostración con unas pocas PCBs. Una vez se lleva a cabo, un operario real tomará el control y de la misma manera que hicimos de manera interna se le explica el funcionamiento del software y se deja llevar a cabo un par de comprobaciones. Una vez el responsable dé el visto bueno, se da por hecho que el proyecto ha sido entregado y terminado.

4 CONCLUSIONES Y TRABAJOS FUTUROS

Como hemos visto a lo largo de la memoria, los objetivos que se plantean han sido satisfechos de una manera eficaz, proporcionando al cliente un software de calidad y fácil de usar por sus empleados.

De este proyecto se aprende la complejidad de trabajar y unificar distintas tecnologías así como investigar sobre estas, marcando así pautas a seguir en futuros casos.

Aprendemos la importancia de una investigación previa de las tecnologías que intervienen para evitar posibles desastres en la implementación, como hubiera sido no probar por separado la tarjeta de relés o la comunicación con el PLC.

Como principal conclusión, que podemos sacar sobre estos trabajos, es la alta flexibilidad que debe tener un programador a la hora de abordar los problemas, sin dejar de lado la participación activa que tiene que tener dentro de una solución multidisciplinar como es el caso, teniendo comunicación, siempre que se pueda, tanto con los compañeros del proyecto como con el cliente final.

Este proyecto, sin embargo, no queda exento de posibles mejoras, por un lado, es posible mejorar la flexibilidad del software dando lugar a una aplicación que se pudiera configurar en distintas camas de pinchos, dando lugar a un producto independiente.

Por otro lado, la gestión de los errores, aunque efectiva, puede ser más sofisticada estudiando más exhaustivamente los casos.

Siendo la interfaz simple, es posible mejorando el sistema de propiedades que podría implementarse dentro de la misma con algún sistema de usuario-contraseña para que el operario común no pueda cambiar estas propiedades.

En definitiva, la solución propuesta es un éxito que puede seguir mejorando.

5 APÉNDICES

5.1 Guía original del Trabajo Fin de Título

PROPUESTA DE TRABAJO DE FIN DE GRADO
GRADO EN: Grado en Ingeniería Informática **ESPECIALIDAD:** General **MENCIÓN:** General
TÍTULO DEL TRABAJO: Software de pruebas para cama de pinchos **Idioma:** Castellano
DESCRIPCIÓN CORTA DEL TFG: Las placas de prueba para circuitos integrados (PCB test fixture), o “camas de pinchos”, como se suelen denominar en el ámbito industrial, constituyen una herramienta esencial para comprobar el buen funcionamiento de microcontroladores durante el proceso de su fabricación. Generalmente, este proceso de prueba puede automatizarse por medio de diferentes soluciones software. En este trabajo fin de grado, se propone el diseño e implementación de una herramienta software para la programación de un microcontrolador y la automatización de los procesos de pruebas semifuncionales sobre el mismo, optimizando dicho software para una máxima usabilidad y eficiencia en su aplicación, medida como el número circuitos impresos testados en un determinado espacio de tiempo.

- **Objetivos del TFG**:
    1. Desarrollo de una herramienta software para la programación de microcontroladores.
    2. La aplicación debe ser extensible y fácil de usar y comprender por parte del usuario.

3. Implementación de los mecanismos necesarios para realizar pruebas semifuncionales sobre circuitos impresos.

- **Metodología a Desarrollar**:

1. Análisis del problema y definición de los requisitos necesarios.
2. Estado del arte: estudio de soluciones similares a la que se desea desarrollar
3. Diseño e implementación de la herramienta software.
4. Integración de la herramienta en un proceso industrial de desarrollo y prueba de circuitos impresos.
5. Preparación de un manual de usuario final.
6. Todo el proceso anterior quedará convenientemente registrado y documentado en una memoria escrita junto con los manuales y apéndices necesarios.

5.2 Programa Arduino MEGA

Como vimos en el apartado de implementación, *3.1.1*, el problema causado por las tarjetas de relés conllevó el desarrollo de una solución basada en Arduino MEGA para el control de esta. En este apartado se explicará de manera breve la solución adoptada y se explicará así el funcionamiento.

Como es de costumbre en las implementaciones orientadas a Arduino, el programa está dividido entres partes: la declaración de variables, el setup() (configuración del Arduino, normalmente) y la función loop(), la cuál ejecutará el funcionamiento normal del Arduino.

```
// CÓDIGO PARA ARDUINO MEGA, ATmega 2560
byte cad;

void setup() {

  for(int i = 22; i<54;i++){
    pinMode(i, OUTPUT);
    digitalWrite(i, HIGH);
  }
  Serial.begin(9600);

}

void loop() {
  // put your main code here, to run repeatedly:
  if (Serial.available()>0){
    //leemos la opcion enviada
    cad=Serial.read();
    byte resultado = cad & 0x7F;
    if(resultado > 21 && resultado < 54) {
      digitalWrite(resultado,  !(cad>>7));
      Serial.print(cad);
    }

    if(cad==1){
      Serial.print(cad);
    }

  }
}
```

Ilustración 27 - Código tarjeta de relés Arduino

- Como ya se vio, en el apartado 3.1.1, es necesaria una trama byte que leerá el Arduino para saber la acción que llevar a cabo (activar o desactivar un relé) y el "id" del relé, en el código podemos verlo como la declaración "***byte cad;***".
- En el **setup()** por otro lado vemos como se configuran cada uno de los pines que intervienen en esta pequeña aplicación, configurando cada uno de los pines [22, 53] como salidas digitales y activando cada una de ellas. Por otro lado, vemos que la configuración de la comunicación es el *baudrate* típico de 9600.

- Por último, la función ***loop()*** será la encargada de comprobar el puerto serie del Arduino por donde entrarán los mensajes para activación/desactivación de los ya mencionados relés. El byte que envía la aplicación principal al Arduino, se desglosa de la siguiente manera:

La línea *byte resultado = cad & 0x7F* es la encargada de sacar el número de relé mediante la operación lógica AND aplicada a un byte

(Ejemplo: (byte IN) - 1 0 0 0 1 0 0 1 & (0x7F) - 0 1 1 1 1 1 1 1 = 0 0 0 0 1 0 0 1)

Como vemos el primer 1 del byte queda "eliminado" dejando así un byte que marcaría el pin/relé que activar.

El siguiente paso una vez comprobado que el byte entra dentro del rango de pines que podemos activar es activarlo o desactivarlo, esto se comprueba observando el bit más significativo de la trama cad. Esto se consigue mediante la operación *!(cad >> 7)* que desplaza todos los bits 7 veces a la derecha dejando el último como un valor de 1 o 0 (la operación de negación es porque las tarjetas están activas a "negativos", es decir, cuando es 0 se activa y cuando es 1 se desactiva.

Como última mención, para comprobar la conexión correcta con el Arduino se comprueba si la trama byte es 0x01, de ser así se devuelve la trama para comprobar que la comunicación ha sido exitosa.

5.3 Manuales de usuario

El equipo queda entregado con el software ya instalado y preparado para trabajar con él, por ello sólo se incluye un breve documento para saber configurar desde el fichero de propiedades, una breve descripción de elementos de la interfaz y cómo se conecta el equipo para ser funcional. El manual de usuario queda incluido con los demás documentos de la entrega con el nombre de ManualDeUsuario.pdf. Por confidencialidad, la portada y algunos nombres del documento han sido eliminados.

6 DEFINICIONES Y ABREVIATURAS

PCB (Printed Circuit Board): Traducido al español como "**placa de circuito impreso**", es un circuito electrónico constituido por pistas de algún material conductor que interconecta diversos componentes electrónicos como microcontroladores, resistencias, diodos, etc.

PLC (Programmable Logic Controller): Traducido al español como "controlador lógico programable", también conocido como **autómata industrial** es un dispositivo muy parecido al PC común, sin embargo su diseño está preparado para la industria y el control de distintos dispositivos mediante entradas/salidas.

7 BIBLIOGRAFÍA

Hedgecock, W. (14 de 10 de 2021). *jSerialComm*. Obtenido de fazecast jSerialComm: https://fazecast.github.io/jSerialComm/

Nardella, D. (14 de 10 de 2021). *Moka7 deploy*. Obtenido de Snap7: http://snap7.sourceforge.net/moka7.html

National Instruments. (14 de 10 de 2021). *National Instruments*. Obtenido de National Instruments: https://www.ni.com/es-es.html

Pressman, R. (2010). *Ingeniería del Software.* McGraw-Hill.

SEGGER. (10 de 11 de 2021). *wiki de segger*. Obtenido de JLink Commander de SEGGER: https://wiki.segger.com/J-Link_Commander

Soloaga, A. (19 de 08 de 2018). *Principales Usos de Python*. Obtenido de https://www.akademus.es/: https://www.akademus.es/blog/programacion/principales-usos-python/

SRC. (15 de 10 de 2021). *¿QUÉ ES UN PLC? ¿CÓMO FUNCIONA? ¿PARA QUÉ SIRVE?* Obtenido de SRC: https://srcsl.com/que-es-un-plc/

Universidad de Cantabria. (14 de 10 de 2021). *Software Labview*. Obtenido de https://sdei.unican.es/: https://sdei.unican.es/Paginas/servicios/software/Labview.aspx

Universidad de Jaén. (8 de 09 de 2004). *https://intranet.ceautomatica.es/.* Obtenido de Intranet: https://intranet.ceautomatica.es/old/actividades/jornadas/XXV/documentos/146-uanesjaezo.pdf