Trabajo Final de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Diseño de tests unitarios para el TWR-KM34Z50M

Autor: Daniel Neira Jaén

Tutor: Ramón González Carvajal

Co-tutor: Rafael Girona García

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Final de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Diseño de tests unitarios para el TWR-KM34Z50M

Autor:

Daniel Neira Jaén

Tutor:

Ramón González Carvajal

Profesor Catedrático de Universidad

Co-tutor:

Rafael Girona García

Personal Investigador en Formación

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Final de Grado: Diseño de tests unitarios para el TWR-KM34Z50M

Autor: Daniel Neira Jaén
Tutor: Ramón González Carvajal
Co-tutor: Rafael Girona García

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Me ha costado escribir estas líneas porque suponen el cierre a cuatro años de una etapa de mi vida dura pero a la vez muy bonita. En primer lugar, darle las gracias a Ramón y a Rafael Girona (cienti) por todo en general, tanto del proyecto como fuera, por ayudarme y aconsejarme sobre el mundo de los embebidos y por tranquilizarme en mis momentos de agobio viendo las fechas.

A mi familia, a mi hermana, mis padres y mis abuelos, por aguantarme en momentos de presión y darme apoyo cuando más lo necesitaba, de verdad, gracias.

A Ignacio, Ana, Pablo, Jesús, Fernando, Sergio, José Luis y Juanje, por los momentos previos a un examen difícil, por las prácticas complicadas y por alguna que otra clase menos interesante, en definitiva, por acompañarme en este camino. No me puedo olvidar tampoco de esos ratitos de café y merienda en las máquinas o las comidas en el Gimnasio, no sabéis cuánto los echo de menos.

A Patricio, Pablo y Juan Carlos por ser los mejores compañeros de piso que pude tener, por vuestros consejos, por esas conversaciones sobre la vida durante la noche y por nuestros debates sobre el fútbol, al fin y al cabo, por hacerme sentir como en casa.

A mis maestros y profesores, sin ellos no hubiera sido posible.

En definitiva, a todos los que de uno u otro modo habéis hecho que esté hoy aquí, gracias de corazón.

Daniel Neira Jaén

Sevilla, 2020

Resumen

El avance imparable de la tecnología nos lleva a incorporarla a mundos en los que hasta hace relativamente poco resultaba prácticamente impensable. Nos referimos al entorno de los contadores o medidores de energía, ya sean de electricidad, agua o gas. Atrás quedaron esos contadores analógicos donde había que apuntar a mano las lecturas. Hoy en día, ya es posible automatizar las medidas y enviar los datos de forma totalmente segura a los servidores de la empresa, pero siempre se pueden ir incorporando mejoras y actualizaciones.

En ese sentido, como todos los desarrollos, es necesario empezar por el principio, por eso en este trabajo nos centraremos en el estudio del microcontrolador TWR-KM34Z50M y en la realización de tests unitarios para entender su funcionamiento y aprender cómo hacerlos. Este micro está diseñado especialmente para su uso en medidas y posee además elementos de antisabotaje.

En este trabajo comenzaremos especificando las herramientas software elegidas para llevarlo a cabo. Continuaremos con una descripción del hardware, el ya mencionado TWR-KM34Z50M, posteriormente expondremos la metodología de los tests para luego explicarlos uno a uno. Por último, justificaremos los resultados para concluir con las líneas futuras de desarrollo.

Abstract

The unstoppable advance of technology leads us to incorporate it into worlds in which until relatively recently it was practically unthinkable. We refer to the environment of energy meters, whether they are electricity, water or gas. Gone are those analog counters where you had to write down the readings by hand. Nowadays, it's already possible to automate the measurements and send the data in a totally secure way to the company's servers, but improvements and updates can always be incorporated.

In that sense, like all developments, it is necessary to start at the beginning, so in this work we will focus on the study of the TWR-KM34Z50M microcontroller and on performing unit tests to understand its operation and learn how to do them. This micro is specially designed for its use in measurements and also has anti-tamper elements.

In this work we will begin by specifying the software tools chosen to carry it out. We will continue with a description of the hardware, the already mentioned TWR-KM34Z50M, later we will expose the methodology of the tests and then explain them one by one. Finally, we will justify the results to conclude with future lines of development.

Índice Abreviado

Índice

1 Introducción

If you want to go quickly, go alone. If you want to go far, go together.

African Proverb

En la actualidad, donde un nuevo mundo digital da cada vez más importancia a la información y recabado de datos, surgen nuevos sistemas compuestos autónomos y nuevas corrientes, como la de Internet Of Things (IoT), que exigen una mayor calidad de dichos datos junto a una mayor capacidad de adaptación de los dispositivos que los manejan.

En respuesta a esta demanda, nacen nuevos microcontroladores donde priman la calidad, precisión, seguridad y resistencia al desgaste entre otros. Estos micros deben tener la capacidad de manejar los nuevos sensores y actuadores cada vez más precisos, así como su ingente cantidad de datos.

Todo esto motiva el reemplazo de los procedimientos manuales por nuevos procesos automatizados más eficientes, rápidos y robustos que tengan la capacidad para reaccionar ante situaciones adversas sin perder sus capacidades, y en caso de hacerlo, recuperarlas a la mayor celeridad.

1.1 Objetivos

La finalidad global de este proyecto es la introducción al mundo del Test-Driven Development (TDD) orientado a entornos embebidos, mediante la adquisición de una metodología de trabajo con un foco muy fuerte en la automatización, verificación y documentación.

Para llevar a cabo dicha tarea se establecen los siguientes objetivos específicos:

- Aprender y usar Linux como sistema operativo orientado a desarrollo de software y firmware.
- Adoptar rutinas de trabajo basadas en el uso de sistemas de control de versiones.
- Introducir el manejo de IDE con capacidades de depuración sobre hardware.
- Crear una estructura software orientada a TDD y que dé soporte a hardware.
- Elaborar los tests unitarios necesarios para validar el diseño.
- Elaborar una estrategia para la aplicación en un sistema de Continuous Integration/Continuous Deployment (CI/CD).
- Aplicar lo anterior orientándolo a embebido sobre una placa de evaluación, concretamente la TWR-KM34Z50 de NXP.
- Aplicar lo anterior orientándolo a documentación, concretamente el propio documento del TFG.

Los tests unitarios son formas de verificar el funcionamiento de partes o unidades de código. Los tests deben poder realizarse de manera independiente, de forma que cada uno pruebe una parte específica del código. Esto permite crear listas de tests que, una vez ejecutados secuencialmente, llevarían a validar por completo el diseño y desarrollo de un sistema. La idea final del proyecto es poder aplicarlo en un sistema de CI/CD, de forma que cada vez que hagamos un nuevo commit se ejecuten los tests y se verifique que todo funciona correctamente, terminando con la generación y despliegue de los artefactos definidos.

1.2 Revisión de las herramientas disponibles

A continuación se expondrá un estudio de estado actual de cada objetivo específico, sobre el que se basará de forma total o parcial la elección final de cada una de las partes.

1.2.1 Sistema operativo de trabajo

Aunque en algunos casos Windows puede dar resultados satisfactorios, en líneas generales no es un sistema orientado a desarrolladores ni herramientas open-source, y como además el proyecto tendrá cierto trabajo sobre servidores que no usan Windows, no se estudia más a fondo su posible viabilidad.

Mac por otra parte sí que soporta de forma nativa la mayoría o todos los puntos tratados en el proyecto, por lo que sería una opción más que viable.

Finalmente Linux es el sistema que por lo general está más que preparado para soportar trabajos de desarrollo, automatización, compilación e interacción entre diferentes máquinas de forma nativa. Tiene un amplio soporte software y hardware, además de una gran comunidad asentada de usuarios que facilitan el acceso a la información y solución de problemas.

La elección final será Linux, y entre las diferentes distribuciones disponibles, se escoge Debian 10 (Buster) por ser una de las más asentadas, estables y seguras, aparte de estar orientada a servidores, algo que se busca de cara a la automatización de pruebas - CI/CD. Esto además abre la puerta a trabajar con múltiples sistemas bajo una única base de conocimiento, como pueda ser el salto entre PC y un sistema embebido tipo Beaglebone, sin necesitar por ello una nueva curva de aprendizaje.

1.2.2 IDE de trabajo

En este caso el IDE viene impuesto por el fabricante y se trata de Kinetis Design Studio.

Este IDE está basado en Eclipse y su plugin de embebidos, al que añade las funcionalidades necesarias para trabajar con la placa y micro escogidos.

Por desgracia, es un IDE iniciado por Freescale y que heredó NXP, quienes no mantienen ningun tipo de soporte. Esto ha dificultado el acceso a la información y, por lo tanto, ha creado una curva de aprendizaje mas escarpada.

Por suerte, una vez configurado y funcionando, ha demostrado ser un sistema estable y confiable, que permite el flujo completo de desarrollo, compilación, flasheado y depuración de una forma fácil y eficiente.

1.2.3 Control de versiones

Conforme vislumbramos la complejidad del proyecto, se hizo necesario el uso de un control de versiones. En el mercado actual existen dos grandes competidores en ese ámbito : SVN y Git.

SVN y Git poseen una diferencia bastante significante. SVN es un sistema de control de versiones centralizado o Centralized Version Control System (CVCS), mientras que Git es un sistema de control de versiones distribuido o Distributed Version Control System (DVCS).

Un sistema de control de versiones centralizado opera con la idea de que hay una sola copia del proyecto donde los desarrolladores realizan el commit de los cambios, y un solo lugar donde se guardan todas las versiones de un proyecto.

Un sistema de control de versiones distribuido, por otro lado, trabaja con el principio de que cada desarrollador *clona* el repositorio del proyecto al disco duro de su dispositivo. Se guarda una copia del proyecto en cada ordenador local, y los cambios deben ser subidos (pushed) y bajados (pulled) hacia y desde el repositorio online para actualizar la versión que cada desarrollador tiene en su equipo local.

Ventajas de Git

- **Sistema Distribuido**: El hecho de ser un sistema distribuido significa que múltiples repositorios redundantes y ramificaciones son conceptos de la herramienta. En un sistema distribuido como Git, cada usuario tiene una copia completa guardada del proyecto, haciendo que el acceso a la historia de cada uno sea extremadamente rápido. Debido a ello, permite que puedas utilizarlo sin conexión a Internet. También significa que cada usuario tiene una copia completa del repositorio, es decir que si por algún caso alguno queda corrupto, sólo se perderán los cambios que no estén subidos al servidor. En SVN por otro lado, solamente el repositorio central contiene la historial completa. De forma que, los usuarios deben comunicarse a través de la red al repositorio central para obtener el historial de los archivos. Por otro lado, si el repositorio central se pierde por algún fallo del sistema, éste debe ser

restablecido desde un backup y esto puede provocar que algunos cambios se pierdan. Dependiendo de las políticas de backup que hayan, podrían incluso perderse semanas de trabajo.

- **Control de Acceso**: Al ser un sistema distribuido, no hay que otorgar acceso a otras personas para que puedan utilizar las funciones de control de versiones. En vez de eso, es el dueño del repositorio el que decide a qué cambios realizar el merge y de quién. En Git, los usuarios pueden tener control de versión de su propio proyecto, mientras que el proyecto principal está controlado por el propietario del repositorio.
  SVN por otro lado sí lo controla, por ejemplo, el usuario requiere acceso para realizar commits.
- **Manejo de Ramas (Branches)**: En Git utilizar branches es muy común y fácil, mientras que se puede considerar que en SVN es un proceso un poco más engorroso y no tan habitual. De hecho, la queja más común sobre SVN es lo tan tedioso que es trabajar con ramas. En SVN, las braches se crean como directorios, algo que a muchos desarrolladores no les gusta.
  Además en SVN 1.6, se introdujo un concepto llamado conflicto de árboles, causados por cambios en la estructura de los directorios. Ocurre con frecuencia y debido a ello, realizar commits entre branches se vuelve más complejo.
- **Rendimiento**: Dado que Git trabaja con un repositorio local, no hay latencia en la mayoría de las operaciones, exceptuando push y fetch, donde sí necesita conectarse al repositorio central.

Ventajas de SVN

- **Historial Permanente**: SVN siempre obtendrá exactamente la misma información del repositorio, tal y como estaba en el pasado. Así como también puede rastrear todos los cambios de un archivo o carpeta, debido a que el historial en SVN es permanente.
  En Git por otro lado, puede que se pierda el historial de un archivo/directorio. Git no se preocupa del rastro o la historia precisa de cada archivo en los repositorios. Por ejemplo, renombrar un archivo o el comando «git rebase» hace difícil encontrar el historial «verdadero» de los ficheros.
- **Un único repositorio**: Dado que SVN solamente permite tener un repositorio, no debemos preocuparnos por dónde se guarda algo. En caso de necesitar un backup o querer buscar algo, no nos quedará duda de que todo lo que necesitemos se encuentra en el repositorio central.
  Dado que Git trabaja con repositorios distribuidos, puede que sea más dificil saber dónde están ubicados algunos ficheros.
- **Control de acceso**: Dado que SVN tiene un repositorio central, es posible especificar allí el control de lectura y escritura y será forzado en todo el proyecto.
- **Archivos Binarios**: Los sistemas de control de versiones tienen como idea que la mayoría de los archivos que serán versionados son fusionables. Es decir, que debería ser posible funsionar dos cambios simultáneos realizados en un archivo. Este modelo es llamado Copy-Modify-Merge y tanto Git como SVN lo utilizan. El único problema es que esto generalmente no es aplicable a archivos binarios, y es por ello que SVN ofrece soporte para el modelo Bloquear-Modificar-Desbloquear para estos casos.
  Por otra parte Git no admite bloqueos de archivos exclusivos, lo que complica la labor en empresas con proyectos donde existen muchos archivos binarios no fusionables. En caso de querer utilizar Git con archivos binarios, hay que especificar cuáles de ellos lo son.

Después de todo lo expuesto, nos decantamos por Git porque no vamos a usar demasiados archivos binarios que es su principal hándicap, y porque una vez superada su complejidad inicial, es una herramienta más potente y flexible.

Una vez elegido Git, vamos a usar herramientas basadas en él como GitLab y GitKraken. Este conjunto de software nos facilitará las cosas, por un lado la web de GitLab para tener acceso al repo y por otro, GitKraken, con su interfaz gráfica y la posibilidad de ver las ramas y subir nuevos commits.
Existen otros modos de nombrar las versiones del proyecto, pero dentro de Git aparece Git Flow junto con SemVer, que serán los que usaremos.

1.2.4 Tests unitarios

Los tests unitarios son pruebas que se realizan sobre cada módulo de software de manera independiente. El objetivo es comprobar que el módulo, entendido como una unidad funcional de un programa independiente, está correctamente codificado.

En general, un módulo se entiende como un componente software que cumple las siguientes características:

- Debe ser un bloque básico de construcción de programas.
- Debe implementar una función independiente simple.
- Podrá ser probado al cien por cien por separado.
- No deberá tener más de 500 líneas de código.

Los tests deben estar optimizados para ser rápidos y eficaces. Es preciso que se pruebe la totalidad del código con este tipo de pruebas. Si no se cumple esto, habrá una parte del software que estará sin probar y la probabilidad de resultados inesperados en la ejecución será exponencialmente más alta que si probamos todo o casi todo el código. Las pruebas unitarias tienen que ejecutarse aisladamente unas de otras y la ejecución de un test no debe influir en otro. Estas características que se han descrito están recogidas en el principio FIRST (Fast, Isolation, Repeteable, Small y Transparent). Si no se cumplen estas propiedades existe el riesgo de que un test unitario no sea válido.

Las herramientas de código abierto más conocidas y utilizadas para ejecutar este tipo de pruebas son:

- NUnit
- JUnit
- TestNG
- DBUnit
- Sahi
- XUnit
- Jmock
- Cgreen

Las herramientas comerciales más famosas son:

- Parasoft C/C++ TEST
- Typemock
- AgitarOne

Estas herramientas así como el uso de tests unitarios en general está extendido en el desarrollo de software de alto nivel, no así en sistemas embebidos, debido principalmente a los siguientes factores :

- **Complejidad de toolchains**: Disponer de un entorno común para desarrollar de manera sencilla los tests y poder ejecutarlos en la placa sería lo ideal pero no siempre es así. Depender de hardware específico y de herramientas propias de cada fabricante complica el desarrollo de los tests.
- **Lenguaje de programación C**: C es el lenguaje más usado en los sistemas embebidos y hace que el desarrollo y la ejecución de los tests sea más compleja que en un lenguaje orientado a objetos.
- **Motivos históricos**: Hasta hace relativamente poco, los desarrolladores de embebidos provenían de otras ramas de ingeniería y no solían tener demasiada formación específica en cuanto a la programación. En el momento en que el desarrollo se volvió más complejo, resultó imprescindible usar herramientas como el control de versiones, técnicas orientadas a objetos, etc … muy usadas en software de alto nivel pero no en embebidos. Es entendible por tanto que en esta evolución vaya por detrás.

A pesar de los motivos expuestos, también existen herramientas para el desarrollo de test unitarios en dispositivos embebidos, de las que se destaca *Ceedling*.

Ceedling es un sistema de compilación para proyectos C que es similar a una extensión del sistema de compilación Ruby's Rake (basado en *make*). *Ceedling* está dirigido principalmente al desarrollo basado en pruebas en C y está diseñado para unir CMock, Unity y CException, todo esto sin perder las capacidades sobre entornos embebidos.

Para este proyecto hemos querido desarrollar un pequeño sistema propio, que sirva para aprender e interiorizar cada parte del flujo o cadena de trabajo, de forma que una vez se dé el salto a herramientas más completas se comprenda todo de una forma más asentada y con más background.

1.2.5 Microcontrolador

En cuanto al microcontrolador, en primer lugar hicimos una búsqueda amplia en las principales empresas que fabrican microcontroladores : Microchip, Atmel, Freescale, Analog-Devices, Texas Instrument, ST, etc…, quedándonos finalmente con 3 micros : STPM34 de ST, ADE7880 de Analog-Devices y TWR-KM34Z50M de Freescale (NXP):

- **STPM34 de ST**: El STPM3x es una familia ASSP diseñada para la medición de alta precisión de potencia y energías en sistemas de líneas eléctricas que utilizan la bobina Rogowski, el transformador de corriente o resistencias shunt. El STPM3x proporciona formas de onda instantáneas de voltaje y corriente y calcula los valores RMS de voltaje y corriente, potencia activa, reactiva y aparente y energías. El STPM3x es una familia de circuitos integrados de señal mixta que consta de una sección analógica y una digital. La sección analógica consta de hasta dos amplificadores de baja compensación y bajo ruido de ganancia programable y hasta cuatro Analog to Digital Converter (ADC) Sigma-Delta (SD) de 24 bits de segundo orden, dos referencias de voltaje de banda prohibida con compensación de temperatura independiente, un regulador de voltaje de baja caída y búfer de CC. La sección digital consta de una etapa de filtrado digital, un DSP cableado, DFE a la entrada y una interfaz de comunicación en serie (UART o SPI). El STPM3x es totalmente configurable y permite una rápida calibración del sistema digital en un solo punto sobre todo el rango dinámico actual.
- **TWR-KM34Z50 de Freescale (ahora NXP)**: Es un micro basado en el núcleo ARM Cortex-M0 + de 32 bits con Analog Front End (AFE). Las velocidades de reloj de la CPU en estos dispositivos pueden alcanzar hasta 50 MHz. La familia de dispositivos KM incluye ADC SD de alta precisión, amplificador de ganancia programable (PGA), referencia de voltaje interno de alta precisión, memoria flash, RAM, bloque lógico de compensación de fase y otros periféricos. Está diseñado para ser usado en metrología, a parte de su potente AFE, también posee características de seguridad como su antitampering, unidad de protección de memoria, generador de números aleatorios y su unidad de aceleración criptográfica mapeada en memoria (MMCAU) para el cifrado Advanced Encryption Standard (AES). Permite un control muy fino en el ajuste de los filtros para realizar las medidas.
- **ADE7880 de Analog-Devices**: El ADE7880 es un micro de medición de energía eléctrica trifásico de alta precisión con interfaces seriales y tres salidas de pulso flexibles. El dispositivo ADE7880 incorpora ADC SD de segundo orden, un integrador digital, circuitos de referencia y todo el procesamiento de señales necesario para realizar mediciones de energía aparente, cálculos rms, así como de energía activa y reactiva.
  Además, el ADE7880 calcula el valor eficaz de los armónicos en las corrientes de fase y neutro y en los voltajes de fase, junto con las potencias activa, reactiva y aparente, y el factor de potencia y la distorsión armónica en cada armónico para todas las fases. La distorsión armónica total (THD) se calcula para todas las corrientes y tensiones. Un procesador de señales digitales (DSP) de función fija ejecuta este procesamiento de señales. El programa DSP se almacena en la memoria ROM interna.

Una vez expuestas todas las características, nos decantamos por el TWR-KM34Z50M de Freescale (NXP) por el alto grado de control alcanzable en su metrología, y por sus capacidades de configuración. A través de la herramienta de configuración proporcionada por el fabricante podemos ajustar los filtros para obtener las medidas necesarias de la forma deseada.

Otras familias de micros ofrecen opciones de una gran calidad en cuanto a los resultados, pero que no permiten tanta o ninguna configuración sobre los periféricos de metrología, limitando su configuración prácticamente solo a la calibración necesaria.

1.3 Alcance

Antes de comenzar el proyecto, se llevó a cabo un estudio inicial de diferentes microcontroladores y sus capacidades de metrología, seleccionando uno que cumplía con nuestras necesidades mínimas.

Tras esta selección preliminar, analizamos lo que provee el fabricante. Destacamos que el diccionario de funciones no es estándar, si no específico de este micro o compañía, algo muy común de cualquier micro que no soporte CMSIS.

Siguiendo con la idea inicial de que los tests fueran reutilizables, decidimos crear una capa de abstracción a modo de librería de microcontroladores (que llamaremos *mculib*) en la que definir las diferentes partes, como puedan ser: gestión de GPIOs, módulo de comunicaciones, metrología, etc…

Dentro de cada módulo, diseñamos drivers que sirven de nexo de unión entre esa nueva Application Programming Interface (API) (diccionario de funciones de *mculib*) y la capa que exista por debajo (un mock, este micro u otro diferente).

La parte reutilizable es todo lo que gira en torno a los drivers, menos los drivers en sí, que son específicos para cada microcontrolador concreto. De esa forma, atacar una simulación (mock), un hardware u otro, solo dependerá de cambiar el driver (y su configuración), pero no la aplicación ni el resto de capas.

2 Descripción del entorno de desarrollo

2.1 Debian

Debian es una distribución software libre basada en el núcleo Linux. Esta distribución está formada por un gran número de paquetes. Cada paquete contiene ejecutables, scripts, documentación e información de configuración y tiene un *encargado*, quien es el principal responsable de mantener el paquete actualizado, hacer un seguimiento de los informes de fallo y comunicarse con los autores principales del programa empaquetado. La gran base de usuarios en conjunto con el sistema de seguimiento de fallos aseguran que los problemas se encuentren y resuelvan rápidamente.

Debian introdujo muchas características a Linux, que ahora son comunes, por ejemplo, Debian fue la primera distribución de Linux en incluir un sistema de gestión de paquetes para permitir una instalación y desinstalación fácil del software. Además, fue la primera que podía actualizarse sin necesidad de una reinstalación.

El proyecto Debian comenzó en 1993, cuando Ian Murdock hizo una invitación a todos los desarrolladores de software a contribuir a una distribución completamente coherente basada en el, entonces nuevo, núcleo Linux. Este grupo relativamente pequeño, al principio patrocinados por la Free Software Foundation e influenciados por la filosofía GNU, ha crecido a lo largo de los años hasta convertirse en una organización de alrededor de 1026 desarrolladores Debian. Los desarrolladores Debian están involucrados en una gran variedad de tareas, incluyendo la administración de la web y FTP, diseño gráfico, análisis legal de licencias software, escribir documentación y, por supuesto, mantener paquetes de software.

2.2 Rufus

Para poder instalar Debian en nuestro ordenador vamos a hacer uso de Rufus, una aplicación también de código libre para crear unidades USB de arranque o booteables. Es especialmente útil para:

- Crear medios de instalación USB a partir de ISOs arrancables (Windows, Linux, UEFI, etc.)
- Trabajar en un equipo que no tenga un sistema operativo instalado.
- Actualizar el firmware o BIOS de un ordenador desde DOS.
- Ejecutar una utilidad de bajo nivel.

Rufus también tiene como ventaja su velocidad, es casi dos veces más rápido que UNetbootin, Universal USB Installer o la herramienta de descarga a USB de Windows 7, en la creación de un instalador USB de Windows desde una ISO. También es ligeramente más rápido en la creación de USB de arranque de Linux a partir de ISO, que es nuestro caso.

Como hemos apuntado anteriormente, para instalar Debian primero se necesita Rufus por lo que se descarga Rufus siguiendo los pasos de su web, luego, usándolo, se copia en un pen-drive una ISO de Debian 10. A continuación, se reinicia el ordenador con el pen-drive pinchado y se accede al menú de arranque. Se cambia la prioridad para arrancar primero con la imagen de Debian del pen-drive y se siguen las instrucciones de configuración de Debian.

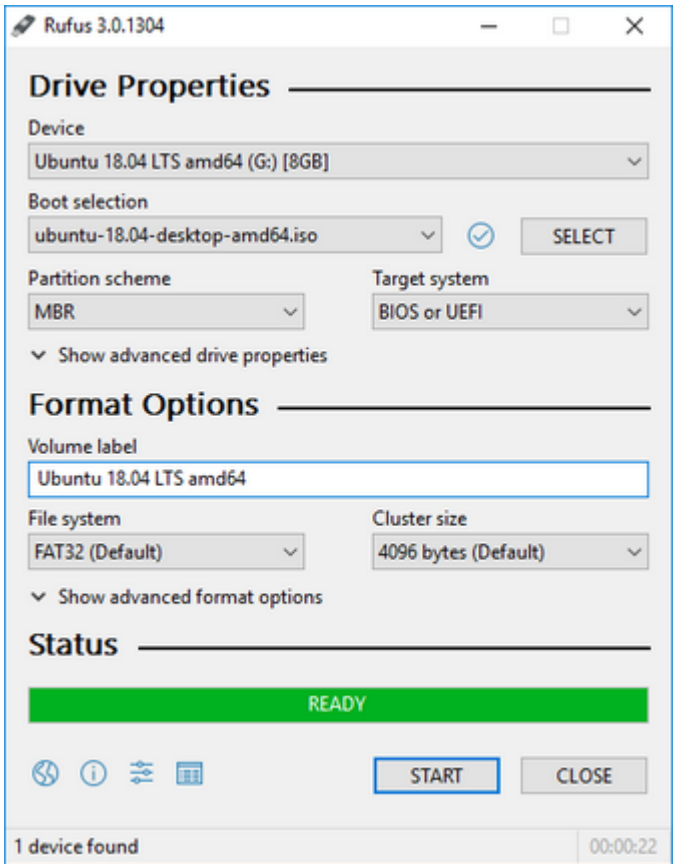

Figura 2.1 Interfaz de Rufus.

2.3 Git

Git es un software libre de control de versiones diseñado por Linus Torvalds. El control de versiones es la gestión de los diversos cambios que se realizan en los ficheros de nuestro proyecto. Git es, por tanto, una herramienta que permite controlar todas esas modificaciones de forma mucho más sencilla y hacer también más fácil la administración de las distintas versiones de cada producto desarrollado.

Figura 2.2 Logotipo de Git.

Las características más importantes de Git son :

- Software libre.
- Gestión distribuida, permitiendo tener el repositorio en local o en un servidor externo y que todos los miembros del equipo puedan acceder a él.
- Mantenimiento del historial completo de versiones.
- Gestión eficiente de proyectos grandes.
- Rapidez en la gestión de ramas.
- Realmacenamiento periódico de paquetes.

Para poder utilizar Git en Debian, se instala mediante el comando `sudo apt-get install git`. A partir de ese momento se podrá comenzar a usar comandos de Git, de los que los mas comunes son `git add`, `git commit`, `git diff`, `git stash` o `git clone`.

2.4 GitLab

GitLab es una herramienta web de control de versiones, un gestor de repositorios en código libre basada en Git. Ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado en código abierto.

Desarrollado por los programadores ucranianos Dmitriy Zaporozhets y Valery Sizov en el lenguaje de programación Ruby con algunas partes reescritas posteriormente en Go. Inicialmente era una solución de gestión de código fuente para colaborar con su equipo en el desarrollo de software. Luego evolucionó a una solución integrada que cubre el ciclo de vida del desarrollo de software y luego a todo el ciclo de vida de DevOps. La arquitectura tecnológica actual incluye Go, Ruby on Rails y VUe.js.

GitLab Inc. cuenta con un equipo de 1308 miembros y organizaciones como la NASA, el CERN, IBM o Sony lo utilizan.

En el caso de este proyecto, todo el desarrollo se hará en los servidores oficiales de Gitlab, aunque tambien permite la instalación en servidores privados. Para poder acceder entramos en *http://gitlab.com*, y una vez se tenga creado el usuario se podrá comenzar a trabajar tanto con los repositorios como con sus 'extras': Wiki, Issue Tracker, CI/CD, ...

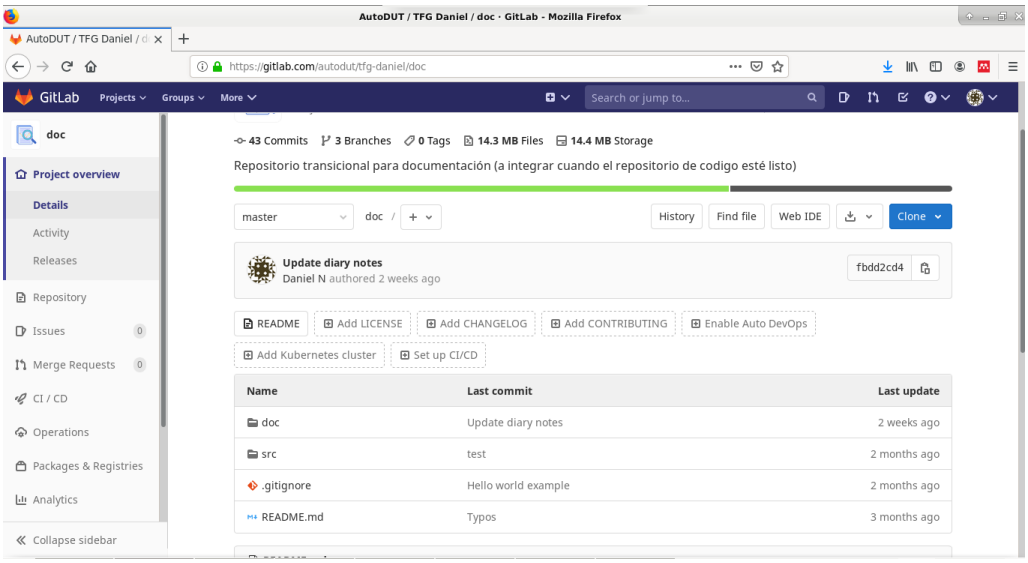

Figura 2.3 Servidor web de GitLab.

2.5 GitKraken

GitKraken es una herramienta gráfica de control de versiones desarrollada con Electron. Con ella podemos, de forma sencilla, clonar nuestros repositorios de GitLab, así como ver todos los commits, crear ramas, tags, ver todo el historial de nuestro trabajo, etc. Es de pago aunque posee una versión gratuita muy funcional.

Como principales características:

- Mejor optimizada que la competencia.
- Permite deshacer la última acción con un solo click.
- Solución sencilla a los merges, pudiendo seleccionar con qué líneas de código quedarte por medio de su propia herramienta.
- Ejecutar las acciones con sencillos drag-and-drop.
- Árbol con el histórico de commits altamente funcional, pudiendo administrar tus ramas y commits directamente sobre él para tener una forma visual de los cambios que estás realizando.
- Permite manejar varios perfiles a la vez.
- Interfaz simple, intuitiva y personalizable.

Empresas como Netflix, Tesla o Apple lo utilizan.

Para poder usar GitKraken se instala siguiendo los pasos de su web *https://www.gitkraken.com/download*. Luego, hay que configur las keys ssh para que Gitlab y GitKraken reconozcan al mismo usuario. La clave pública está en `/home/login/id_rsa.pub`, se copia y se pega en Gitlab.

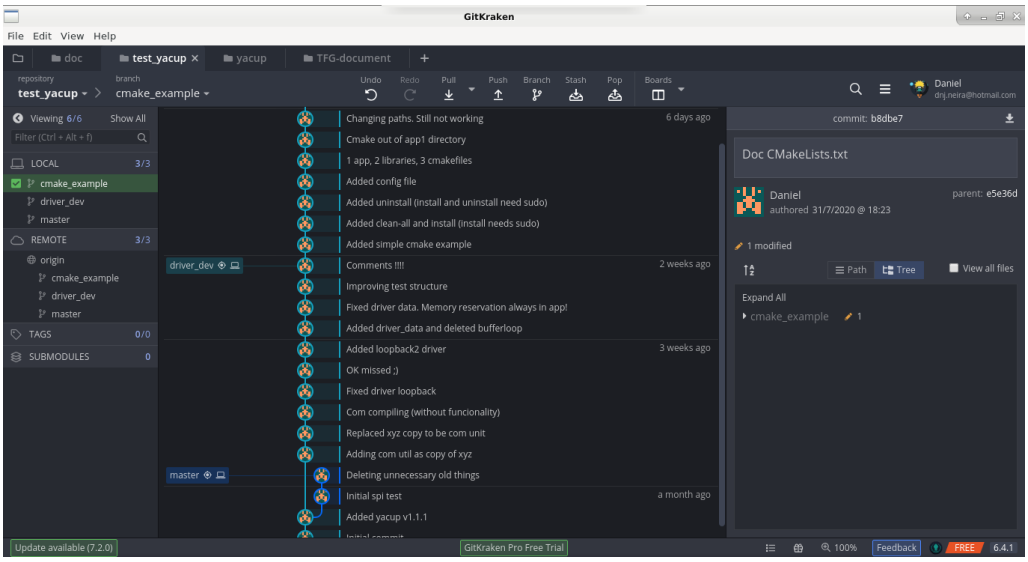

Figura 2.4 Interfaz de GitKraken.

2.6 Sublime Text

Sublime Text es un editor de texto y editor de código fuente multiplataforma. Aunque no es software libre, tiene una versión de evaluación gratuita y totalmente funcional.

Soporta muchos lenguajes de programación, entre ellos: ActionScript, AppleScript, ASP, Batch File, C, C#, C++, Clojure, CSS, D, Diff, Erlang, Expresión regular, Go, Graphviz (DOT), Groovy, Haskell, HTML, Java, JavaScript, LaTeX, Lua, Lisp, Makefile, Markdown, MATLAB, Objective-C, Ocaml, Perl, PHP, Python, R, Rails, ReStructuredText, Ruby, Scala, Shell Script (Bash), SQL, Tcl, Texto plano, Textile, XML, XSL, YAML.

Como características principales:

- "GOTO".
- Selecciones múltiples.
- Paleta de comandos.
- Poderosa API y sistema de paquetes.
- Muy personalizable.
- Editar edición.
- Cambio rápido entre proyectos.
- Cross Platform.

Para poder usa Sublime Text, se instala siguiendo los pasos de su web *https://www.sublimetext.com/3*, así como su editor de paquetes y demás herramientas necesarias.

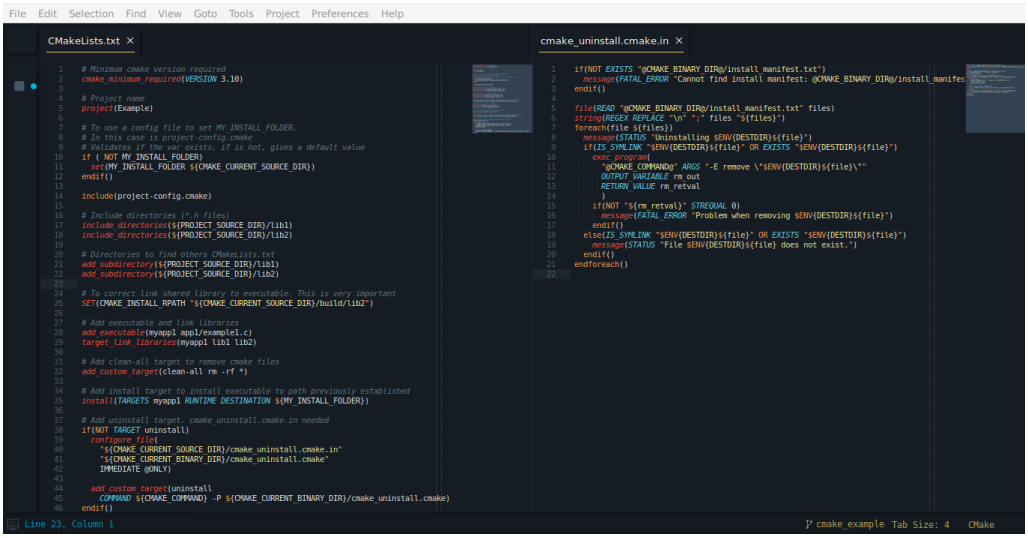

Figura 2.5 Interfaz de Sublime Text.

2.7 Kinetis Design Studio 3

La herramienta de desarrollo de software Kinetis Design Studio es un entorno de desarrollo basado en GNU/Eclipse para dispositivos Freescale Kinetis. Admite dispositivos Kinetis basados en Cortex-M y se integra con Processor Expert y Kinetis Software DevelopmentKit. KDS es compatible con SEGGER J-Link / J-Trace, P&E USB Multilink Universal / USBMultilink Universal FX y adaptadores de depuración CMSIS-DAP y utiliza la biblioteca newlib-nano Cruntime. Esta biblioteca de tiempo de ejecución ayuda a reducir la huella de memoria de una aplicación incorporada.

Kinetis Design Studio (KDS) es un entorno gratuito para Kinetis MCU que permite una edición, compilación y depuración sólidas de sus diseños. Basado en un software gratuito de código abierto que incluye Eclipse, GNU Compiler Collection (GCC), GNU Debugger (GDB) y otros, Kinetis Design Studio IDE es una herramienta de desarrollo simple sin limitaciones de tamaño de código.

Como principales características destacamos :

- Eclipse Luna 4.4.
- Sistemas operativos host:
  - Windows 7/8/10 (32 y 64 bits).
  - Linux de 64 bits (Ubuntu 10.04 (deb), RedHat / Centos 7 (rpm)).
  - Mac OS X (10.10 y 10.11) con soporte SEGGER J-Link.
- Depurador GDB con soporte para las siguientes de interfaces de depuración:
  - SEGGER J-Link (con servidor SEGGER GDB, Windows, Linux, Mac).
  - P&E Multilink (w / P & E GDB Server, Windows, Linux).
  - CMSIS-DAP (con circuito integrado OpenOCD GDB y OpenSDA, Windows, Linux).
  - Depuración en línea de comandos (CL) con GDB y OpenOCD TCL.
- Compatibilidad con complementos Eclipse descargables adicionales que incluyen reconocimiento RTOS (incluidos MQX ™ y FreeRTOS).
- Asistente de proyectos para crear bare metal, Kinetis SDK v1.xy v2 y proyectos de software Processor Expert.
- Software Processor Expert con soporte para Kinetis SDK v1.x.
- Lenguajes compatibles: ensamblador, C y C ++ (todos sin restricciones de tamaño de código).
- Bibliotecas incluidas: launchpad newlib y newlib-nano.
- Controladores periféricos Kinetis SDK y código de inicio compatible con CMSIS.
- Complementos Eclipse de GNU Arm para proyectos de marca administrada.
- Hacer proyectos de archivo.
- Herramientas GNU para la cadena de herramientas y bibliotecas de procesadores integrados Armpad (launchpad).
- Registro Eclipse CMSIS-SVD integrado y visor periférico.
- Soporte para la creación de proyectos MQX y el conocimiento del núcleo MQX (disponible como descarga MQX).
- Abierto para cualquier complemento de Eclipse, ya sea del ecosistema de Eclipse o de los socios de NXP Community Support.
- Se agrega soporte adicional para dispositivos a través de las actualizaciones de Eclipse entregadas con Kinetis SDK.

Para poder usar Kinetis Design Studio 3, se instala desde su web, *https://www.nxp.com/design/designs/design-studio-integrated-development-environment-ide:KDS_IDE*. KDS incluye el GCC ARM Embedded toolchain preparado para sistemas de 32 bits, por lo que si el ordenador es de 64 bits, al compilar el primer programa de ejemplo se obtiene un error de `cc1` (no encuentra este archivo).

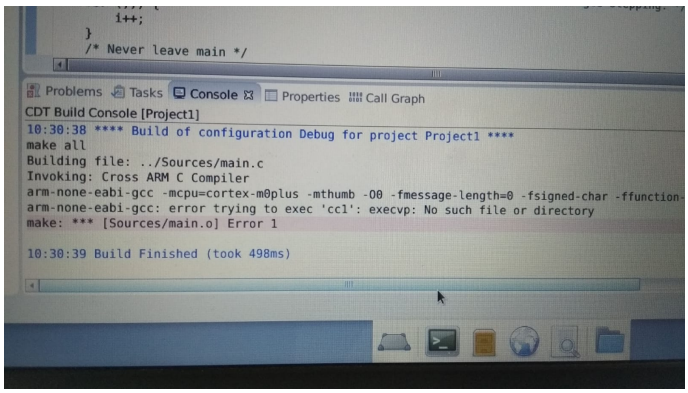

Figura 2.6 Error de compilación.

En el manual de instalación está la solución, para sistemas de 64 bits es necesario la instalación de algunos paquetes extra, que son: `libc6:i386`, `libncurses5:i386` & `libstdc++6:i386`.

NOTE

KDS includes the GCC ARM Embedded toolchain, which is built for 32 bit hosts. If you are using a 64 bit system, be sure you have the appropriate 32 bit packages installed.

- For **Ubuntu** 1404 these packages are required to be installed: libc6:i386, libncurses5:i386, & libstdc++6:i386.
- For **RPM based packages** these packages are required to be installed: glibc.i686 and libncurses.so.5.

Figura 2.7 Paquetes que faltan por instalar.

Se descargan de la web de Debian y se instalan mediante:

```
sudo dpkg -i <paquete>.deb
```

Entonces se pasa a resolver dependencias mediante:

```
sudo apt-get -f install
```

A partir de aqui, el programa debería compilar correctamente.

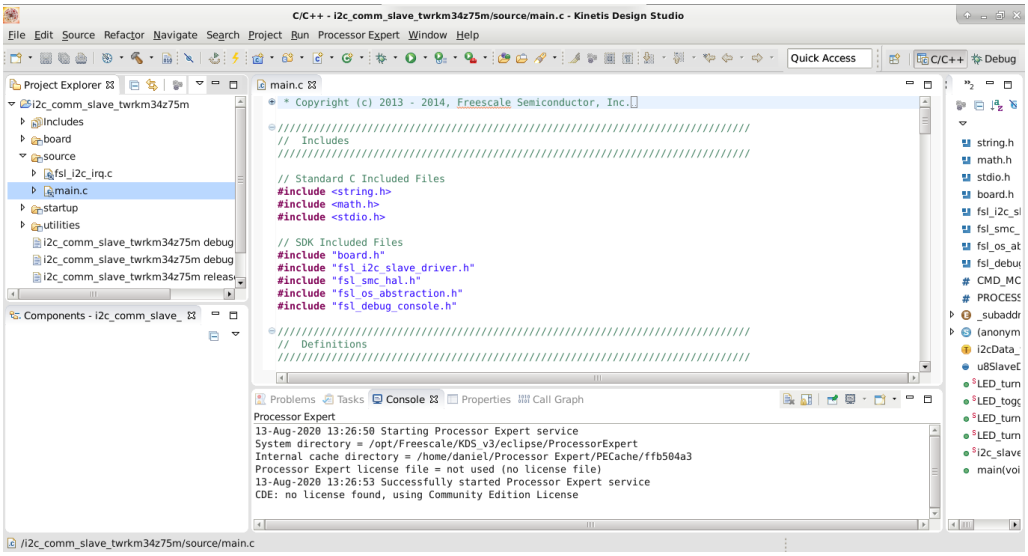

Figura 2.8 Interfaz de Kinetis Design Studio.

2.8 PlayOnLinux

PlayOnLinux es un software que permite instalar y utilizar fácilmente numerosas aplicaciones diseñados para ejecutarse con Microsoft Windows.

Características principales:

- No es necesario tener una licencia de Windows para usar PlayOnLinux.
- Está basado en Wine, por lo que se beneficia de todas sus funciones, pero evita que el usuario tenga que lidiar con su complejidad.
- Es un software gratuito.
- Usa Bash y Python.

Sin embargo,también tiene algunos inconvenientes, como todo software:

- Disminución ocasional del rendimiento (la imagen puede ser menos fluida y los gráficos menos detallados).

- No todas las aplicaciones son compatibles. No obstante, se puede utilizar el módulo de instalación manual.

Para poder usar PlayOnLinux, se instala siguiendo el manual de su web *https://www.playonlinux.com/en/*

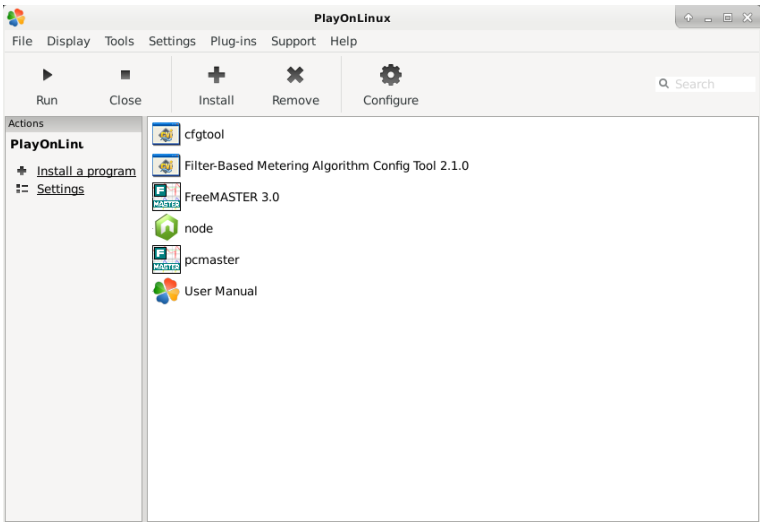

Figura 2.9 Interfaz de PlayOnLinux.

2.9 FreeMASTER

FreeMASTER es un software desarrollado por NXP para controlar una aplicación integrada mediante un entorno gráfico que se ejecuta en un ordenador. La aplicación se creó inicialmente para desarrolladores de aplicaciones de control de motores en tiempo real, pero muchos usuarios la encontraron muy útil para su desarrollo específico.

FreeMASTER se puede instalar en cualquier sistema basado en el sistema operativo Windows. El paquete FreeMASTER 3.0 también contiene el servicio FreeMASTER "Lite" que puede ejecutarse en otros sistemas operativos (como Linux) y puede actuar como una interfaz de comunicación entre las páginas de control y los microcontroladores de destino.

2.9.1 Características

Características principales de FreeMASTER son:

- Entorno gráfico y navegación sencillos.
- Conexión nativa en serie simple y posibilidad de otras opciones en plataformas seleccionadas (BDM, JTAG, CAN y otras).
- Acceso en tiempo real a variables C en una aplicación de microcontrolador de destino en ejecución.
- Visualización de datos de tiempo en la ventana del osciloscopio.
- Adquisición de cambios rápidos de datos usando Recorder.
- Soporte integrado para tipos de variables estándar (entero, punto flotante, campos de bits) y tipos extendidos (fraccional).
- Interpretación de valores usando mensajes de texto personalizado.
- Varias transformaciones integradas para variables de tipo real.
- Extracción automática de direcciones y tamaños de variables de archivos de salida del compilador.
- Soporte para Target-Side Addressing (TSA) sobre objetos y tipos variables que se recuperarán de la aplicación de destino.
- Modo demo con contraseña de protección.
- Control basado en HTML o páginas de descripción representadas por los engines de Internet Explorer o Chromium.
- Interfaz ActiveX para habilitar VBScript o JavaScript sobre aplicaciones integradas desde páginas de Internet Explorer o cualquier aplicación de terceros que admita la tecnología ActiveX.

- Interfaz JSON-RPC para habilitar el control desde una página Chromium, un navegador Chrome independiente, scripts node.js, Pythonscripts o cualquier aplicación de terceros que admita el protocolo JSON-RPC.

2.9.2 Usos

Como hemos apuntado antes el objetivo principal del desarrollo de FreeMASTER era ofrecer una herramienta para depurar y demostrar algoritmos y aplicaciones de control de motores. El resultado es una herramienta versátil que se puede utilizar para algoritmos y aplicaciones multipropósito. Algunos de los usos son:

- **Depuración en tiempo real**: FreeMASTER permite a los usuarios depurar aplicaciones en tiempo real a través de su capacidad para detectar variables. Además, permite la depuración a nivel de algoritmo, lo que ayuda a acortar la fase de desarrollo.
- **Herramienta de diagnóstico**: la capacidad de control remoto de FreeMASTER permite que se utilice como una herramienta de diagnóstico para depurar aplicaciones de clientes de forma remota a través de una red.
- **Demostraciones**: FreeMASTER es un excelente herramienta para demostrar el algoritmo o la ejecución de la aplicación y las salidas variables.
- **Educación**: FreeMASTER puede utilizarse con fines educativos. Sus funciones de control de aplicaciones permiten a los estudiantes jugar con la aplicación en el modo de demostración, aprendiendo a controlar la ejecución del programa.

2.9.3 Funciones de visualización

FreeMASTER se comunica con la aplicación del sistema de destino a través de la comunicación en serie para leer y escribir las variables internas de la aplicación. Proporciona las siguientes funciones de visualización para mostrar información variable en un formato fácil de usar:

- **Variable Watch**: los valores de las variables seleccionadas se presentan en una cuadrícula en un formato definido basado en texto. Los valores se pueden modificar directamente en la cuadrícula.
- **Osciloscopio**: proporciona monitoreo / visualización de las variables de la aplicación de la misma manera que un osciloscopio estándar con un CRT. En este caso, las tasas de monitorización están limitadas por la velocidad de comunicación en serie.
- **Recorder**: proporciona la monitorización / visualización de las variables de la aplicación que cambian a una velocidad más rápida que la tasa de muestreo del osciloscopio. Mientras que el osciloscopio lee periódicamente los valores de las variables FreeMASTER y los traza en tiempo real, Rcorder se ejecuta en la placa de destino. Los valores variables se muestrean en un búfer de memoria en el tablero y los datos muestreados se descargan del tablero al FreeMASTER. Este mecanismo permite un período de muestreo mucho más corto y permite el muestreo y el trazado de acciones muy rápidas.

2.9.4 Conexión

FreeMASTER requiere un puerto de comunicación en serie en el hardware de desarrollo de destino. La conexión se realiza mediante un cable serie estándar RS-232. Por un lado, el cable se conecta al puerto serie del ordenador (COM1, COM2 u otro) y, en el lado opuesto, al conector serie de la placa de desarrollo de destino. Además del enlace RS232, FreeMASTERlos puede escribir y utilizar los módulos enchufables de comunicación personalizados. Hay complementos de comunicación disponibles para el protocolo de calibración CAN, el puerto de intercambio de datos en tiempo real JTAG en 56F800E, la interfaz BDM en los dispositivos HCS08 / 12, etc.

Como estamos usando Linux para todo el proyecto y FreeMASTER sólo se puede usar en Windows vamos a hacer uso de un software que permite instalar programas de Windows en Linux como hemos apuntado anteriormente, PlayOnLinux.

Para instalar FreeMASTER, se sigue tanto el manual de PlayOnLinux *https://www.playonlinux.com/en/* como el propio de FreeMASTER *https://www.nxp.com/docs/en/user-guide/FMSTERUG.pdf*

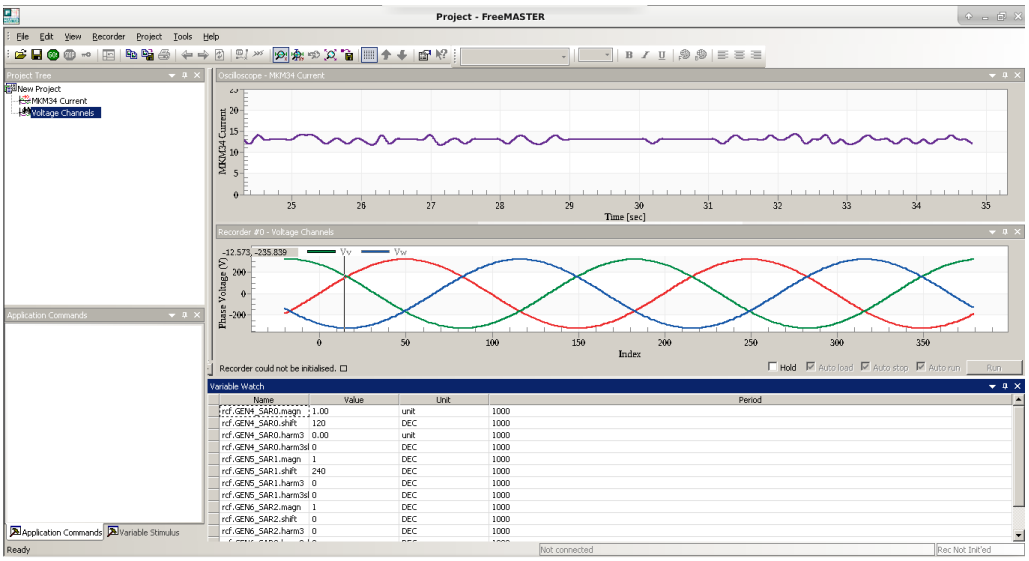

Figura 2.10 Interfaz de FreeMASTER.

2.10 Configuration tool

La configuración general del algoritmo se puede realizar fácilmente mediante Cfg_tool. Esta herramienta permite ajustar el algoritmo de medición de forma interactiva con respecto al hardware del medidor de potencia y las capacidades del firmware. La sesión de configuración siempre debe terminar generando un archivo de encabezado C que contenga todas las configuraciones.

El software de configuración Cfg_tool tiene como objetivo principal ajustar los filtros para que coincidan con el rendimiento requerido y generar el archivo de encabezado C que contiene el código fuente con los parámetros que describen el comportamiento del algoritmo de medición basado en filtros. Automatiza el procedimiento de configuración y optimización del algoritmo, al tiempo que proporciona una estimación aproximada de la carga computacional requerida. La herramienta genera el código C para la configuración de algoritmos de medición de 32 bits y admite topologías de medidores de potencia monofásicos, bifásicos (Form-12S) y trifásicos.

Para poder usar Configuration tool, se instala siguiendo el manual *https://www.nxp.com/docs/en/application-note/AN4265.pdf* .

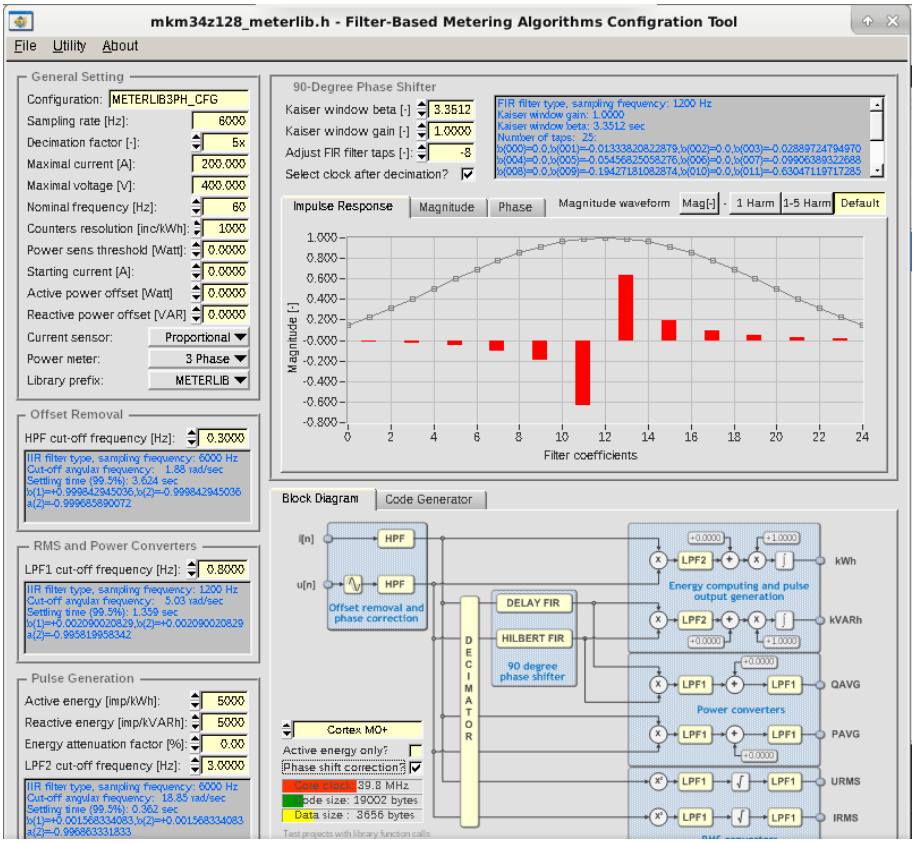

Figura 2.11 interfaz de Configuration tool.

2.11 Terminales de puerto serie

Durante el proyecto hemos usado varios terminales de puerto serie :

- **Putty**: es un cliente SSH, Telnet, rlogin, y TCP raw con licencia libre. Su característica principal es su interfaz gráfica y la posibilidad de guardar sesiones con la configuración para después poder volverlas a usar.
- **Screen**: Terminal serie en linea de comandos.
- **Minicom**: Programa de comunicaciones gratuito. Funciona en la mayoría de entornos. También es posible guardar la sesión.

Se instalan los terminales de la siguiente manera:

- **Putty**: Se descargan de su web y se siguen los pasos allí indicados, *https://www.putty.org/*.
- **Screen**: Se instala con el comando `sudo apt install screen`.
- **Minicom**: Se instala con el comando `sudo apt-get install minicom`.

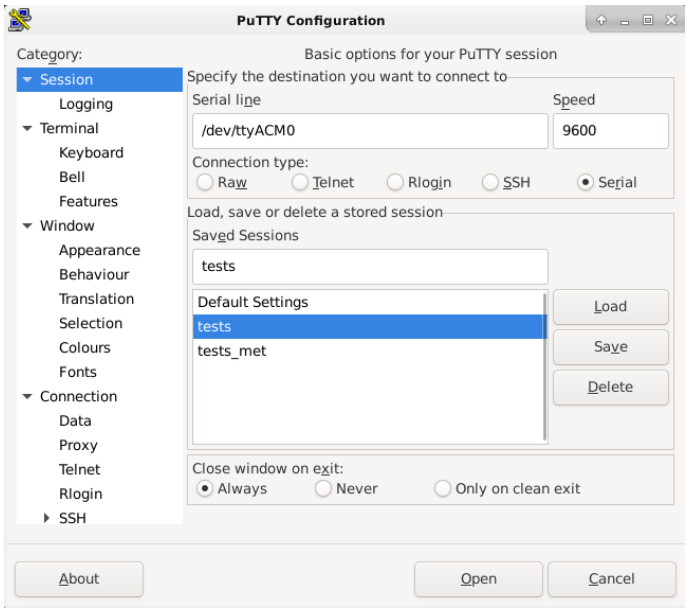

Figura 2.12 Interfaz de Putty.

3 Descripción del hardware que se va a usar como plataforma de pruebas

En este proyecto vamos a usar como hardware la placa de desarrollo TWR-KM34Z50M, de NXP.

3.1 Características

Las características principales del TWR-KM34Z50M son las siguientes:

- Módulo de microcontrolador compatible con torre.
- MKM34Z128CLL5 (para TWR-KM34Z50M) MCU (50 MHz, 128 KB Flash, 16 KB RAM, low power, LQFP100 package).
- Interfaz USB con conector USB Mini-AB.
- Liquid Crystal Display (LCD) con cristal de 160 segmentos.
- Circuito de depuración *Onboard*: open source JTAG/SWD (OpenSDA) con puerto serie virtual.
- Acelerómetro de tres ejes y sensor de inclinación antitampering (MMA8491Q).
- Cuatro LEDs controlables por el usuario.
- Dos botones de usuario para interrupciones de General-Purpose Input/Output (GPIO).
- Un botón de usuario para detección de manipulación.
- Un botón de usuario para restablecer MCU.
- Potenciómetro.
- Headers para acceso directo GPIO y ADC.
- Pasadores de sabotaje externos.
- Fuente de alimentación independiente que funciona con batería para el Real-Time Clock (RTC) y módulos de detección de manipulación.
- Generador de señal sinusoidal de 7 canales a bordo con interfaz USB para emular las formas de red de Alternating Current (AC).
- Soporte Infrared Data Association (IrDA).
- Sensor de temperatura Negative Temperature Coefficient (NTC).
- Socket de propósito general Tower Plug-In (TWRPI).

3.2 Descripción hardware

El TWR-KM34Z50M es una placa de desarrollo para MKM34Z128CLL5, un microcontrolador de la familia de Kinetis-M en un paquete LQFP-100 package. Los otros componentes hardware son :

- ADC Sigma-delta.
- Módulo de detección de tampering.
- Reloj seguro en tiempo real.
- Fuente de alimentación independiente que funciona con baterías.
- Alimentación, circuito de depuración OpenSDA mediante SWD y puerto serie virtual a traves de un único Mini-USB controlado por un micro auxiliar K20.
- Generador de señales que permite la emulación de señales de redes eléctricas, tanto de tensión como de intensidad de forma simultánea, controlado por un segundo micro auxiliar K20.

El TWR-KM34Z50M está diseñado para su uso en el sistema de torre Freescale, pero también puede funcionar en modo independiente.

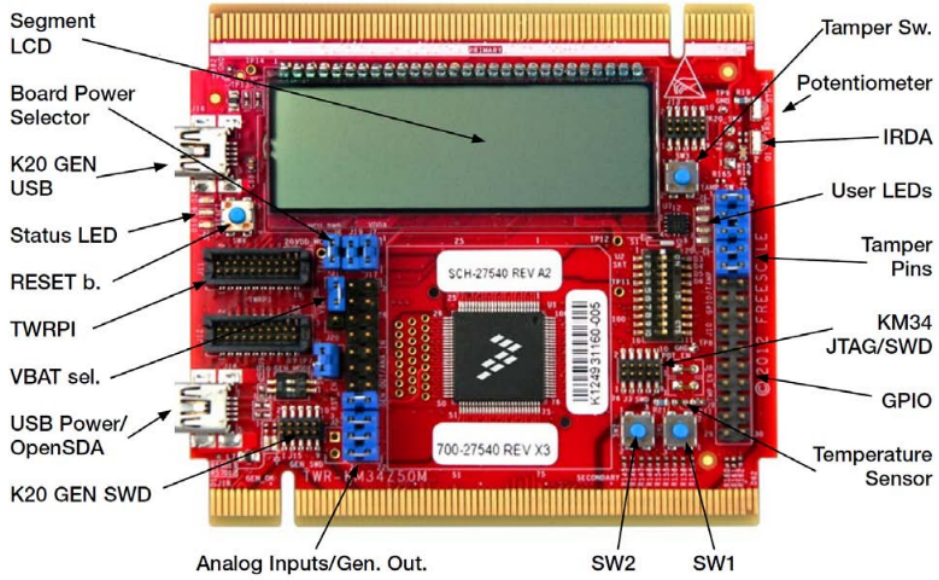

Figura 3.1 Frontal del módulo TWR-KM34Z50M (Dispositivos TWRPI no mostrados).

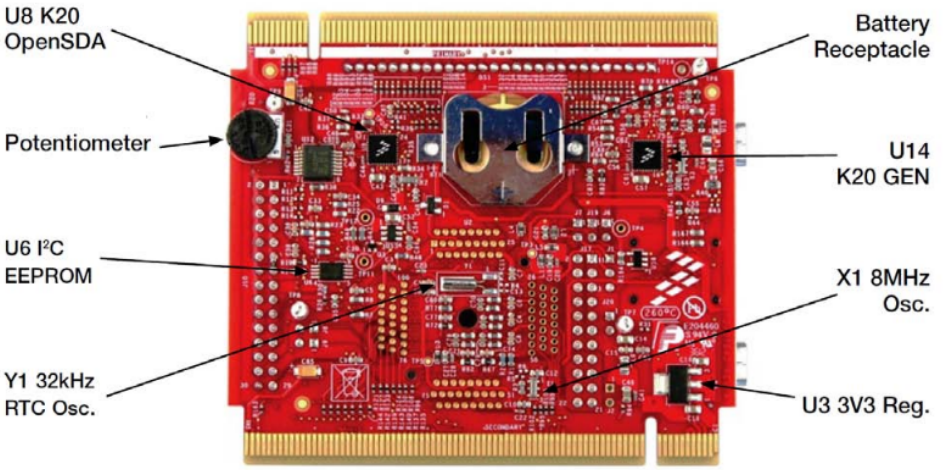

Figura 3.2 Trasera del módulo TWR-KM34Z50M.

3.2.1 Diagrama de bloques de la placa de desarrollo

La siguiente figura representa el diagrama de bloques del TWR-KM34Z50M.

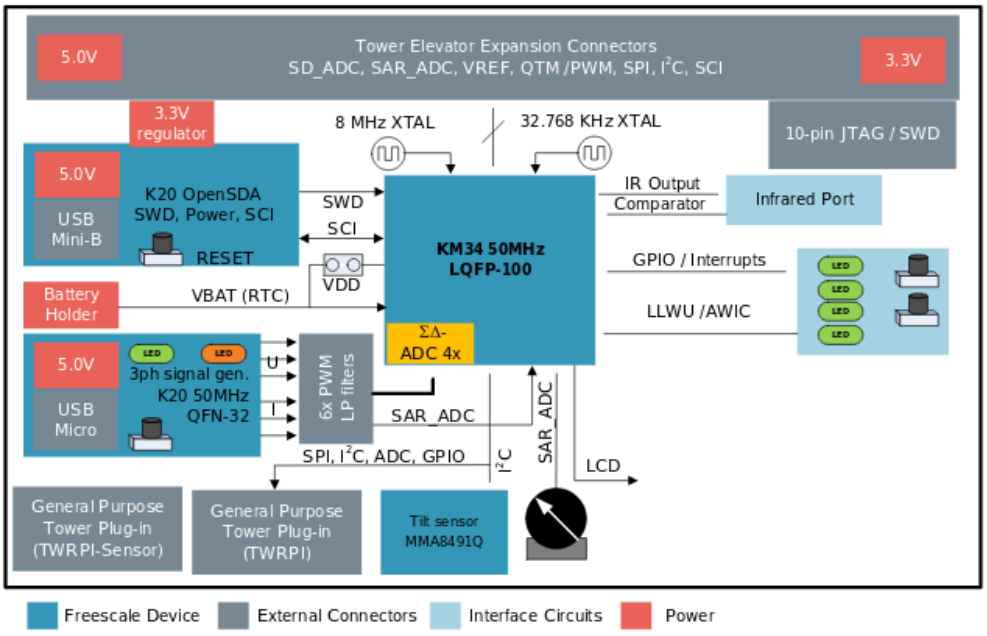

Figura 3.3 Diagrama de bloques de la tower board TWR-KM34Z50M.

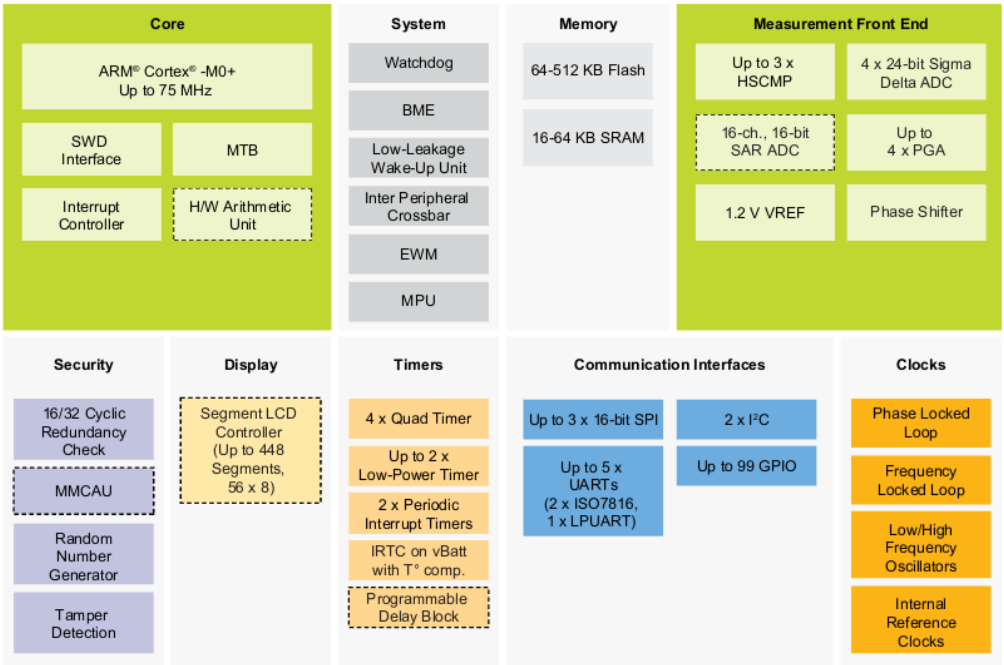

Figura 3.4 Diagrama de bloques de la serie KM MCU.

3.2.2 Microcontroladores de la familia Kinetis M

Los dispositivos de la familia Kinetis M (KM) son MCU de 32 bits con tecnología flash integrada en 90 nm Thin Film Storage (TFS). Estos dispositivos están enfocados principalmente para servir a los mercados de medición de medidores de energía monofásicos inteligentes en India, China y los países de la Unión Europea y medidores de dos fases en los Estados Unidos y Japón.

Los dispositivos KM se basan en el núcleo ARM Cortex-M0 + de 32 bits con AFE. Las velocidades de reloj de la CPU en estos dispositivos pueden alcanzar hasta 50 MHz. La familia de dispositivos KM incluye ADC sigma-delta (SD) de alta precisión, amplificador de ganancia programable (PGA), referencia de voltaje interno de alta precisión, memoria flash, RAM, bloque lógico de compensación de fase y otros periféricos. La familia KM proporciona detección de manipulaciones y un reloj preciso en tiempo real en todos los dispositivos. Las características importantes de la MCU MKM34Z128CLL5 se enumeran en la siguiente tabla:

Tabla 3.1 Características del MKM34Z128CLL5.

Característica	Descripción
Características operativas	• Rango de voltaje 2.7-3.6 V (con AFE) • Rango de voltaje 1.71-3.6 V (sin AFE) • Voltaje de programación flash de 1.71-3.6 V • Rango de suministro de batería iRTC 1.71-3.6 V • Rango de temperatura (TA) -40 °C a 85 °C • Modos de operación flexible
Core	• ARM Cortex-M0 + Core de alto rendimiento • Hasta 50 MHz de frecuencia de reloj central • Admite la arquitectura de conjunto de instrucciones v6-M (ISA) que incluye todas las v7-Mins de 16 bits más una serie de Thumb-2 de 32 bits • 100 % compatible con Cortex-M0 • Rendimiento de 0.95 DMIPS por MHz cuando se ejecuta desde RAM interna • Controlador de interrupción vectorial anidado • 32 interrupciones vectoriales • Cuatro niveles de prioridad programables

Característica	Descripción
Reloj	• Oscilador MHz • Rango medio 1-8 MHz • Rango alto: 8-32 MHz • Oscilador de cristal de 32.768 kHz en el dominio de • Dos referencias de reloj ajustables internas: 32 kHz y 4 MHz • Oscilador interno de baja potencia de 1 kHz • Phase-Locked Loop (PLL) para generar relojes para AFE • Rango de entrada: 31.25-39.0625 kHz • Rango de salida: 11.72-14.65 MHz • Frequency-Locked Loop (FLL) para generar núcleos, sistemas y relojes flash • Rango de entrada: 31.25-39.0625 kHz • Rango de salida: 20-50 MHz • Ratio de reloj : – Core: Bus: Flash = 2: 1: 1 para reloj central > 25 MHz – Core: Bus: Flash = 1: 1: 1 para reloj central <= 25 MHz
Analógico	• ADC Sigma-Delta de 24 bits • PGA con ganancias de 1 a 32 • VREF de 1.2 V • ADC SAR de 16 bits • CMP con Digital to Analog Converter (DAC) de 6 bits
Características de control de energía, protección y sistema	• Varios modos de Parar, Esperar y Ejecutar para proporcionar una menor potencia en función de las necesidades de la aplicación • AWIC para despertar desde los modos Parar y VLPS • El registro de habilitación de reloj periférico puede desactivar relojes a módulos no utilizados, reduciendo así las corrientes • Advertencia de bajo voltaje y detección con puntos de disparo seleccionables • Código de operación ilegal y detección de dirección ilegal con reinicio • Módulo CRC programable por hardware para admitir comprobaciones de redundancia cíclica rápida (CRC) • Generador de números aleatorios (RNGA) • Identificador de chip único de 128 bits • Watchdog de software y hardware con pin de monitor externo • Cuatro entradas de sabotaje para Detección de Sabotaje (Parte de iRTC) • Seguridad de flash y protección de bloque • Barra transversal periférica para permitir la reasignación de señal interna para flexibilidad

Característica	Descripción
Depuración	• 2-pin Serial Wire Debug (SWD): Interfaz de depuración ARM estándar
Temporizadores	• Reloj de tiempo real independiente (iRTC) alimentado independientemente por batería y compensación de deriva de reloj de cristal en chip • Temporizador cuádruple (*Quad Timer*) (cuatro canales) • Temporizador de interrupción programable (PIT) • Temporizador de baja potencia (LPTMR)
Comunicaciones	• Receptor / transmisor asíncrono universal (Universal Asynchronous Receiver/Transmitter (UART)) (todo el control de flujo de hardware compatible con módulos UART) • ISO 7816 (en 2 UART) • Capacidad IrDA en todos los UART • Direct Memory Access (DMA) (todos los UART) • Un módulo UART con soporte de 5 V AMR (UART1) • Serial Peripheral Interface (SPI) con FIFO (SPI1) y sin FIFO (SPI0) • Un módulo SPI con soporte de 5 V AMR (SPI1) • Inter-Integrated Circuit (I2C) (x2) con soporte de protocolo SMBUS
Entrada/Salida	• Hasta nueve puertos eGPIO y un puerto con filtros de falla digital • Capacidad de solicitud de interrupción de pines / DMA • Pines eGPIO conectados al bus local de 32 bits del procesador • Plataforma de bus RGPIO • Pines eGPIO también accesibles a través de accesos de bus IPS (protegidos a través de AIPS) • Módulo eGPIO para tener protección de acceso incorporada
Memoria	• Array simple de FLASH de 128 KB / 64 KB • Memoria flash leída y escrita a 1.71 V • No FlexMemory • Núcleo: La frecuencia de flash es 2: 1 • 1: 1 para baja frecuencia de núcleo (<= 25 MHz) • 16 KB de RAM de acceso simple
Display	• LCD de 4x40 segmentos • LCD de 8x36 segmentos • Cristal LCD de 3 V compatible con detección de segment fault

Característica	Descripción
Consumo de energía	• Modo RUN (todos los módulos apagados): 6.0 mA Ecuación IDD: 1.14 mA + 97 uA / MHz

3.2.3 Reloj

El Kinetis-M arranca desde un reloj de referencia interno de 2 MHz de reloj de core y 1 MHz de reloj de bus. Cuando la MCU arranca en modo LPBOOT (arranque de baja potencia), los relojes del núcleo y del sistema se dividen por 8.

El software de la CPU puede habilitar el oscilador RTC conectado a EXTAL0 / XTAL0 o se puede usar el segundo oscilador de alta frecuencia conectado a EXTAL1 / XTAL1.

El rango del oscilador de cristal RTC es de 31.25 kHz a 39.0625 kHz (típicamente 32.768 kHz)

El rango de oscilador de cristal de alta frecuencia es de 1 kHz a 32 MHz.

Opcionalmente, se pueden usar dos referencias de reloj interno (IRC) (rápido = 4 MHz y lento = 32.768 kHz).

La frecuencia de la CPU se puede aumentar usando las funciones de reloj FLL (DCO) o PLL. En la mayoría de las aplicaciones, se considera el uso del cristal externo único de 32 kHz. PLL con un multiplicador fijo (x375) se puede usar para registrar el AFE que funciona a ~ 12.2 MHz y el FLL se puede usar para marcar el núcleo de la CPU (hasta 50 MHz) y el resto de los módulos MCU.

Para obtener los mejores resultados de AFE, se recomienda el uso del reloj externo preciso.

3.2.4 Sistema de alimentación

Cuando se instala en un sistema de torre, el TWR-KM34Z50M puede alimentarse desde una fuente a bordo o desde otra fuente en el sistema de torre ensamblado.

Cuando opera independientemente, la fuente de alimentación principal (5.0 V) para el módulo TWR-KM34Z50M se deriva del conector OpenSDA USB mini-B (J14). Un regulador de baja caída proporciona un voltaje de suministro de 3.3 V desde el voltaje de entrada de 5.0 V. Todas las opciones seleccionables por el usuario se pueden configurar con los jumpers: J1, J6, J7, J19 y S1.

3.2.5 Detección de manipulación iRTC y RTC VBAT

El módulo de detección de manipulación y el módulo de reloj de tiempo real (RTC) en el MKM34Z128CLL5 tienen dos modos de operación: encendido del sistema y apagado del sistema. Durante el apagado del sistema, el módulo de detección de sabotaje y el RTC se alimentan de la fuente de alimentación de respaldo (VBAT) y están aislados eléctricamente del resto de la MCU. El TWR-KM34Z50M proporciona un receptáculo de batería para una batería de celda de moneda que se puede utilizar como suministro VBAT. El receptáculo puede aceptar baterías de celda de moneda de litio de 3 V y 20 mm de diámetro.

3.2.6 Interfaz de depuración

Se proporcionan dos opciones de interfaz de depuración: el circuito OpenSDA integrado y un conector ARM JTAG / SWD externo. El conector ARM-JTAG / SWD (J3 SWD) es un conector estándar de 2x5 pines que proporciona un cable depurador externo con acceso a la interfaz JTAG del MKM34Z128CLL5. Alternativamente, la interfaz de depuración OSJTAG integrada se puede utilizar para acceder a la interfaz de depuración del MKM34Z128CLL5.

- El conector J13 (K20_SWD) se usa de fábrica para actualizar el firmware de OpenSDA al dispositivo K20.
- El conector J15 (GEN_SWD) se usa para el generador de a bordo auxiliar basado en K20.

3.2.7 OpenSDA

Un circuito OpenSDA basado en MK20DX128VMF5 integrado proporciona una interfaz de depuración SWD al MKM34Z128CLL5. Se puede utilizar un cable USB estándar A macho a mini-B macho (incluido) para la depuración a través del conector USB (J14). La interfaz OpenSDA también proporciona un puente USB a serie. Los controladores para la interfaz OpenSDA se proporcionan en el P&E Micro OSBDM / OSJTAG Tower Toolkit.

Conector de depuración Cortex

El conector de depuración Cortex es un conector de 10 clavijas (0.05 ") que proporciona acceso a las señales SWD disponibles en el dispositivo KM34. Las conexiones (J3) se muestran en la siguiente tabla.

Tabla 3.2 Conector de depuración Cortex.

Pin	Función	Conexión
1	V_BRD	Suministro de 3.3 V MCU (MCU_PWR)
2	WD_DIO_TGTMCU	PTE6/CMP0P2/PXBAR_IN5/SCI2_RXD/LLWU_P5/SWD_IO
3	GND	GND
4	SWD_CLK_TGTMCU	PTE7/AD6/PXBAR_OUT5/SCI2_TXD/SWD_CL
5	GND	GND
6	NC	-
7	NC	-
8	NC	-
9	NC	-
10	RST_TGTMCU_B	PTE1/RESET

3.2.8 Sensor de inclinación / acelerómetro

El acelerómetro digital MMA8491Q está conectado a la MCU MKM34Z128CLL5 a través de una interfaz I2C (I2C0) y de señales de sabotaje (TAMPER0, 1, 2).

3.2.9 Potenciómetro, sensor de temperatura, pulsadores, LED

El TWR-KM34Z50M también tiene las siguientes características:

- Un potenciómetro conectado a una señal de entrada ADC (PTG1 / AD10)
- Sensor de temperatura (NTC) conectado a una señal de entrada ADC (PTF0 / AD7)
- Medición del voltaje de la batería conectada a la entrada ADC (PTF2 / AD9)
- Dos interruptores de botón (SW1 y SW2 conectados a PTD0 y PTE4, respectivamente)
- Interfaz IrDA a través de PTC2, PTC3
- Cuatro LED controlables por el usuario conectados a señales GPIO (opcionalmente aislado con el interruptor S1):
  - LED verde (D3) a PTE5
  - LED Rojo (D4) a PTF1
  - LED naranja (D5) a PTD1 / GPIO3
  - LED amarillo (D6) a PTC1 / GPIO4 / CMP1P1

3.2.10 Interfaz USB-a-serie

El circuito K20 OpenSDA integrado admite la emulación de puerto USB a serie a través del dispositivo USB CDC Serial Class.

Esto significa que cada vez que la tarjeta Tower se conecta al puerto USB de la computadora, la placa se reconoce como un nuevo virtual Puerto COM (COM5 o COM6 en la mayoría de los casos). El usuario puede conectarse con la aplicación integrada que se ejecuta en el dispositivo KM34 y ver la salida en la PC que ejecuta la aplicación de terminal en serie.

3.3 Generador de señales K20 onboard

El generador de señal K20 es capaz de generar siete formas de onda sinusoidales para emular la red eléctrica real (corriente trifásica + voltaje trifásico) con frecuencia amplitud, desplazamiento de fase y distorsión armónica configurables. La amplitud máxima de las formas de onda generadas es 1.0 V (pk-to-pk), por lo que pueden ser medidas por ADC trabajando en modo Single-Ended. El generador se puede conectar a través del conector USB dedicado al ordenador que ejecuta la aplicación Graphic User Interface (GUI) FreeMASTER, donde los parámetros netos se pueden sintonizar fácilmente. Las señales analógicas externas se pueden conectar directamente a las entradas ADC mediante el encabezado J17.

3.3.1 Medición de la corriente de entrada de la CPU KM34 (Idd)

El K20 puede medir la corriente Idd del KM34 y enviar el valor a través de la conexión USB al PC que ejecuta la aplicación FreeMASTER.

Uno de los canales K20 ADC está dedicado a la medición del consumo de corriente Idd del KM34. La corriente de entrada se escanea como la caída de voltaje en la resistencia shunt de 10 Ohmios conectada en el fuente Vdd del MCU. Esta caída de voltaje se amplifica y convierte mediante un amplificador diferencial U/I (U18) y luego se convierte al rango de voltaje apropiado para la entrada ADC de K20. Con estos valores de componentes específicos, se pueden medir corrientes Idd en el rango de 0.5-20 mA, con una precisión razonable. Con valores de resistencia shunt de detección de corriente más grandes, se puede obtener una mayor precisión para las corrientes más bajas.

3.3.2 Características del generador K20

Las principales características del generador de señal sinusoidal K20 son:

- Basado en la subfamilia K20 (QFN32) de MCU Kinetis
- Siete señales PWM, capaces de generar formas de onda sinusoidales (3xI + 3xU, es decir, corriente trifásica + voltaje trifásico)
- Amplitud, frecuencia y fase configurables
- Jumpers en las entradas del ADC
- Puerto serie virtual (USB-CDC) para el PC que ejecuta la aplicación FreeMASTER
- Medición de corriente Idd a través del canal del ADC del K20

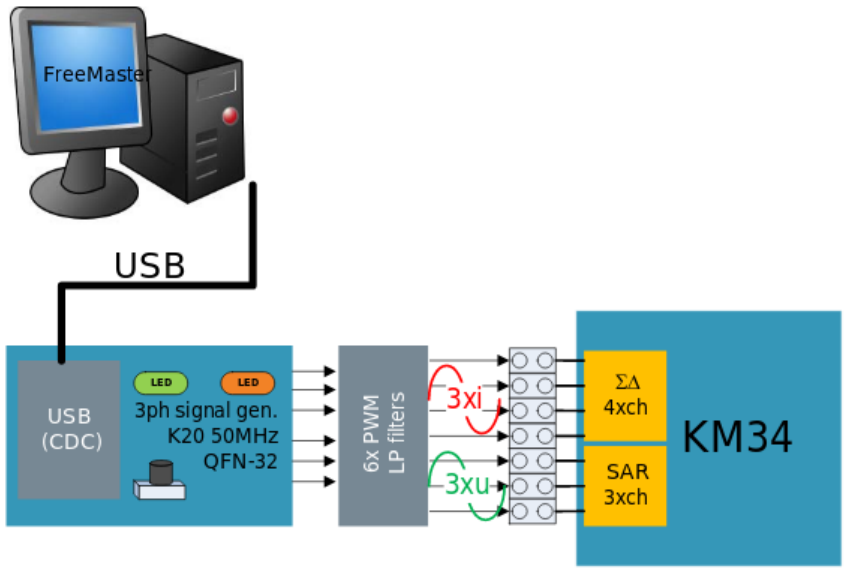

Figura 3.5 Concepto de generador de señales K20.

3.3.3 Generador K20, USB-CDC y FreeMASTER GUI

Para usar el generador K20 integrado, las salidas K20 USB y del generador deben estar habilitadas por el interruptor DIP, SW5.

La configuración predeterminada de SW5 se lee por programa después del *power-on-reset*, por lo que se requiere el ciclo de alimentación de la placa para aplicar los cambios.

SW5 puede desactivar el USB y el generador para reducir el ruido potencial causado por los circuitos K20.

Esto es importante en los casos donde la función del generador no es necesaria y el ruido puede afectar el rendimiento del ADC. Si la comunicación USB K20 está desactivada, la MCU pasa al modo de parada de muy baja potencia (VLPS) inmediatamente después del reinicio de encendido de la placa. Cuando solo se necesita la funcionalidad de detección de corriente Idd MK34 MCU, la salida del generador se puede desactivar para reducir el ruido proveniente de PWM, mientras que la medición de corriente Idd a través de USB todavía está activa. Alternativamente, se puede cortar (puentear) la resistencia shunt de detección de corriente con el puente J19 para evitar la caída de voltaje en la resistencia. Ver la tabla a continuación.

Tabla 3.3 Configuración del DIP SW5.

Posición	Configuración	Descripción
1	OFF	Comunicación USB K20 deshabilitada, K20 en modo de parada de muy baja potencia (VLPS)
2	OFF	Salida del generador deshabilitada
1	ON	Comunicación USB y detección de corriente KM34 habilitada, LED verde encendido
2	ON	Generator outputenabled, amber LED ON

3.4 Configuración de la placa: Jumpers

La siguiente es una lista de todas las opciones de puentes en TWR-KM34Z50M. La configuración predeterminada de los puentes instalados se indica en la Tabla A con el texto en negrita.

El GPIO Header J10 (Tabla B) contiene pines de sabotaje, dos entradas analógicas, señal de verificación de salida CLK y la mayoría de los pines BUS de la interfaz periférica como SPI, I2C y Serial Communications Interface (SCI). Las señales del modulador externo AFE también están disponibles en este header, de modo que la placa personalizada con el modulador AFE externo se puede interconectar. La mayoría de los pines se pueden usar normalmente como GPIO. Los pines de sabotaje se pueden conectar a las salidas X, Y, Z del sensor de inclinación (U7), o al botón de sabotaje externo (SW3) utilizando jumpers.

El header J17 (Tabla C) es el header de las entradas analógicas / salida del generador que contiene las salidas del generador de señal auxiliar, las entradas analógicas de SD y los ADC SAR. Las entradas del ADC se pueden conectar directamente a las señales del generador o a las señales analógicas, usando jumpers. Si el ADC SD se usa para la medición en modo single-ended, los cables ADMx pueden conectarse a tierra VSSA utilizando jumpers.

3.4.1 Socket de complemento de torre de uso general (TWRPI)

El TWR-KM34Z50M presenta un socket (J11 y J12) que puede aceptar una variedad de diferentes módulos de complemento de torre con sensores, transceptores de RF y otros periféricos. El socket TWRPI de uso general proporciona acceso a I2C, SPI, IRQ, GPIO, temporizadores, señales de conversión analógicas, señales de identificación TWRPI, reinicio y suministros de voltaje. El pinout para el socket TWRPI se define en esta tabla.

Tabla 3.4 Pinout del socket de propósito general TWRPI.

J12-Pin	Descripción	J11-Pin	Descripción
1	Vcc 5 V	1	GND
2	Vcc 3.3 V	2	GND
3	GND	3	I2C: SCL
4	VDDA 3.3 V	4	I2C: SDA
5	VSS (GND analógico)	5	GND
6	VSS (GND analógico)	6	GND
7	VSS (GND analógico)	7	GND
8	CAD: Analog 0	8	GND
9	CAD: Analog 1	9	SPI: MISO
10	VSS (GND analógico)	10	SPI: M0SI
11	VSS (GND analógico)	11	SPI: SS
12	CAD: Analog 2	12	SPI: CLK
13	VSS (GND analógico)	13	GND
14	VSS (GND analógico)	14	GND
15	GND	15	GPIO: GPIO0/IRQ
16	GND	16	GPIO: GPIO1/IRQ
17	CAD: TWRPI ID 0	17	UART: UART RX or GPIO: GPIO2
18	CAD: TWRPI ID 1	18	UART: UART TX or GPIO: GPIO3
19	GND	19	UART: UART_CTS or GPIO: GPIO4/Timer
20	RESET	20	UART: UART_RTS or GPIO: GPIO5/Timer

3.4.2 Conexiones principales

El resto de conexiones con jumpers se adjuntan en el Anexo A.

Tabla 3.5 Jumpers de configuración.

Jumper	Descripción
J1	Conecta VBAT a la alimentación o a MCU_PWR/VBATD0
J2	Conecta/desconecta VREF del *ELEVATOR edge*
J6	Conecta MCU a V_BRD 3.3V o a voltaje externo
J7	Conecta/desconecta tensiones analógicas a MCU_PWR
J8	Conecta/desconecta PTG1/AD10 al potenciómetro R20
J9	Conecta/desconecta PTF0/AD7 al sensor de temperatura
J19	Anula/habilita la medición de corriente de baja potencia
J20	Conecta/desconecta la entrada de reset del KM34 al K20 OpenSDA

Tabla 3.6 Conexiones del Switch S1 DIP (Jumper S1).

Jumpers	Descripción
1-20	Conecta PTE5 al LED Verde (D3)
2-19	Conecta PTF1 al LED Rojo (D4)
3-18	Conecta PTD1 al LED Naranja (D5)
4-17	Conecta PTC1 al LED Amarillo (D6)
5-16	Conecta el LED IrDA Tx (D7) al PTC2
6-15	Conecta el Transistor IrDA Rx (Q1) al PTC3
7-14	Conecta el dato EEPROM serie al dato I2C0
8-13	Conecta el reloj EEPROM serie al reloj I2C0
9-12	Conecta la señal TILT_ENABLE (MMA8491Q) al PTF7
10-11	Conecta la señal de medida de tensión VBAT al PTF2/AD9

4 Descripción de la metodología

La metodología realizada durante el proyecto fue la siguiente:

- Definir un nuevo concepto a implementar. Por ejemplo: un módulo de metrología.
- Crear una nueva rama de desarrollo en el repositorio para esta nueva feature siguiendo Git Flow.
- Realizar commits atómicos para cada tarea concreta o modificación. Por ejemplo: Renombrar una variable en todos los ficheros, corregir headers, etc… en las que se incluyen:
  - Replicar la estructura de carpetas/ficheros (descrita posteriormente) según la nomenclatura establecida.
  - Desarrollar el código necesario (generalmente C o shell script).
  - Escribir de forma simultánea su documentación (en el caso de C se usará Doxygen).
- Finalizar cerrando la rama de desarrollo y haciendo release.
  - Si el desarrollo no rompe retro-compatibilidad, según SemVer, se incrementa sólo la segunda cifra de la versión (siendo la tercera en caso de hotfix)
  - Si por el contrario se rompe la retro-compatibilidad, necesariamente la primera cifra se debe incrementar (dejando la segunda y tercera a 0)
- Verificación de resultados y estado de artefactos de CI/CD.

Como hemos apuntado anteriormente, hemos usado Semantic Version (semver) para nombrar las distintas versiones, y Git flow, para las ramas master, develop, feature y hotfix del proyecto.

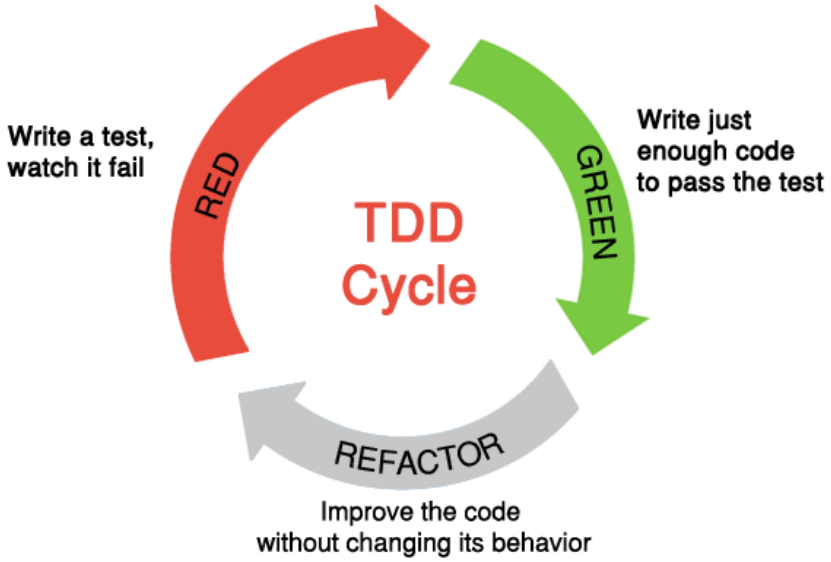

Figura 4.1 Ciclo de Test-Driven Development(TDD).

La metodología está basada también en Test-Driven Development (TDD) o desarrollo conducido a través de tests. De forma que primero diseñamos el test, luego añadimos el código funcional para pasar el test y por último mejoramos el código sin cambiar su comportamiento.

La disposición de archivos seleccionada para este proyecto intenta cubrir varios aspectos:

- Cada módulo solo dependerá de sí mismo, pero no del código de nivel superior en este proyecto. Por ejemplo: Un driver no puede depender del código de un test.
  - Esto permite incluir libremente los módulos en cualquier otro software.
- Como entidad independiente, pero perteneciente a un proyecto, cada módulo contendrá automáticamente todos los archivos y encabezados necesarios, pero los encabezados estarán dentro de una carpeta `mculib`.
  - El encabezado del primer nivel del módulo estará allí, por lo que funcionará `#include "mculib / mod_ab.h"`.
  - Cualquier otro encabezado puramente relacionado con el módulo, estará disponible dentro de una subcarpeta de una manera que funcione `#include "mculib / mod_ab / something.h"`.
- Mirando desde el punto de vista del proyecto, cada módulo y dependencias se encontrarán sin problemas.

El siguiente apartado resume cómo se organizan los archivos dentro de esta carpeta / repositorio:

4.1 Estructura de directorios

Tenemos un directorio por cada módulo donde vamos a realizar los test, cada uno con una estructura similar

```
.
|--- appconfig.h
|-- com
|   |-- com.c
|   |-- debug.c
|   |-- driver
|   |   |-- i2c.c
|   |   |-- loopback.c
|   |   |-- spi.c
|   |   --- uart.c
|   |-- mculib
|   |   |-- com
|   |   |   |-- debug.h
|   |   |   |-- driver
|   |   |   |   |-- i2c.h
|   |   |   |   |-- loopback.h
|   |   |   |   |-- spi.h
|   |   |   |   --- uart.h
|   |   |   --- driver.h
|   |   --- com.h
|   --- test
|       |-- test_com_i2c.c
|       |-- test_com_loopback.c
|       |-- test_com_spi.c
|       --- test_com_uart.c
|-- common
|   |-- debug.c
|   |-- mculib
|   |   --- debug.h
|   --- unit_test
|       --- unit_test.h
|-- freemaster_cfg.h
```

```
|-- gpio
|   |-- debug.c
|   |-- driver
|   |   --- mkm34_gpio.c
|   |-- gpio.c
|   |-- mculib
|   |   |-- gpio
|   |   |   |-- debug.h
|   |   |   |-- driver
|   |   |   |   --- mkm34_gpio.h
|   |   |   --- driver.h
|   |   --- gpio.h
|   --- test
|       --- test_gpio_mkm34.c
|-- metering
|   |-- driver
|   |   |-- mkm34z128_meterlib.c
|   |-- mculib
|   |   |-- metering
|   |   |   |-- driver
|   |   |   |   --- mkm34z128_meterlib.h
|   |   |   --- driver.h
|   |   --- metering.h
|   |-- metering.c
|   --- test
|       --- test_metering_mkm34.c
|-- TWR-KM34Z50M_unit-tests.c
```

Cada módulo está compuesto por los siguientes directorios :

- driver : Donde se encuentra el archivo de código .c del driver.
- mculib : Aquí están todas las cabeceras o includes .h del módulo.
- test : Donde está el fichero o ficheros de test de prueba del módulo.

Fuera de los directorios de módulos, está el fichero top main donde usamos todos los test así como el fichero de cabecera de configuración del micro.

4.2 Función de depuración

Definimos una función de depuración _dbg(...) en `common/mculib/debug.h` que incluiremos en todos nuestros ficheros para poder hacer uso de ella. Ofrece las siguientes posibilidades:

- `MCULIB_DEBUG` no definido, en este caso, se obvia la función de depuración y no se muestran los resultados por ninguna salida.
- `MCULIB_DEBUG` definido, con 2 variantes:
  - `PRINTF` no definido, usamos la función printf estándar para mostrar los resultados.
  - `PRINTF` definido, usamos esa función de `PRINTF` para mostrar los resultados.

Nuestro caso es este último, hemos definido `PRINTF` como una función que envía los resultados a través del puerto serie, para que podamos visualizarlos mediante el ordenador.

5 Descripción de los tests unitarios

Los tests unitarios realizados son los siguientes :

5.1 Módulo de GPIO

GPIO - General Purpose Input Output - Salidas y entradas de propósito general, es un sistema de pines que pueden ser configurados tanto de entradas como de salida para múltiples usos.

El objetivo del test es verificar el correcto arranque y programación del micro así como la inicialización y el uso de los periféricos. Tenemos dos posibles modos de ejecución del test, por un lado, manual, donde el usuario es el que pulsa el botón y por otro lado, automático, donde un pin secundario conectado físicamente a la salida del botón, simulará la acción del usuario.

Como hemos apuntado en la metodología se ha seguido un desarrollo basado en TDD :

- Estudio y análisis de las posibles partes comunes y específicas, a depender del fabricante u otra condición.
- Implementación de funciones como operaciones del driver: configuración, escritura y lectura

Código 5.1 Funciones del driver del módulo de GPIO.

```
1  /**
2   * @brief     Configure a 'gpio'
3   * @details   Checks and initializes 'gpio' common data, then calls the lower
4   *            level init function passed by argument.
5   *
6   * @param      gpio             Pointer to a gpio
7   * @param[in]  gpio_driver_init Pointer to a driver initializer function
8   * @param[in]  driver_data      Pointer to a driver data structure
9   *
10  * @return    Success if 0, otherwise error
11  */
12 int gpio_setup(struct gpio *gpio,
13                int (*gpio_driver_init)(struct gpio *, void *),
14                void *driver_data);
15
16 /**
17  * @brief     Write 'value' in the gpio pin
18  *
19  * @param     gpio      Pointer to a correctly initialized gpio
20  * @param[in] value     Pointer to value data
21  *
22  * @return    Success if 0, otherwise error
23  */
```

```
24 int gpio_write(struct gpio *gpio, uint8_t *value);
25
26 /**
27  * @brief     Read 'value' from the gpio pin
28  *
29  * @param     gpio      Pointer to a correctly initialized gpio
30  * @param[in] value     Pointer to value data
31  *
32  * @return    Success if 0, otherwise error
33  */
34 int gpio_read(struct gpio *gpio, uint8_t *value);
```

- Se implementa un estructura de driver gpio así como una estructura secundaria donde salvar las configuraciones.

Código 5.2 Estructura del driver del módulo de GPIO.

```
1  /**
2   * @brief     Structure that defines a gpio
3   */
4  struct gpio
5  {
6    /**
7     * @brief     Pointer to a operations driver_data structure
8     */
9    struct driver_data *driver_data;
10 };
```

- Diseño de unos tests preliminares para comprobar el estado de compilación y de respuesta en casos afirmativos y positivos.
- Implementación del mock.
- Implementación del test, de esta forma cumplimos el ciclo Dev-Mock-Test-Restart.
- Por último, implementación de un driver específico, para esta placa de desarrollo y este micro concreto.

5.2 Módulo de Comunicaciones

Como hemos apuntado en la metodología se ha seguido un desarrollo basado en TDD :

- Estudio y análisis de las posibles partes comunes y específicas, a depender del fabricante u otra condición.
- Implementación de funciones como operaciones del driver: configuración, enviar y recibir datos.

Código 5.3 Funciones del driver del módulo de Comunicaciones.

```
1  /**
2   * @brief     Configure a 'com'
3   * @details   Checks and initializes 'com' common data, then calls the lower
4   *            level init function passed by argument.
5   *
6   * @param     com              Pointer to a communication instance
7   * @param     com_driver_init  Pointer to a driver initializer function
8   * @param     driver_data      Pointer to a driver data
```

```
 *
 * @return    Success if 0, otherwise error
 */
int com_setup(struct com *com, int (*com_driver_init)(struct com *, void *),
              void *driver_data);

/**
 * @brief     Send a buffer by the driver
 *
 * @param     com      Pointer to a correctly initialized communication
 *                     instance
 * @param[in] buffer   Buffer to be sent
 * @param[in] size     Total amount of space on buffer to send
 *
 * @return    Success if 0, otherwise error
 */
int com_send(struct com *com, uint8_t *buffer, size_t size);

/**
 * @brief     Reveive a buffer by the driver
 *
 * @param     com      Pointer to a correctly initialized communication
 *                     instance
 * @param[in] buffer   Buffer to be received
 * @param[in] size     Total amount of space on buffer to receive
 *
 * @return    Success if 0, otherwise error
 */
int com_receive(struct com *com, uint8_t *buffer, size_t size);
```

- Se implementa un estructura de driver com así como una estructura secundaria donde salvar las configuraciones.

Código 5.4 Estructura del driver del módulo de Comunicaciones.

```
/**
 * @brief     Structure that defines a communication instance
 */
struct com
{
  /**
   * @brief     Pointer to real reserved data holder (provided by the user)
   */
  uint8_t *buffer;
  /**
   * @brief     Total amount of space on buffer that we can use
   */
  size_t size;
  /**
   * @brief     Pointer to a operations driver structure
   */
  struct com_driver *driver;
};
```

- Diseño de unos tests preliminares para comprobar el estado de compilación y de respuesta en casos afirmativos y positivos.

- Implementación del mock.
- Implementación del test, de esta forma cumplimos el ciclo Dev-Mock-Test-Restart.
- Por último, implementación de un driver específico, para esta placa de desarrollo y este micro concreto.

A continuación describiremos los 3 drivers de comunicaciones implementados:

5.2.1 Puerto serie

Cuando hablamos de puerto serie en todo momento y salvo que especifiquemos lo contrario nos estamos refiriendo a UART.

UART- Universal Asynchronous Receiver-Transmitter - Transmisor/Receptor Asíncrono Universal. Este protocolo utiliza solo dos pines, Rx y TX. Como aquí no se necesita ningún reloj, ambos dispositivos deben hacer uso de sus relojes internos independientes para funcionar. Sin embargo, existe un término llamado tasa en baudios que ayuda a que estos dispositivos permanezcan sincronizados al fijar la velocidad del intercambio de datos. La velocidad en baudios se refiere a la cantidad de bits de datos transmitidos por segundo, por lo que ambos dispositivos deberían tener la misma velocidad en baudios para conseguir un correcto funcionamiento.

La gran limitación de UART es que solo dos dispositivos pueden comunicarse usando este protocolo a la vez. El pin TX de un dispositivo transmite datos al pin RX de otro dispositivo y, de forma similar, el TX de este último transmite datos al RX del dispositivo anterior. Así es como se produce el intercambio de datos.

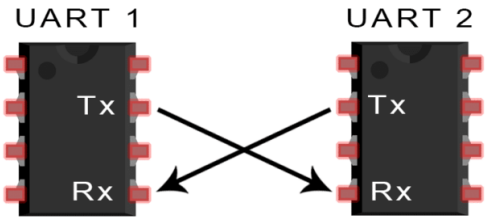

Figura 5.1 Estructura UART.

El test del puerto serie fue uno de los primeros, y es de los más importantes ya que necesitamos esa interfaz para poder comunicarnos con el micro, de forma que podamos depurar y observar los resultados de los demás tests.

Como hemos explicado en el punto de hardware, nuestra placa posee 4 UARTs, siendo la 1 la que está conectada al USB y es accesible desde el ordenador, la utilizaremos para volcar los resultados por pantalla. Las otras 3 están accesibles en pines, en este caso utilizaremos la 0 para unir RX y TX y formar un bucle, de forma que lo que enviemos por un lado lo podamos recibir. Así verificamos el funcionamiento completo del sistema.

Tras una breve búsqueda por los documentos del micro, observamos que RX y TX de la UART0 están disponible a través de los pines PTF4 Y PTF3 respectivamente. Si los configuramos correctamente, están accesibles en el jumper J10, pines 19 y 20. Con un cable cortoircuiramos estos dos pines.

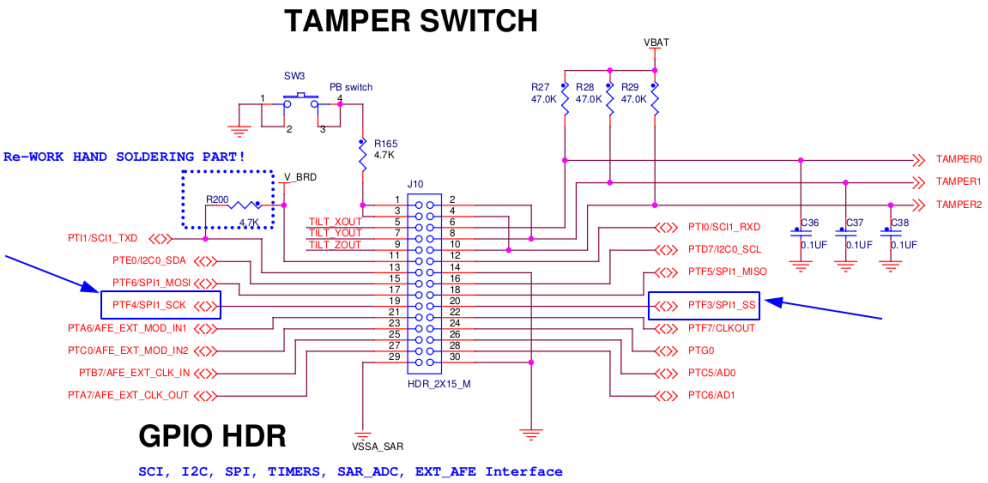

Figura 5.2 Pines asignados a la UART0 en el conector J10.

Siguiendo la estructura del tests, iniciamos hardware, tras realizar varias comprobaciones enviamos una cadena de caracteres y la recibimos de la forma anteriormente explicada, para concluir verificando que lo que recibimos es lo mismo que enviamos.

5.2.2 I2C

I2C - Inter-Integrated Circuit - Circuito inter integrado. Este protocolo utiliza dos líneas para todo el proceso: SDA (datos en serie) y SCL (reloj en serie). I2C puede admitir múltiples dispositivos esclavos, pero a diferencia de SPI, que solo admite un dispositivo maestro, I2C también puede admitir múltiples dispositivos maestros.

Cada dispositivo envía / recibe datos usando solo una línea que es SDA mientras que SCL mantiene la sincronización entre los dispositivos a través del reloj común que proporciona el maestro activo.

Cada mensaje se inicia con una condición de inicio y finaliza con una condición de detención. Un solo mensaje puede contener múltiples bytes de datos, cada uno con un bit de reconocimiento (ACK) o reconocimiento negativo (NACK) entre ellos.

Las resistencias pull-up con SDA y SCL son necesarias para ejecutar este protocolo.

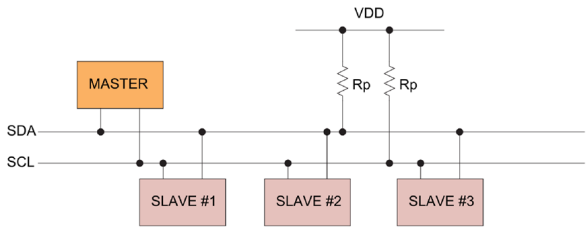

Figura 5.3 Estructura I2C.

Para realizar el test i2c vamos a hacer uso de una memoria EEPROM que viene incorporada en nuestra placa, una CAT24C05 con la que nos podemos comunicar a través del bus i2c. Tenemos que utilizar también la uart ya que es la única forma que disponemos por el momento para depurar y verificar que el test funciona como debería. Por lo tanto, iniciamos hardware, así como uart e i2c y seguimos la estructura similar de todos los tests. En este caso, la función de enviar dentro del driver de i2c es una escritura en la EEPROM y la función de recibir es una lectura de la misma memoria a través del mismo bus i2c. Por último como en todos los tests, verificamos que recibimos lo mismo que enviamos y damos el test por concluido

A destacar de esa memoria EEPROM, que posee un buffer de entrada de 16 bytes, de forma que si enviamos más datos juntos sobreescribimos los primeros. La forma correcta de enviar más bytes sería mandarlos en grupos de 16, para que la memoria pudiera procesar los datos y vaciar el buffer de entrada antes de que llegaran los nuevos 16 bytes.

5.2.3 SPI

SPI - Serial Peripheral Interface - Interfaz de Periféricos Serie - Es un protocolo de comunicación en serie de tipo síncrono que surgió a principios de 1980 en Motorola y que consta de dos líneas de datos (MOSI y MISO), una línea de reloj (SCK) y una línea de selección esclava (SS).

- MOSI (Master Out Slave In):. Línea utilizada para llevar los bits que provienen del maestro hacia el esclavo.
- MISO (Master In Slave Out):. Línea utilizada para llevar los bits que provienen del esclavo hacia el maestro.
- CLK (Clock):. Línea proviniente del maestro encarga de enviar la señal de reloj para sincronizar los dispositivos.
- SS (Slave Select):. Línea encargada de seleccionar y a su vez, habilitar un esclavo.

Toda la comunicación es manejada por el maestro; ningún esclavo puede enviar datos por su propia voluntad. El maestro envía datos a través de MOSI mientras los esclavos responden a través de la línea MISO. En todo el proceso SCK (reloj serie) juega un papel muy importante, cada dispositivo esclavo depende de este reloj para leer datos de MOSI y responder a través de MISO. SS (Slave Select) se usa para hacer que un esclavo en particular esté despierto y el maestro quiera comunicarse.

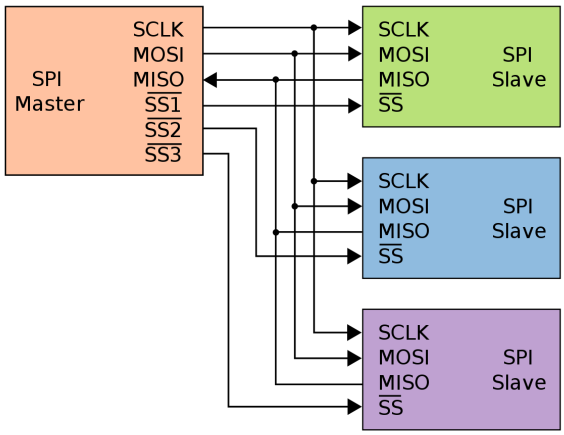

Figura 5.4 Estructura SPI.

El objetivo de este test es comprobar el funcionamiento del bus SPI. En este caso al no tener ningún dispositivo SPI conectado más allá del propio micro, sólo pudimos realizar el código y compilarlo, a la espera de poder probarlo en algún momento. El test sigue la misma estructura que los demás. Utilizamos la UART como interfaz de depuración para poder ver los resultados en el ordenador. En este caso la función de envío y la de recepción es a través del bus SPI. Terminamos verificando que lo que enviamos es igual que lo que recibimos.

5.3 Módulo de Metrología

Como hemos apuntado a lo largo de todo el documento, hemos elegido este micro por sus características metrológicas, está diseñado específicamente para eso. La metrología es la ciencia de la medida, y en este caso, nos referimos a magnitudes eléctricas: tensión y corriente.

La alta precisión es una característica esencial de una contador de energía. La precisión es un atributo muy importante porque una medición inexacta puede resultar en cantidades sustanciales de pérdida de ingresos. Además, una medición inexacta también puede resultar indeseablemente en una sobrecarga de la red. Las fuentes comunes de inexactitud en la medida incluyen los dispositivos del sensor, los circuitos de acondicionamiento, el Analog Front-End (AFE) y el algoritmo de medición ejecutado normalmente en un microcontrolador.

La tarea crítica para el microcontrolador en una aplicación de medición es el cálculo preciso de energía activa, energía reactiva, potencia activa, potencia reactiva, potencia aparente, voltaje RMS y corriente RMS. Las energías activa y reactiva a veces se denominan cantidades de facturación. Sus cálculos deben cumplir con las normas europeas EN50470-1 y EN50470-3 para contadores electrónicos de clase de energía activa B y C, y las normas internacionales IEC 62053-21 e IEC 62052-11 para contadores electrónicos de clases de energía activa 2 y 1, y la norma internacional IEC 62053-23 para contadores estáticos de clases de energía reactiva 2 y 3. Las cantidades restantes se calculan con fines informativos y se denominan no facturables.

Los algoritmos de medida realizan cálculos en el dominio del tiempo o de la frecuencia. Nosotros nos hemos basado en un algoritmo de medición con filtros. Este algoritmo calcula todas las cantidades facturables y no facturables en el dominio del tiempo, con un amplio soporte de los filtros digitales de Respuesta de Impulso Finito (FIR) y Respuesta de Impulso Infinito (IIR).

El algoritmo de medición basado en filtros puede ser fácilmente integrado en una aplicación de medidor de potencia electrónico. El algoritmo solo requiere que se proporcionen muestras instantáneas de voltaje y corriente a intervalos de muestreo constantes. Estas muestras instantáneas de voltaje y corriente las tomaremos mediante un AFE con la ayuda de un divisor resistivo, en el caso de una medición de voltaje de fase, y una resistencia shunt, transformador de corriente o una bobina Rogowski en el caso de una medición de corriente de fase. Todos los sensores de medición de corriente introducen un cambio de fase en la medición. Por lo tanto, es necesario alinear las fases de las muestras instantáneas de voltaje y corriente utilizando el método software de corrección de fase incluido en el algoritmo de medición basado en filtros o con la ayuda de muestreo retardado antes de usarlos.

La herramienta de configuración de software permite configurar fácilmente el algoritmo. Está diseñada para ajustar los filtros digitales para que coincidan con el rendimiento requerido y generar un archivo de encabezado C con datos de configuración específicos para el tipo de medidor de potencia. La herramienta automatiza el procedimiento de configuración y optimización del algoritmo, al tiempo que proporciona una estimación aproximada de la carga computacional requerida.

La siguiente figura muestra un diagrama de bloques del algoritmo de medición basado en filtros en una aplicación típica de medidor de potencia monofásica. Las medidas de corriente y voltaje están representadas por fuentes de señal i(t) y u(t). Estas fuentes proporcionan muestras instantáneas de tensión y corriente alineadas en fase a intervalos de muestreo constantes. Las nuevas muestras de voltaje y corriente activan un nuevo cálculo de todos los bloques de algoritmos. Después de cada nuevo cálculo, estarán disponibles nuevas cantidades facturadas y no facturadas. Todas las cantidades calculadas generalmente se muestran en la pantalla LCD y se archivan en una base de datos para su posprocesamiento y lectura a través de la interfaz de comunicación de lectura automática del medidor (AMR). Además, las energías activa y reactiva también controlan sus respectivos LED de salida de pulsos para fines de calibración y prueba.

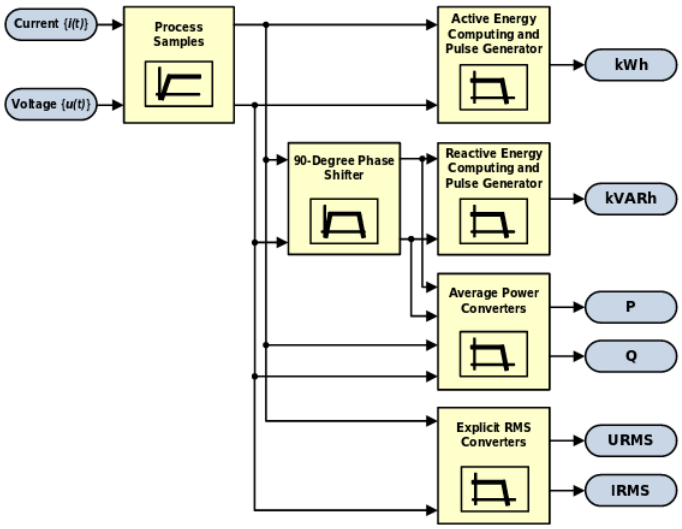

Figura 5.5 Diagrama de bloques del algoritmo de medida basado en filtros.

Como hemos apuntado en la metodología se ha seguido un desarrollo basado en TDD :

- Estudio y análisis de las posibles partes comunes y específicas, a depender del fabricante u otra condición.
- Implementación de funciones como operaciones del driver: configuración, medida de todos los armónicos, medida de tensión RMS, medida de instensidad RMS, medida de potencia activa (P), medida de potencia reactiva (Q) y medida de potencia aparente (S).

Código 5.5 Funciones del driver del módulo de Metering.

```
1  /**
2   * @brief     Configure a 'metering'
3   * @details   Checks and initializes 'metering' common data, then calls the
4   *            lower level init function passed by argument.
5   *
6   * @param     meter                Pointer to a metering structure
7   * @param     metering_driver_init Pointer to a driver initializer function
8   * @param     driver_data          Pointer to a driver data
9   *
10  * @return    Success if 0, otherwise error
11  */
12 int metering_setup(struct metering *meter,
13                     int (*metering_driver_init)(struct metering *, void *),
14                     void *driver_data)
15
16 /**
17  * @brief     Read all harmonics
18  *
19  * @param     meter      Pointer to a correctly initialized metering
20  * @param[in] harms      Pointer to struct to write data
21  *
22  * @return    Success if 0, otherwise error
23  */
24 int metering_read_harmonics(struct metering *meter, struct harms *harms);
25
26 /**
```

```
27  * @brief     Read voltage RMS
28  *
29  * @param     meter      Pointer to a correctly initialized metering
30  * @param[in] voltage    Pointer to struct to write data
31  *
32  * @return    Success if 0, otherwise error
33  */
34 int metering_read_VRMS(struct metering *meter, struct voltage *voltage);
35
36 /**
37  * @brief     Read current RMS
38  *
39  * @param     meter      Pointer to a correctly initialized metering
40  * @param[in] current    Pointer to struct to write data
41  *
42  * @return    Success if 0, otherwise error
43  */
44 int metering_read_IRMS(struct metering *meter, struct current *current);
45
46 /**
47  * @brief     Read P
48  *
49  * @param     meter      Pointer to a correctly initialized metering
50  * @param[in] p          Pointer to double to write data
51  *
52  * @return    Success if 0, otherwise error
53  */
54 int metering_read_P(struct metering *meter, double *p);
55
56 /**
57  * @brief     Read Q
58  *
59  * @param     meter      Pointer to a correctly initialized metering
60  * @param[in] q          Pointer to double to write data
61  *
62  * @return    Success if 0, otherwise error
63  */
64 int metering_read_Q(struct metering *meter, double *q);
65
66 /**
67  * @brief     Read S
68  *
69  * @param     meter      Pointer to a correctly initialized metering
70  * @param[in] s          Pointer to double to write data
71  *
72  * @return    Success if 0, otherwise error
73  */
74 int metering_read_S(struct metering *meter, double *s);
```

- Se implementa un estructura de driver metering así como una estructura secundaria donde salvar las configuraciones.

Código 5.6 Estructura del driver del módulo de Metering.

```
1  /**
2   * @brief     Structure that defines a metering
3   */
4  struct metering
5  {
6    /**
7     * @brief     Pointer to calibration function
8     */
9    struct cal_data *cal;
10   /**
11    * @brief     Pointer to driver structure
12    */
13   struct metering_driver *diver;
14 }
```

- Diseño de unos tests preliminares para comprobar el estado de compilación y de respuesta en casos afirmativos y negativos.
- Implementación del mock.
- Implementación del test, de forma que cumplamos el ciclo Dev-Mock-Test-Loop.
- Por último, implementación de un driver específico, para esta placa de desarrollo y este micro concreto.

Para poder llevar a cabo este test necesitamos:

- Configurar filtros para generar librería.
- Configurar generador de señales K20.
- Usar las librerías meterlib y fraclib junto con la generada con cfg_tool para poder medir
- Usar puerto serie para poder volcar los resultados.

Como hemos apuntado, primero configuramos los filtros con la herramienta configuration tool, quedando de la siguiente forma:

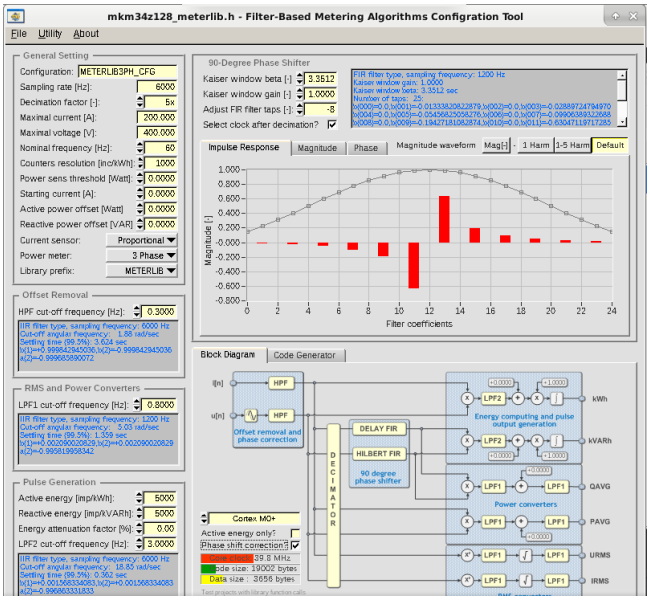

Figura 5.6 Configuración de filtros.

El siguiente paso fue configurar un proyecto de FreeMASTER para ser capaces de usar el generador de señales K20 incorporado en la placa. Para ello, primero conectamos la placa (dos cables USB) e iniciamos la comunicación para que el programa la detecte. Una vez realizado esto, ya tenemos disponibles las variables de las 7 posibles señales del generador que podemos configurar a la vez. Podemos fijar parámetros generales como la frecuencia o si son rectificadas, así como otros parámetros de cada una como la magnitud y fase. También podemos definir hasta 3 armónicos: 3,5 y 7 su magnitud y su fase.

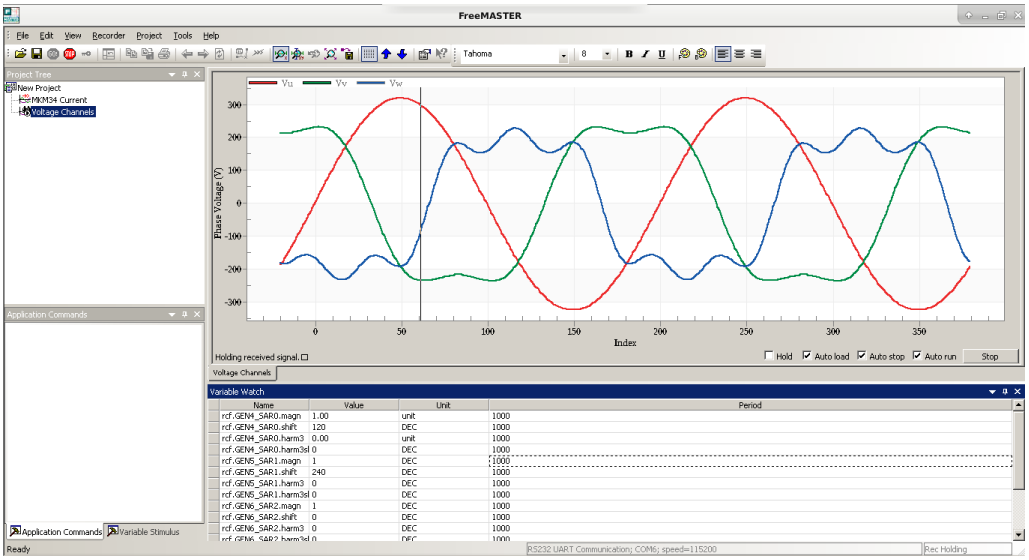

Figura 5.7 Interfaz de FreeMASTER configurado para generar señales con armónicos.

El siguiente paso es arrancar el test en el KDS para correrlo en la placa, cargamos el fichero de configuración de los filtros que hemos definido en cfg_tool y usamos las librerías meterlib y fraclib. Con todas estas herramientas somos capaces de realizar una gran variedad de medidas como: valores RMS y media de tensiones e intensidades de cada una de las fases, potencias activa, reactiva y aparente.

En este punto, cabe destacar que para que el generador k20 esté conectado con los CAD de la placa tenemos conectar los jumpers J17 de la placa :

Tabla 5.1 Configuración de pines generador K20 (J17).

J17 Pin	MCU Signal
1	GEN_OUT0
2	EXT_SD_ADP0
3	GEN_OUT1
4	EXT_SD_ADP1
5	GEN_OUT2
6	EXT_SD_ADP2
7	GEN_OUT3
8	EXT_SD_ADP3
9	GEN_OUT4
10	EXT_SAR_AD0
11	GEN_OUT5
12	EXT_SAR_AD1
13	GEN_OUT6
14	EXT_SAR_AD2

Conectamos 1 con 2, 3 con 4 y así sucesivamente con las 7 posibles señales generadas. Así como verificamos que SW5 que es el que habilita el generador está en ON.

6 Resultados

Los resultados se exponen a continuación:

6.1 Módulo de GPIO

Se ha obtenido una unidad llamada gpio para la que hemos diseñado pruebas unitarias, esta unidad se concluye completamente testeada y con mock por lo que ya es posible incluirla en CI/CD. Además, el driver de prueba de concepto hardware tambien se demuestra que funciona correctamente, detectando como acierto presionar un botón y como fallo si no se hace

6.2 Módulo de Comunicaciones

Hemos obtenido una unidad denominada com para la que hemos diseñado pruebas unitarias, esta unidad sale completamente testeada y con mock por lo que ya es posible incluirla en CI/CD. Además, los drivers UART e I2C se demuestra que funcionan, el primero con un cortocircuito físico y el segundo a través de una memoria EEPROM. No se ha podido verificar el driver SPI al no disponer de ningún dispositivo con esa interfaz de comunicaciones

6.3 Módulo de Metrología

Se ha obtenido una unidad llamada metering para la que hemos diseñado pruebas unitarias, esta unidad se concluye completamente testeada y con mock por lo que ya es posible incluirla en CI/CD. Además, usando la herramienta FreeMASTER configuramos y generamos las señales deseadas y, con el driver de metering del KM34, somos capaces de realizar las medidas. Salvo discrepancias por falta de calibración, no apreciamos ningún error en la medida.

6.4 Documento TFG

En el desarrollo del proyecto no se llega a implementar {CI/CD sobre el código, pero sí se monta una prueba de concepto de forma paralela, sólo que, orientada a documentación. Hemos realizado otro repositorio para ir escribiendo este propio documento, siguiendo el formato de la escuela. En este caso, la metodología es escribir cada apartado en un archivo de markdown y en un fichero de configuración indicar el orden de enlazado de documentos.

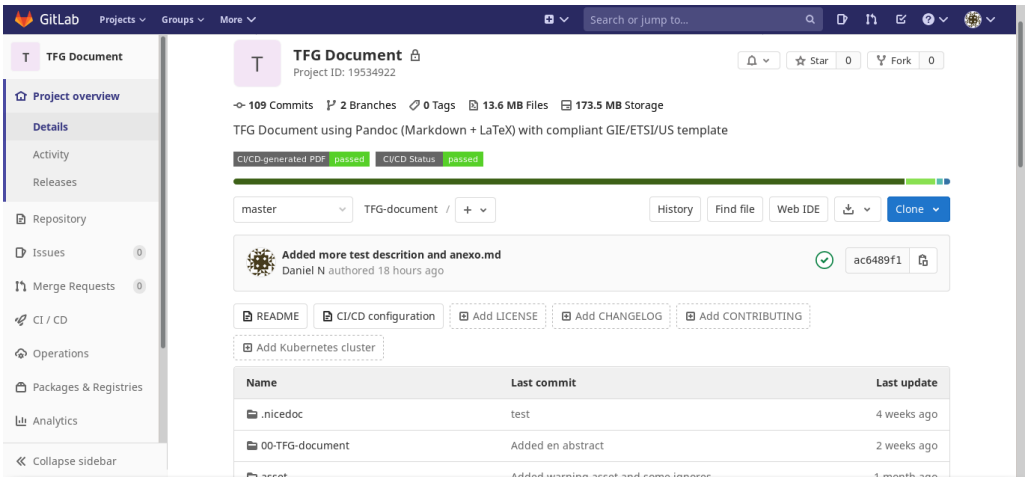

Figura 6.1 Repositorio de documentación TFG.

Como características:

1. Se ha escrito casi al completo en Markdown, con pequeñas cuñas de LaTeX.
2. Una vez preparado el nuevo contenido, se envía al servidor de Git.
3. Si el commit pertenece a las ramas `mastero` `develop` se lanzarán los jobs del CI/CD.
4. Una vez terminadas las pruebas y conversión del documento, éste se genera como artefacto en PDF, comprimido en ZIP y 100 % relacionado con los commits.

Cada vez que subimos un nuevo commit se ejecuta el siguiente pipeline con 3 etapas:

- Preparación de la estructura de carpetas y ficheros para la compilación.
- Generación del PDF.
- Despliegue del artefacto liberado, para tenerlo accesible desde fuera.

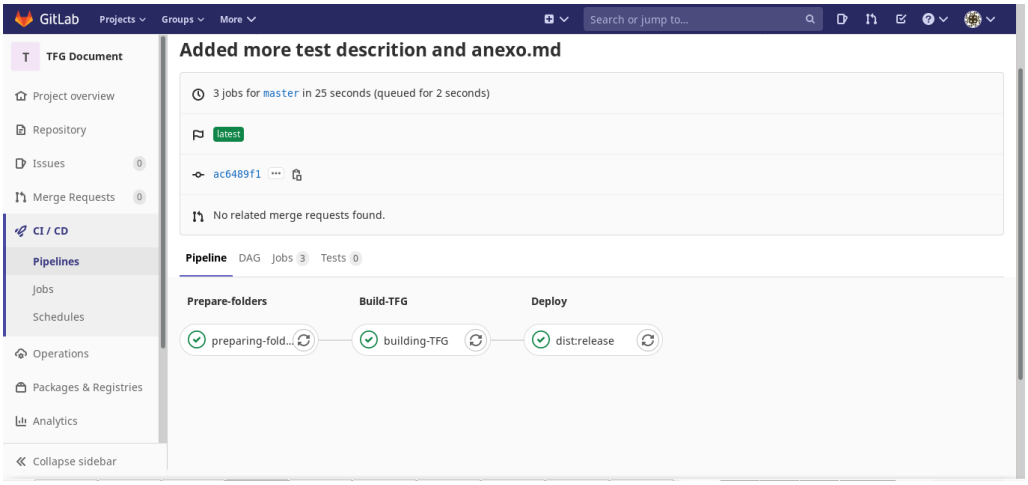

Figura 6.2 Pipeline del repositorio de documentación TFG.

De esta manera, una vez que está todo configurado, sólo nos tenemos que limitar a escribir cada nuevo apartado y al subirlo, se compilará y se generará un fichero PDF como artefacto. Este fichero PDF es el documento completo con la plantilla de la escuela. También se han configurado lo que GitLab llama "badgets" (imágenes genradas en base a unos criterios configurables), de forma que tanto informen del estado de la pipeline, mientras se está ejecutando :

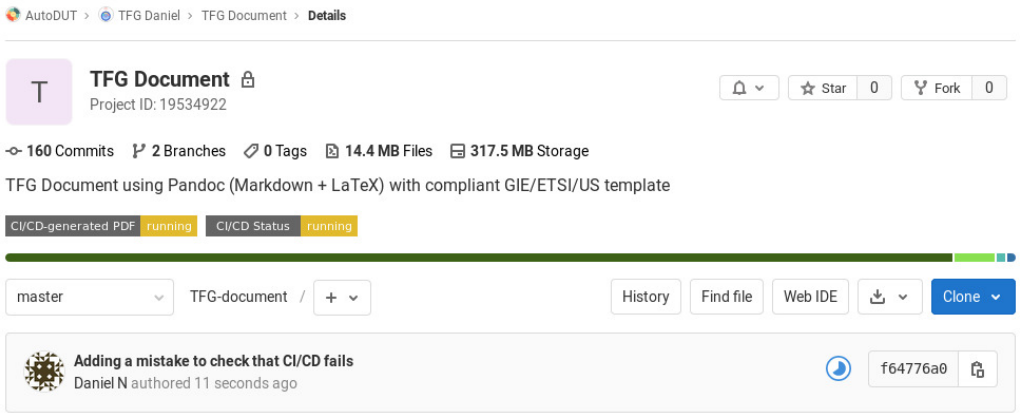

Figura 6.3 Ejecutando pipeline.

Si no se pudiera compilar, se detiene la pipeline, informando por email al usuario que hizo el commit del error.

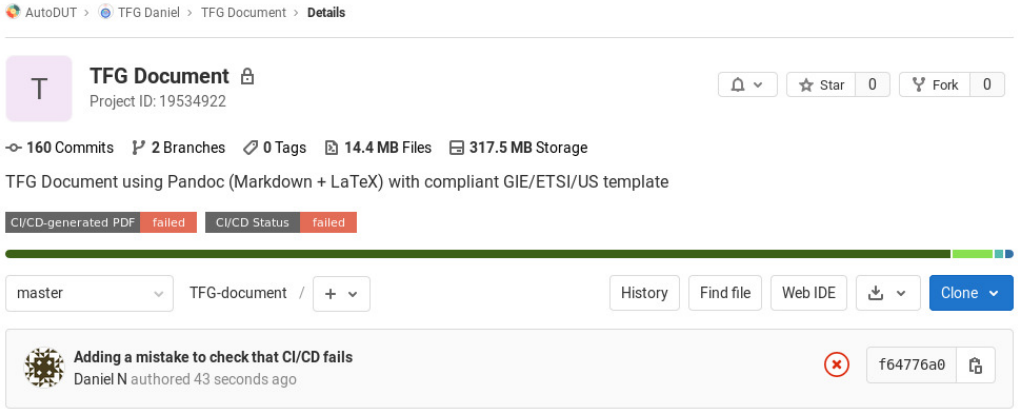

Figura 6.4 Fallo de pipeline.

Si no hay ningún problema, se puede descargar el último artefacto liberado en el momento del despliegue.

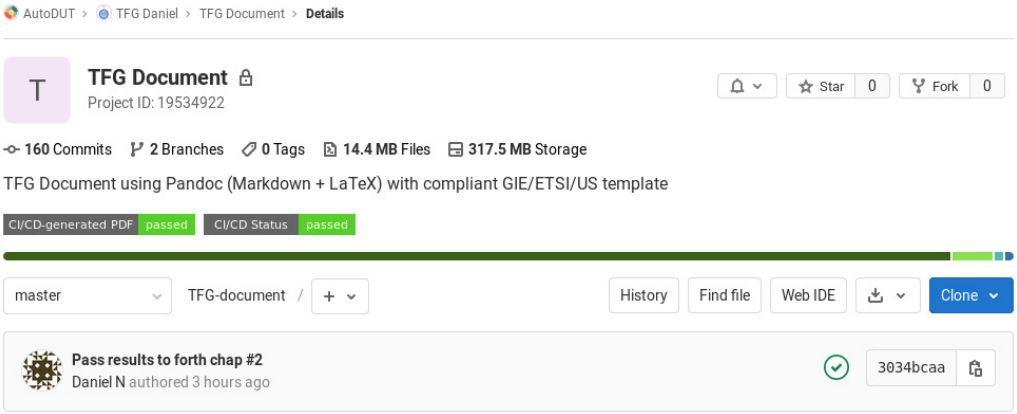

Figura 6.5 Pipeline terminado sin fallos, artefacto disponible.

7 Conclusiones

Se ha desarrollado un proyecto basado en TDD orientado a entornos embebidos, mediante la adquisición de una metodología de trabajo con un foco muy fuerte en la automatización, verificación y documentación.

Además de aprender y usar Linux como sistema operativo, preparado para desarrollo de software y firmware, también se han adoptado rutinas de trabajo fundamentadas en el ciclo desarrollo-test-refactor, gestionado mediante Git como sistema de control de versiones, GitLab como servidor y launcher de CI/CD y Gitkraken para facilitar la gestión de ramas y el uso de Git Flow y Semantic Version.

Para ello se ha diseñado e implementado una capa de abstracción de software, de forma que una vez se defina la API, una aplicación de un nivel superior sea capaz de acceder a un hardware modelado en un nivel inferior, sin que por esto necesite saber detalles concretos del hardware. Junto a esto, y siguiendo la ya mencionada metodología TDD, se diseñan una serie de tests unitarios para validar dichas capas media y baja, dejando un sistema testeado y verificado listo para usarse en una capa de nivel superior.

Todo lo anterior se ha aplicado a un sistema embebido en forma de placa de evaluación, concretamente la TWR-KM34Z50M, dejándose modelados y comprobados módulos de comunicaciones serie e I2C, como los módulos de GPIO y Metrología.

De forma paralela, también se ha aplicado orientándolo a documentación, ya que concretamente este documento TFG se ha desarrollado con la misma metodología, sólo que además se ha configurado correctamente el sistema de CI/CD de forma que cada versión del documento se comprueba, compila y genera versión en forma de artefacto PDF de forma automática.

8 Líneas futuras

Una vez analizado y establecido el flujo de trabajo y desarrollo sobre micros de la familia KM34, y tras la evaluación de sus periféricos y funcionalidades, se establece como siguiente paso la migración a la familia KM35, que está 100 % soportada por NXP y además dispone de un periférico dedicado para las operaciones de cálculo y filtrado relacionadas con el metering (Metering specific Memory Mapped Arithmetic Unit (MMAU)).

Una vez analizada la oferta de herramientas provista por NXP se procederá a aislar el toolchain de forma que todo el proyecto pueda procesarse completamente sin el uso de herramientas gráficas.

Es entonces cuando se creará un contenedor de Docker que contendrá un sistema mínimo funcional de compilación. Esto por un lado evita la necesidad de instalar todo el set completo de aplicaciones en un sistema diferente, y por otro permite la automatización del proceso, permitiendo por tanto incluirlo como parte del flujo de cara a CI/CD.

Este flujo se diseñará de forma que:

- Se ejecuten los tests automáticos pertinentes sobre un código recien subido en el momento del commit/push.
- Se verifique su funcionamiento y calidad, y en caso de no existir problemas, genere los artefactos necesarios.
- Cada release irá asociada a su artefacto correspondiente, que estará configurado en modo “no expire”. Gracias a esto, cada versión de código siempre tendrá disponible un fichero comprimido con el código fuente y diferentes ficheros generados (como firmware, ejecutables, PDF, etc…).

Se usará Ceedling en lugar de un procedimiento personalizado como el que hemos expuesto. La API creada, totalmente abstraída de la arquitectura, servirá como base para proyectos que actualmente están en curso, al igual que se perfeccionará la capa de metrología de forma que se use tanto para fines informativos como de facturación en smart meters (contadores inteligentes), y para fines de monitoreo y control para otros proyectos relacionados con smart cities e IoT.

Apéndice A
Configuración de la placa. Jumpers

Tabla A.1 Pinout del socket de propósito general TWRPI.

J12-Pin	Descripción	J11-Pin	Descripción
1	Vcc 5 V	1	GND
2	Vcc 3.3 V	2	GND
3	GND	3	I2C: SCL
4	VDDA 3.3 V	4	I2C: SDA
5	VSS (GND analógico)	5	GND
6	VSS (GND analógico)	6	GND
7	VSS (GND analógico)	7	GND
8	CAD: Analog 0	8	GND
9	CAD: Analog 1	9	SPI: MISO
10	VSS (GND analógico)	10	SPI: M0SI
11	VSS (GND analógico)	11	SPI: SS
12	CAD: Analog 2	12	SPI: CLK
13	VSS (GND analógico)	13	GND
14	VSS (GND analógico)	14	GND
15	GND	15	GPIO: GPIO0/IRQ
16	GND	16	GPIO: GPIO1/IRQ
17	CAD: TWRPI ID 0	17	UART: UART RX or GPIO: GPIO2
18	CAD: TWRPI ID 1	18	UART: UART TX or GPIO: GPIO3
19	GND	19	UART: UART_CTS or GPIO: GPIO4/Timer
20	RESET	20	UART: UART_RTS or GPIO: GPIO5/Timer

Tabla A.2 Jumpers de configuración.

JUMPER	Descripción
J1	Conecta VBAT a la alimentación o a MCU_PWR/VBATD0
J2	Conecta/desconecta VREF del *ELEVATOR edge*
J6	Conecta MCU a V_BRD 3.3V o a voltaje externo
J7	Conecta/desconecta tensiones analógicas a MCU_PWR
J8	Conecta/desconecta PTG1/AD10 al potenciómetro R20
J9	Conecta/desconecta PTF0/AD7 al sensor de temperatura
J19	Anula/habilita la medición de corriente de baja potencia
J20	Conecta/desconecta la entrada de reset del KM34 al K20 OpenSDA

Tabla A.3 Conexiones del Switch S1 DIP (Jumper S1).

Jumpers	Descripción
1-20	Conecta PTE5 al LED Verde (D3)
2-19	Conecta PTF1 al LED Rojo (D4)
3-18	Conecta PTD1 al LED Naranja (D5)
4-17	Conecta PTC1 al LED Amarillo (D6)
5-16	Conecta el LED IrDA Tx (D7) al PTC2
6-15	Conecta el Transistor IrDA Rx (Q1) al PTC3
7-14	Conecta el dato EEPROM serie al dato I2C0
8-13	Conecta el reloj EEPROM serie al reloj I2C0
9-12	Conecta la señal TILT_ENABLE (MMA8491Q) al PTF7
10-11	Conecta la señal de medida de tensión VBAT al PTF2/AD9

Tabla A.4 Conexiones de señales del header GPIO (J10).

J10 Pin	MCU Signal
1	SW3 (Tamper switch)
2	SW3 to TAMPER1 (when closed)
3	SW3 (Tamper switch)
4	SW3 to TAMPER2 (when closed)
5	TILT_XOUT
6	TILT_XOUT to TAMPER0
7	TILT_YOUT
8	TILT_YOUT to TAMPER1
9	TILT_ZOUT
10	TILT_ZOUT to TAMPER2
11	V_BRD
12	PTI0/SCI1_RXD
13	PTI1/SCI1_TXD
14	GND
15	PTE0/I2C0_SDA
16	PTD7/I2C0_SCL
17	PTF6/SPI1_MOSI
18	PTF5/SPI1_MISO
19	PTF4/SPI1_SCK
20	PTF3/SPI1_SS
21	PTA6/AFE_EXT_MOD_IN
22	PTF7/CLKOUT
23	PTC0/AFE_EXT_MOD_IN2
24	PTG0
25	PTB7/AFE_EXT_CLK_IN
26	PTC5/AD0
27	PTA7/AFE_EXT_CLK_OUT
28	PTC6/AD1
29	VSSA_SAR
30	GND

Tabla A.5 Entradas analógicas/Salida generada (J17).

J17 Pin	MCU Signal
1	GEN_OUT0
2	EXT_SD_ADP0
3	GEN_OUT1
4	EXT_SD_ADP1
5	GEN_OUT2
6	EXT_SD_ADP2
7	GEN_OUT3
8	EXT_SD_ADP3
9	GEN_OUT4
10	EXT_SAR_AD0
11	GEN_OUT5
12	EXT_SAR_AD1
13	GEN_OUT6
14	EXT_SAR_AD2
15	VSSA
16	EXT_SD_ADM0(can short to VSSA)
17	VSSA
18	EXT_SD_ADM1
19	VSSA
20	EXT_SD_ADM2
21	VSSA
22	EXT_SD_ADM3

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] Analog Device Inc., *ADE7880 Datasheet and Product Info | Analog Devices*.

[2] Pete Batard, *Rufus - Cree unidades USB arrancables fácilmente*.

[3] Xabier Crespo, *Tests unitarios en sistemas embebidos: ¿realidad o ficción?*, may 2018.

[4] Antonio Félix Sánchez-Oro Portillo, *Agile Testing. Estado del arte. Su aplicación en empresas TIC de Extremadura.*, Ph.D. thesis, Universidad de Extremadura, jul 2017.

[5] GitLab Company, *The first single application for the entire DevOps lifecycle - GitLab | GitLab*.

[6] Alexander Herrmann and Richard Fix, *Extreme programming without fear - Embedded.com*, feb 2004.

[7] Martin Mienkina, *Filter-Based Algorithm for Metering Applications*, nov 2019.

[8] NXP Company, *FreeMASTER for Embedded Applications*, 2019.

[9] NXP Semiconductors, *Design Studio Integrated Development Environment (IDE) | NXP*.

[10] PlayOnLinux Company, *Home - PlayOnLinux - Run your Windows applications on Linux easily!*

[11] Sabrina, *Git vs SVN: ¿Cuál es mejor? Ventajas y Desventajas - Guiadev*, apr 2019.

[12] STMicroelectronics, *STPM34 - ASSP for metering applications with up to four independent 24-bit 2nd order sigma-delta ADCs, 4 MHz OSF and 2 embedded PGLNA - STMicroelectronics*, 2020.

[13] Sublime Text Company, *Sublime Text - A sophisticated text editor for code, markup and prose*.

[14] The Debian Installer Team, *1.1. What is Debian?*, 2015.

[15] Mark VanderVoord, Mike Karlesky, and Greg Williams, *Ceedling — Throw The Switch*, 2015.

Glosario

AC Alternating Current. 21

ADC Analog to Digital Converter. 5, 21, 22, 24, 25, 29, 30

AES Advanced Encryption Standard. 5

AFE Analog Front End. 5, 24, 25, 27, 30

API Application Programming Interface. 6, 53, 55

CI/CD Continuous Integration/Continuous Deployment. 1, 2, 33, 49, 50, 53

DAC Digital to Analog Converter. 25

DMA Direct Memory Access. 26

FLL Frequency-Locked Loop. 25, 27

GPIO General-Purpose Input/Output. 21, 28, 30, 31, 58

GUI Graphic User Interface. 29

I2C Inter-Integrated Circuit. 26, 28, 30, 31

IDE Integrated Development Environment. 1, 2

IoT Internet Of Things. 1

IrDA Infrared Data Association. 21, 26, 28, 32, 58

LCD Liquid Crystal Display. 21, 26

MMAU Metering specific Memory Mapped Arithmetic Unit. 55

NTC Negative Temperature Coefficient. 21

PLL Phase-Locked Loop. 25, 27

RTC Real-Time Clock. 21, 27

SCI Serial Communications Interface. 30

SPI Serial Peripheral Interface. 26, 30, 31

TDD Test-Driven Development. 1, 37, 38, 44, 53

TFS Thin Film Storage. 24

TWRPI Tower Plug-In. 21

UART Universal Asynchronous Receiver/Transmitter. 26, 40