**UNIVERSIDAD DE CASTILLA-LA MANCHA**
**ESCUELA SUPERIOR DE INFORMÁTICA**

**GRADO EN INGENIERÍA INFORMÁTICA**

**TRABAJO FIN DE GRADO**

# Sistema de Logging Geoposicional en Tiempo Real para Sistemas Empotrados

**Daniel Aguado Araujo**

Julio, 2014

# Sistema de Logging Geoposicional en Tiempo Real para Sistemas Empotrados


**UNIVERSIDAD DE CASTILLA-LA MANCHA**

**ESCUELA SUPERIOR DE INFORMÁTICA**

**Tecnologías y Sistemas de Información**


**TECNOLOGÍA ESPECÍFICA DE**
**INGENIERÍA DE COMPUTADORES**

**TRABAJO FIN DE GRADO**

# Sistema de Logging Geoposicional en Tiempo Real para Sistemas Empotrados


Autor: Daniel Aguado Araujo

Director: Fernando Rincón Calle

Director: Alberto Gómez de Dios


Julio, 2014

**Daniel Aguado Araujo**

Ciudad Real – Spain

*E-mail:* Daniel.Aguado@alu.uclm.es
*Teléfono:* 629 44 82 49

## TRIBUNAL:

**Presidente:**

**Vocal:**

**Secretario:**

## FECHA DE DEFENSA:

## CALIFICACIÓN:

| PRESIDENTE | VOCAL | SECRETARIO |
|---|---|---|
| Fdo.: | Fdo.: | Fdo.: |

# Resumen

Este Trabajo Fin de Grado (TFG) se ha realizado dentro del proyecto «Biker Assistant (BA)», un firmware para el control de motocicletas eléctricas, desarrollado por la empresa SmartSoC Solutions. Dicha empresa ha impuesto unas restricciones de diseño y calidad propias, necesarias para una correcta y sencilla integración de este TFG dentro del proyecto que está desarrollando. Esta imposición ha influido de manera notoria en la toma de decisiones sobre el desarrollo e implementación del sistema.

En este TFG se implementa un sistema de logging geoposicional, que almacena los datos recibidos por un dispositivo GPS en la memoria FLASH de la placa en la que se ejecuta el sistema. Durante este documento se ofrece una visión detallada de los aspectos más importantes a tener en cuenta en el desarrollo de cualquier sistema empotrado, de los retos que ofrece la programación a bajo nivel y con unas fuertes restricciones de memoria y capacidad de cómputo.

Este tipo de sistemas son muy utilizados hoy en día y existen multitud de dispositivos que lo implementan, como puede ser cualquier *smartphone*. La complejidad del desarrollo de los sistemas empotrados viene dada por las restricciones de hardware y software impuestas por la empresa interesada, lo cual ha llevado a tener que tomar varias decisiones de diseño con el fin de adaptar la funcionalidad requerida de la aplicación a los recursos disponibles.

Entre las restricciones del hardware impuesto por la empresa se encuentra se encuentra la utilización de la placa «CPU ARM Cortex M3 STMF205VGT6» y el dispositivo GPS «GlobalTop FGPMMOPA6H».

El presente trabajo se puede dividir en dos objetivos bien diferenciados: 1) realizar un sistema de comunicación entre la placa del sistema con el componente GPS para extraer los datos que nos proporcione, y 2) la implementación de un sistema de archivos que nos de soporte para el almacenamiento y posterior lectura de los datos extraídos del GPS. Estos objetivos se dividen en varios subobjetivos y su diseño, desarrollo e implementación son los que se recogen en este documento.

Una de las decisiones más destacables del desarrollo de este TFG que se han tenido que tomar está el implementar un Sistema de Archivos de desarrollo propio, ya que, como se verá en el apartado 5.6, el hardware presenta unas restricciones que impiden la utilización de un Sistema de Archivos comercial.

# Abstract

This TFG has been developed within the "Biker Assistant (BA)" project, a firmware to control electrical motorcycles, implemented by the SmartSoC Solutions company. This company has set its own design and quality restrictions which are needed to achieve a proper and simple integration of this TFG into the project named above. This restriction have affected notoriously when taking the decisions related to the system development and design.

A geopositional logging system has been developed throughout the current TFG. This system receives data from a GPS device and stores it on an embedded FLASH memory. Along this document a detailed vision of the most important points about development embedded systems, low level programming and strong restrictions of memory and computing power is explained.

Embedded systems are very used nowadays and there are a lot of devices that implement them, such as any smartphone. Development complexity of this kind of systems depends on hardware and software restrictions imposed by the company. Because of this, several design decisions have been taken in order to fit the required system functionality to the available resources.

Some of the imposed hardware specifications given by the company are the use of the "CPU ARM Cortex M3 STMF205VGT6" board and the "GlobalTop FGPMMOPA6H" GPS device.

The current work can be divided into two distinct goals: 1) to develop a communication system between the system board and the GPS component in order to read the data from this GPS, and 2) to implement a file system to support the data storage and to read the data extracted from the GPS. This goals are divided into several subtasks and their design, development and implementation are included in this document.

One of the most highlighted decisions about the development of this TFG that has been taken is the implementation of an own file system. As is described in the 5.6 section, hardware has constraints that prevent the use of commercial file systems.

# Agradecimientos

A mi familia por apoyarme en todo lo que han podido.

A mis directores de proyecto Alberto y Fernando, por darme la oportunidad de trabajar en este proyecto y realizar un trabajo real y diferente al que estaba acostumbrado.

Daniel

# Índice general

# Índice de tablas

# Índice de figuras

# Índice de listados

# Listado de acrónimos

| | |
|---|---|
| **BA** | Biker Assistant |
| **GNU** | GNU is Not Unix |
| **GPL** | GNU General Public License |
| **MIT** | Instituto Tecnológico de Massachusetts |
| **NMEA** | Asociación Nacional de Electrónica Marina de Estado Unidos |
| **NVIC** | Controlador del Vector de Interrupciones Apiladas |
| **SER** | Especifique, Explore-y-Refine |
| **TFG** | Trabajo Fin de Grado |
| **USART** | Receptor/Transmisor Universal Asíncrono |

Capítulo 1

# Introducción

En este TFG se desarrolla parte del firmware de un Sistema Empotrado, utilizado para la gestión y control de una motocicleta eléctrica, consistente en el control de un dispositivo GPS y el almacenamiento en un log de los datos recibidos. El componente desarrollado en este documento es parte de un sistema mayor construido por la empresa SmartSoC Solutions.

En la actualidad los sistemas empotrados juegan un rol muy importante en muy diversos productos como aplicaciones y servicios en coches, teléfonos móviles, sistemas avanzados de manufactura, equipos médicos, infraestructuras públicas, etc. Los sistemas empotrados suponen un importante aspecto de diferenciación de una empresa dando mejor servicio y experiencia de usuario a sus clientes, representando, en algunos casos, la inversión y desarrollo de este tipo de sistemas hasta el 50 % de su coste operacional [HG13].

Los avances en sistemas empotrados no están limitados únicamente al sector industrial ya que la mayoría de los hitos alcanzados redundan en una mejor calidad de vida de los usuarios o incrementa la productividad y eficiencia de los servicios que realizan. Los sectores que más han implantado sistemas empotrados en su trabajo diario son los relacionados con la sanidad, transporte y logística, seguridad y energía y medio ambiente [HG13].

El tamaño del mercado de los sistemas empotrados es algo difícil de medir ya que son parte de muy diversos productos tecnológicos. Se estima que el volumen de negocio en Holanda en 2014 es de 73 Millones de € al año y con unas previsiones de crecimiento bastante altas [HG13].

## 1.1 Marco de realización del TFG

Este TFG se realiza en colaboración con la empresa SmartSoC Solutions y se encuentra enmarcado dentro del proyecto «BA-2013 Biker Assistant» de la citada empresa.

El proyecto BA consiste en la creación de un sistema electrónico completo que gestione y

controle los sistemas y comunicaciones de una motocicleta eléctrica.

En el desarrollo del proyecto intervienen los siguientes actores:

- LGN Tech Design, como líder del proyecto.
- IS2 del grupo POAS, como coordinador y responsable de electrónica.
- Electrónica Viesca, como responsable del sistema de gestión del motor eléctrico.
- SmartSoC Solutions, como responsable del firmware del sistema «Biker Assistant».
- Grupo ARCO de la UCLM, como desarrollador de firmware.

El firmware del sistema BA ha sido diseñado por SmartSoc Solutions dentro de su ámbito de responsabilidad en el proyecto, siguiendo las especificaciones técnicas y funcionales indicadas por el líder del proyecto.

Una vista general del sistema BA aparece en la figura 1.1.

El sistema planteado en este TFG es el módulo de LOG junto con los sistemas del Sistema de Archivos y las comunicaciones con el dispositivo GPS, como se describe con mayor detalle en el apartado 1.3.

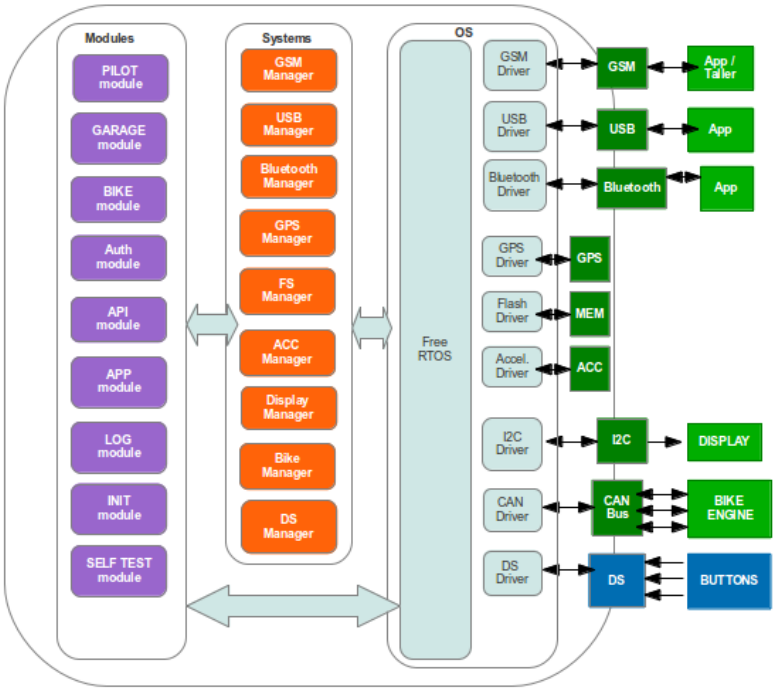



Figura 1.1: Visión global del sistema «Biker Assistant» completo

## 1.2 Motivación del presente TFG

Dentro del sistema BA se debe desarrollar un subsistema de acceso a geolocalización y logging de la misma en tiempo real, que constituye el objetivo de este TFG.

Este sistema debe ser capaz de recoger información del dispositivo GPS integrado en la placa proporcionada y almacenarlo en su memoria interna.

En este TFG se explica cómo ha sido diseñado y desarrollado este subsistema, teniendo en cuenta las restricciones de los sistemas empotrados, así como las características de tiempo real necesarias y las específicas del hardware disponible.

## 1.3 Subsistema de Logging Geoposicional en Tiempo Real

El subsistema de Logging Geoposicional en Tiempo Real consta de los siguientes elementos:

- Acceso al hardware: módulos GPS y memoria Flash.
- Integración con el sistema operativo en tiempo real.
- Gestor de alto nivel para el uso del GPS.
- Sistema de ficheros para la memoria Flash y gestor del mismo.
- Módulo encargado de realizar el logging del geoposicionamiento.

En la figura 1.2 se puede ver un diagrama general del subsistema.

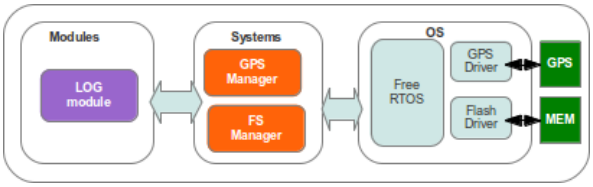



Figura 1.2: Visión global del sistema objetivo de este TFG

## 1.4 Retos a afrontar

Durante la realización del sistema descrito en este TFG se deben afrontar varios retos importantes que se describen a continuación.

El primer reto afrontado ha sido la configuración del entorno de desarrollo, ya que no existe un entorno oficial y se han tenido que utilizar varias alternativas libres, que cumplen bastante bien su cometido pero que presentan algunos problemas de compatibilidad en determinadas versiones.

Otro de los grandes retos afrontados es la utilización de un prototipo hardware diferente al del sistema final, ya que no cuenta con ninguno de los sistemas de interacción con el usuario y la utilización de un un hardware específico que cuenta con sus propias restricciones, como ya se verá más adelante.

El reto de desarrollar un sistema en un prototipo hardware que no dispone de ningún componente con el que pueda interactuar el usuario complica notablemente el desarrollo, ya que el sistema se puede probar únicamente a través de la interfaz de programación y depuración.

La inestabilidad de los prototipos hardware es otro de los problemas que afrontar. Durante la realización de este TFG se produjo un error en uno de los prototipos que provocó la imposibilidad de volver a utilizarlo y se tuvo que remitir a la empresa desarrolladora para que lo revisase.

Además, tener un hardware impuesto por la empresa obliga a un estudio más profundo del sistema y la toma de decisiones específicas para completar las funcionalidades requeridas sobre las restricciones inherentes al sistema. Estos problemas se abordan durante el desarrollo del sistema en el Capítulo 5 fundamentalmente.

## 1.5 Estructura del documento

En el Capítulo 2 se abordan los objetivos del presente TFG. Se detalla la división en varias tareas la realización de las funcionalidades planteadas.

En el Capítulo 3 se realiza una visión general sobre los Sistemas Empotrados, los procesos de desarrollo, su historia, el estado actual y su composición. También se habla de la importancia de los Sistemas Operativos en los Sistemas Empotrados y sobre diferentes Sistemas de Archivos.

En el Capítulo 5 se realiza un seguimiento del proceso de desarrollo llevado a cabo, junto con explicaciones detalladas sobre los retos afrontados y las soluciones planteadas.

En el Capítulo 6 se comparan los resultados obtenidos con los requisitos, tanto funcionales como no funcionales, del sistema propuesto.

En el Capítulo 7 se detallan las conclusiones a las que se han llegado después de realizar este TFG.

En el Capítulo 8 se hace un pequeño resumen sobre los mayores problemas afrontados y posibles líneas de trabajo futuro.

En el Anexo A se encuentran varias imágenes sobre el hardware utilizado. Además se puede ver sobre una imagen de la placa la localización de sus componentes más característicos.

En el Anexo B se encuentra una estimación de los costes de desarrollo, además del coste final una vez completado el sistema.

En el Anexo C se encuentra una pequeña descripción del formato de almacenamiento de los mensajes del GPS en los archivos de log. En este apartado se explica también el motivo por el que se almacena con dicho formato.

En el Anexo D se encuentra una completa documentación sobre el sistema desarrollado. Esta documentación ha sido generada por la herramienta Doxygen después de utilizar un formato específico en los comentarios del código fuente.

Capítulo 2

# Objetivos

El objetivo principal de este TFG es desarrollar un sistema de logging geoposicional que cumpla con las restricciones del proyecto BA para, una vez finalizado, integrarlo dentro del mismo sistema.

Debido al hecho de que este TFG forma parte de un proyecto mayor con el que debe ser totalmente compatible y fácilmente integrable se deben cumplir una serie de restricciones ya fijadas en el proyecto además de las restricciones propias que tendría el desarrollo del sistema de forma independiente.

Las restricciones básicas de recursos y componentes a utilizar por el proyecto BA y que se siguen en este TFG son las siguientes:

1. Hardware
   *a*) «CPU ARM Cortex M3 STMF205VGT6», con 1MByte Flash Memory.
   *b*) «GPS GlobalTop FGPMMOPA6H», acceso comandos AT.

2. Software
   *a*) Compilador cruzado y librerías de programación en C para el Cortex M3.
   *b*) Uso del sistema operativo en tiempo real FreeRTOS.

Las restricciones hardware y software a utilizar condicionan notablemente las decisiones de diseño a tomar, como se especifica en el capítulo 5 de este documento.

Respecto al sistema, éste que debe cumplir las siguientes premisas de calidad:

- Extensible: el sistema debe poder extenderse con nuevas funcionalidades para los distintos actores de forma sencilla.
- Robusto: el sistema debe ser capaz de poder estar en ejecución de forma indefinida sin presentar fallos por el mero hecho de estar activo.

- Flexible: el sistema debe permitir cambios en su estructura e implementación de forma fácil y sin un excesivo coste en tiempo y recursos.
- Recuperación: el sistema debe ser capaz de recuperarse por sí mismo frente a fallos eléctricos que no afecten al hardware (fallos suministro eléctrico).
- Adaptabilidad: el sistema debe poder instalarse en otras plataformas de hardware con el mínimo coste posible.

A raíz de los condicionantes enumerados anteriormente surge una nueva batería de restricciones:

- Tamaño en memoria RAM: todo el sistema debe tener una huella pequeña, ya que debe poder ejecutarse sobre un máximo de 128 KB de memoria.
- Tamaño en memoria FLASH: todo el sistema debe ocupar el mínimo espacio posible ya que dentro de la misma memoria FLASH se debe almacenar el ejecutable del programa y el sistema de archivos donde almacenar el log del GPS y disponiendo de un máximo de 1 MB de memoria.
- Garantía de ejecución de las operaciones: todas las operaciones deben tener garantizada su ejecución.
- Tiempos: se deben respetar los tiempos máximos de respuesta para cada una de las operaciones que pueden realizar los actores simultáneamente.

La complejidad del desarrollo de este TFG se ve incrementada notablemente debido a las restricciones conocidas, restricciones propias del trabajo con el hardware a bajo nivel y la necesidad de probar las soluciones en una plataforma física independiente de la máquina donde se desarrolla el sistema. Además esta plataforma es ligeramente diferente de la máquina donde se implementará el sistema completo una vez desarrollado ya que la única interfaz con el usuario son tres leds, aunque el desarrollador también puede utilizar la interfaz de programación y depuración.

Así pues el objetivo principal de este proyecto consistirá en el diseño y desarrollo de un sistema autónomo para el acceso a datos de geolocalización y logging de los mismos en un fichero de texto sobre la plataforma proporcionada por la empresa.

## 2.1 Objetivos específicos

Dado el alcance del proyecto y la complejidad añadida por el hardware y el sistema operativo en tiempo real a utilizar, se desglosa el objetivo principal en varios subobjetivos que

nos proporcionarán una hoja de ruta para el desarrollo del sistema.

La lista de subobjetivos, junto con su descripción, se listan a continuación y se describen más adelante:

1. Comprensión de la plataforma hardware.
2. Conocimiento software de acceso al hardware.
3. Acceso a la memoria FLASH.
4. Acceso al GPS.
5. Implementación de un Sistema de Ficheros.
6. Programación y configuración del Sistema Operativo en Tiempo Real FreeRTOS.
7. Diseño y programación del sistema final.

### 2.1.1 Comprensión de la plataforma hardware

Se debe analizar la plataforma hardware elegida de modo que se conozcan los distintos componentes que la forma y la interacción entre los mismos.

En resumen se debe estudiar el comportamiento de la placa «CPU ARM Cortex M3 STMF205VGT6», prestando especial atención a la memoria FLASH presente en la misma, y del GPS «GlobalTop FGPMMOPA6H».

Este objetivo se documenta en profundidad en el apartado 5.2.

### 2.1.2 Conocimiento software de acceso al hardware

Este objetivo está estrechamente relacionado con el anterior. Se debe estudiar la programación y bibliotecas de acceso al hardware proporcionado por el fabricante de modo que se esté en disposición de acometer el diseño y desarrollo del sistema.

En resumen se deben estudiar las bibliotecas proporcionadas por el fabricante de la placa «CPU ARM Cortex M3 STMF205VGT6», especialmente la biblioteca general, la específica de acceso a la FLASH y las necesarias para comunicarse con el componente GPS, y las proporcionadas por el fabricante del GPS.

Este objetivo se documenta en profundidad en el apartado 5.3.

### 2.1.3 Análisis del acceso a la memoria FLASH

Analizar y programar varios test de acceso a la memoria FLASH de modo que se gane el conocimiento necesario para su uso y se descarte un mal funcionamiento de la misma.

Este objetivo se documenta en profundidad en el apartado 5.4.

### 2.1.4 Análisis del acceso al GPS

Analizar y programar varios test de acceso al módulo GPS de modo que se gane el conocimiento necesario para su uso y se descarte un mal funcionamiento del mismo.

Este objetivo se documenta en profundidad en el apartado 5.5.

### 2.1.5 Implementación de un Sistema de Ficheros

Estudiar posibles opciones y se realizará el desarrollo del sistema de ficheros a utilizar con la memoria FLASH con el objetivo de gestionar los ficheros de log generados por la recepción de datos del GPS.

Este objetivo se documenta en profundidad en el apartado 5.6.

### 2.1.6 Programación y configuración del Sistema Operativo en Tiempo Real FreeRTOS

Analizar, configurar y programar varios test de prueba del Sistema Operativo FreeRTOS de modo que se gane el conocimiento necesario para su uso.

Este objetivo se documenta en profundidad en el apartado 5.7.

### 2.1.7 Diseño y programación del sistema final

Una vez comprendido el funcionamiento y teniendo implementaciones funcionales de acceso a los componentes necesarios, se retoman las restricciones y necesidades iniciales del sistema BA para implementar el sistema de logging geoposicional objeto de este TFG.

Este objetivo se documenta en profundidad en el apartado 5.8.

Capítulo 3

# Antecedentes

En este capítulo se hará un breve estudio del desarrollo de los sistemas empotrados, al estado actual de los mismos, el grado de penetración y uso en el mundo real, los diferentes periféricos que tiene integrado el sistema empotrado sobre el que se va a realizar este TFG y el estado actual de los sistemas operativos para sistemas empotrados que existen hoy en día. También se hablará sobre los diferentes sistemas de archivos que existen hoy día, alternativas e historia.

## 3.1 Sistemas empotrados

Un sistema empotrado, también llamado «sistema embebido» debido a una mala traducción del anglicismo «*embedded system*», es un sistema computacional diseñado para realizar una o varias tareas muy específicas, generalmente en tiempo real. Al contrario de lo que ocurre con sistemas computacionales de propósito general, que están diseñados para realizar un amplio rango de tareas diferentes, un sistema empotrado está diseñado para cubrir necesidades muy concretas. En un sistema empotrado la mayoría de los componentes están contenidos dentro de la misma placa de silicio y su diseño difiere sustancialmente respecto a lo que solemos asociar con un computador.

Un sistema empotrado no está compuesto solo por hardware ya que necesita un software de control capaz de realizar las tareas para las que se diseña. Generalmente este software de control se desarrolla a la vez que el sistema hardware para poder ajustarse a los requisitos, aunque en los últimos tiempos este paradigma está cambiando al aparecer en el mercado placas que pueden ser reprogramadas y ser orientadas a multitud de diferentes tareas.

Los sistemas de este tipo, al estar pensados para hacer tareas concretas, suelen tener unas restricciones de memoria y capacidad de cómputo ajustadas a las tareas que se pretende que realicen para así controlar y reducir los costes de fabricación.

Normalmente estos sistemas son reactivos y funcionan en tiempo real, entendiendo por

reactivo un sistema que interactúa con el entorno cuando se produce un evento predeterminado y funcionar en tiempo real que el comienzo de la acción a realizar sucede en un período de tiempo acotado desde que sucedió el evento que la produjo [Wij14].

Los componentes fundamentales de un sistema empotrado clásico son una memoria para almacenar el programa de control, un procesador para poder ejecutarlo y buses de comunicación entre los componentes sobre los que se pueden conectar otros componentes externos como pueden ser sensores, actuadores u otros sistemas empotrados [Wij14].

## 3.2 Desarrollo de sistemas empotrados

El desarrollo de los Sistemas Empotrados difiere sustancialmente del desarrollo del software convencional, no sólo por las restricciones de memoria y capacidad de cómputo de estos sistemas sino por la estructura física de este tipo de componentes que puede hacer inviable la utilización de componentes que funcionarían a la perfección en sistemas de propósito general.

El desarrollo de un sistema empotrado empieza por la definición de las tareas que debe realizar seguido de un estudio pormenorizado de las necesidades computacionales y de memoria que se necesitan, así como los periféricos, sensores y actuadores que se tienen que controlar y monitorizar para realizar las tareas propuestas.

El proceso de desarrollo de un sistema empotrado se suele realizar *top-down*, comenzando con la especificación del sistema completo y en iteraciones posteriores ir bajando el nivel de abstracción hasta tener el sistema completo bien definido, pasando, en este orden, por la especificación a nivel de sistema, a nivel de computación, a nivel lógico y por último a nivel de circuito.

Existen otras metodologías más clásicas como puede ser *bottom-up*, en la que se empieza a desarrollar desde un nivel de abstracción menor hacia uno mayor, o *meet in the middle* donde se combinan las metodologías *top-down* y *bottom-up* intentando suplir las carencias de cada una y sumando sus ventajas.

Si a estas metodologías le añadimos un componente de pruebas tenemos la que más se utiliza en el desarrollo de este tipo de sistemas y a la que se le suele llamar Especifique, Explore-y-Refine (SER) [GAGS10].

El flujo de trabajo de la metodología SER se especifica a continuación:

- Especificación ejecutable a nivel de sistema, representando el comportamiento del mismo.
- Incluir varios modelos con diferentes detalles que corresponden a diferentes decisiones de diseño (propiedades del sistema tales como velocidad, consumo de potencia, etc).
- Síntesis de las diferentes opciones para explorar el espacio de diseño.
- Refinamiento del modelo de acuerdo a las decisiones tomadas.

En la figura 3.1 podemos ver las tareas que se necesitan completar para realizar el diseño de un sistema empotrado completo. En el caso del sistema empotrado objetivo de este TFG la rama de desarrollo del hardware no es necesario realizarla ya que se utiliza una placa ya existente.

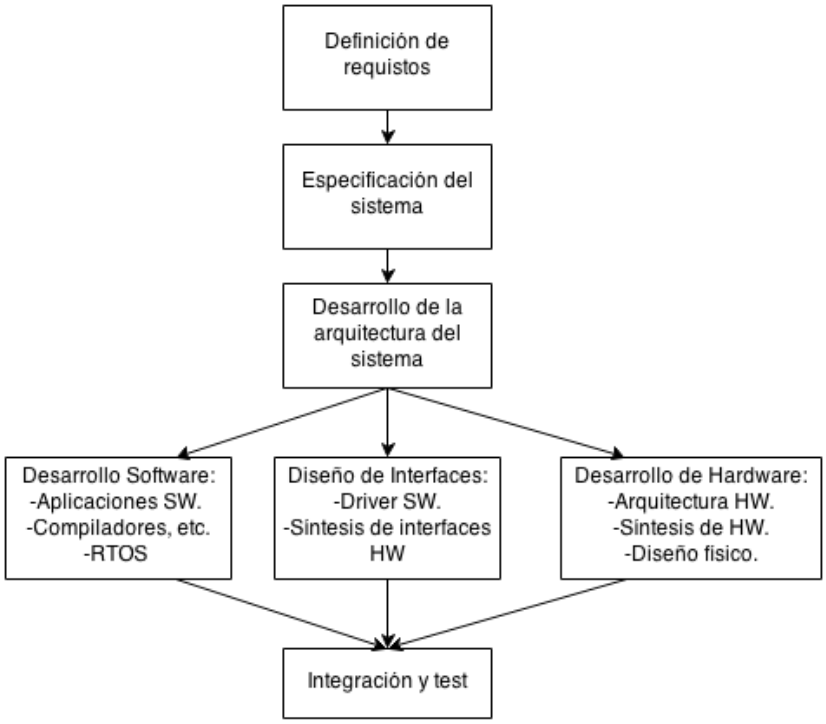



Figura 3.1: Diagrama de flujo del desarrollo de un sistema empotrado

Otra metodología de desarrollo, y más adecuada para la realización de este TFG es el llamado «Desarrollo de Sistemas Basado en Placa», donde el desarrollo del hardware es totalmente independiente del desarrollo del software y pueden estar realizados por diferentes equipos de trabajo [GAGS10]. Esta metodología tiene la ventaja que la placa hardware obtenida puede ser utilizada para diversas funcionalidades, y el inconveniente principal es que se pueden encontrar restricciones que obliguen a cambiar el diseño original del sistema.

Utilizando esta metodología, el equipo de desarrollo del componente software puede utilizar cualquier metodología clásica de desarrollo de software, aunque no podrá aplicarla completamente debido a las características propias del desarrollo de este tipo de sistemas. En el Capítulo 4 se detalla la metodología utilizada para el desarrollo de este TFG, los inconvenientes que han surgido y cómo se han solucionado.

## 3.3 Historia e importancia de los sistemas empotrados

Los sistemas empotrados conviven con nosotros desde los '60 cuando el Instituto Tecnológico de Massachusetts (MIT) desarrolló el primer sistema empotrado para controlar el sistema de guía del proyecto Apolo [MIT09]. Desde entonces han servido para automatizar los más diversos procesos y ofreciendo herramientas de las que no se disponía hasta entonces. Los sistemas empotrados se encuentran en todos los dispositivos electrónicos que nos rodean, desde juguetes, teléfonos móviles y televisiones, hasta los sistemas de control de una fábrica, de un vehículo de transporte (trenes, aviones, coches, barcos, ..) o de los semáforos que controlan el tráfico.

Los sistemas empotrados de hoy día pueden realizar muchas más tareas que los primeros que se desarrollaron ya que la evolución de la tecnología ha permitido no solo el abaratamiento de los componentes sino que también ha permitido aumentar enormemente su rendimiento llegando al punto que algunos sistemas empotrados, como los llamados *smartphones*, igualan o superan a sistemas de propósito general realizando tareas similares [Wir10].

Como hemos visto anteriormente, este tipo de sistemas están diseñados para realizar solo unas cuantas tareas específicas cada uno pero se pueden utilizar en multitud de áreas y campos.

La utilización de los sistemas empotrados en productos electrónicos suele solventar problemas de complejidad y de tamaño del producto. Por ejemplo un automóvil actual contiene docenas de sistemas empotrados para controlar diferentes aspectos del mismo, como el aire acondicionado, la inyección de combustible o la presión de las ruedas. Si los automóviles no tuvieran estos sistemas empotrados la lógica de control sería mucho más compleja y propensa a fallos. Otra alternativa sería tener un sistema computacional de propósito general, pero estaría sobredimensionado y aumentaría notablemente los costes de producción [Wij14].

## 3.4 Componentes de un sistema empotrado

Todos los sistemas empotrados tienen una serie de componentes comunes entre sí:

- Software de control.
- Microprocesador para poder ejecutar el software.
- Memoria volátil (RAM) para la memoria del programa durante la ejecución.
- Memoria no volátil (generalmente FLASH) para almacenar datos persistentes y/o el binario ejecutable del software de control.
- Subsistema de entrada/salida, utilizado para comunicarse con componentes externos como actuadores y sensores.
    - Sensores: estos componentes únicamente miden magnitudes físicas. Ejemplos de este tipo pueden ser un GPS o un termómetro.
    - Actuadores: estos componentes son capaces de relacionarse con el mundo físico y realizar cambios sobre él. En este tipo de componentes se encuentran todos los componentes que interactúan con el usuario, como pueden ser un dispositivo Bluetooth o un *display*, o con el medio físico, como puede ser un motor, una alarma o una luz.
    - Existen otro tipo de componentes que se pueden conectar a un sistema empotrado pero que no se pueden catalogar ni como actuadores ni como sensores. Entre este tipo de componentes podemos encontrar, por ejemplo, las memorias externas u otros sistemas empotrados.

En la Figura 3.2 podemos ver cómo se relacionan entre sí los diferentes componentes de un sistema empotrado.

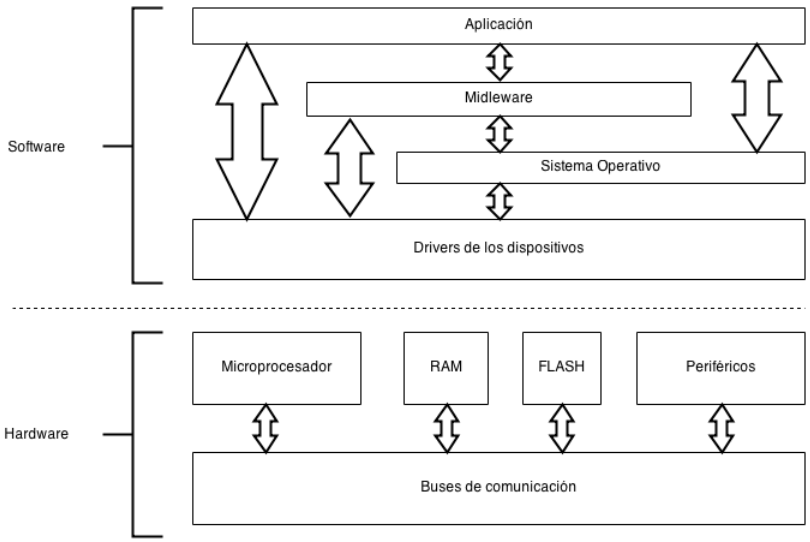



Figura 3.2: Diagrama de interacción entre componentes de un Sistema Empotrado

## 3.5 Sistemas Operativos para Sistemas Empotrados

Un Sistema Operativo es un programa entre cuyas responsabilidades se encuentran ser una capa de abstracción del hardware, gestionar los permisos y tiempos de acceso a los dispositivos, gestionar la ejecución de tareas que funcionan sobre él, la gestión de la memoria que puede utilizar cada tarea o el soporte de varios usuarios, entre otras muchas.

Un Sistema Operativo para un sistema empotrado debe ser mucho más liviano que para un sistema de propósito general debido fundamentalmente a las restricciones de cómputo mencionadas. Un Sistema Operativo de este tipo solo puede hacer las funciones más básicas como puede ser la gestión de la ejecución de las tareas, concretamente del cambio de contexto de ejecución de una tarea a otra.

Muchos sistemas empotrados que no tienen fuertes restricciones de memoria y cómputo utilizan sistemas operativos completos como pueden ser Linux [Lin14b], Android [And14] o iOS [App14].

En caso que el sistema empotrado objetivo ejecute únicamente una tarea entonces el Sistema Operativo puede no existir, disminuyendo así los requisitos computacionales necesarios para su correcto funcionamiento. Si el sistema tiene que realizar muy pocas tareas y se considera que no es necesario un Sistema Operativo, entonces necesitamos diseñar y programar un sistema que cambie el contexto de ejecución correctamente, como puede ser un sistema de *polling* que cambie de contexto de manera automática y periódica con un tiempo preestablecido.

Si el sistema empotrado objetivo ejecuta un número considerable de tareas o necesita ejecutarlas en tiempo real entonces necesitaremos un sistema operativo acorde las necesidades impuestas para facilitar la tarea de desarrollo. Uno de los sistemas operativos para sistemas empotrados más utilizados es FreeRTOS [Fre14], un sistema operativo en tiempo real liberado con una licencia GNU General Public License (GPL), pero hay muchos otros como pueden ser TinyOs [Tin14], eCos [ecO14] o chibiOS [Chi14].

En general, los Sistemas Operativos para sistemas empotrados tienen una estructura y funcionamiento común. El desarrollador del sistema debe separar las funcionalidades requeridas en diferentes tareas y el sistema se encargará de gestionar el uso de los recursos requeridos por cada una, dependiendo de sus prioridades.

Para gestionar la comunicación entre las diferentes tareas de un Sistema Operativo suelen

ofrecer los mecanismos típicos de sincronización y comunicación, como son los semáforos, y las colas de mensajes.

## 3.6 Sistemas de Archivos

Un sistema de archivos es una abstracción del almacenamiento en memoria que permite manejarla con mucha más facilidad. Un sistema de archivos permite gestionar, organizar y buscar archivos, así como realizar operaciones de lectura y escritura sobre los mismos.

Uno de los primeros sistemas de archivos que apareció en el mercado fue FAT, introducido en 1977 por Microsoft para su sistema DOS. Este sistema de archivos se sigue utilizando ampliamente en la actualidad, sobretodo en dispositivos de almacenamiento de memoria externos como las memorias FLASH USB. FAT, como la totalidad de sistemas de archivos posteriores, divide la memoria disponible en sectores de tamaño fijo y configurable, siendo ésta la unidad mínima de trabajo tanto para leer, escribir o borrar. La forma de almacenar la información sobre los archivos que tiene FAT es guardándolo en una tabla en la que cada fila representa, en orden, los sectores en los que se divide la memoria y cada entrada especifica si está libre u ocupado y de estarlo el fichero al que se refiere y dónde encontrar el siguiente sector ocupado que corresponde al mismo archivo (Figura 3.3).

Posteriormente aparecieron nuevos sistemas de archivos que permitían operaciones sobre archivos más rápidas y más complejas, nombres más largos y con diferente codificación, inclusión de metadatos, y soporte para cantidades de memoria cada vez mayores. También evolucionaron sistemas de archivos más antiguos como FAT que dio lugar a FAT16 y posteriormente a FAT32 [Mic14a].

Actualmente Microsoft utiliza en sus productos el sistema de archivos NTFS [Mic14b], privativo y sin soporte oficial para otros sistemas operativos pero del que, gracias a la ingeniería inversa, se ha conseguido entender su funcionamiento y ofrecer un soporte no oficial.

Para los sistemas Linux se utilizan muy diversos sistemas de archivos dependiendo del propósito del computador en cuestión. Uno de los más usados es ext4 [Lin14a], introducido en 2008 y evolución del sistema ext, el primer sistema de archivos soportado por Linux e introducido en 1992, y que soporta particiones de hasta un petabyte de tamaño y archivos de hasta 16 TBytes. El funcionamiento de este sistema de archivos es mediante *extents*, una agrupación de sectores de hasta 120 MB, pudiéndose almacenar hasta 4 en el descriptor de archivo pero, en el caso de que fueran necesarios más, uno de ellos puede referirse a otra dirección de memoria donde se almacenarían otros nuevos 4 *extents*, funcionando como si

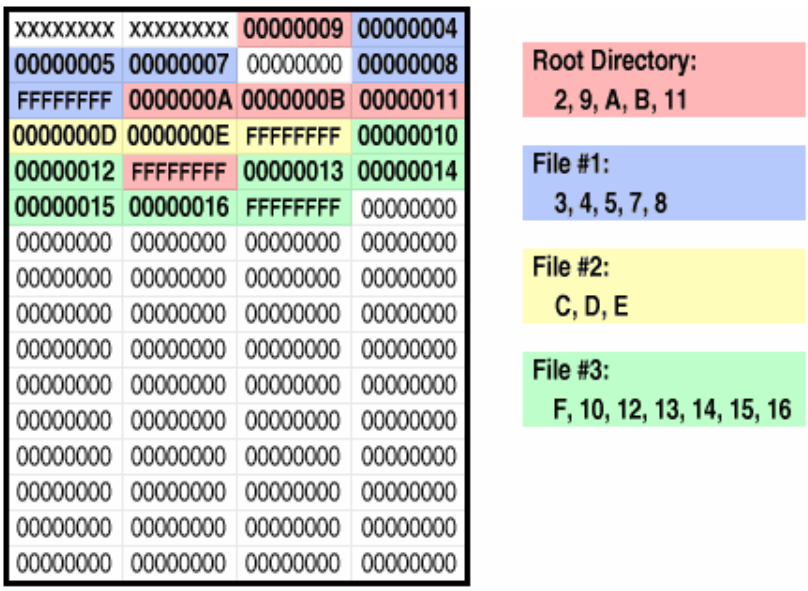



Figura 3.3: Tabla de asignación de archivos del Sistema de Archivos FAT

de un árbol se tratara (Figura 3.4).

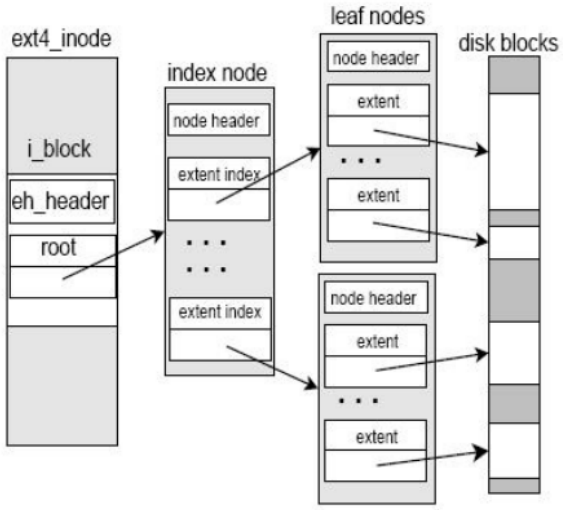



Figura 3.4: Representación gráfica del funcionamiento de HTree

Para un sistema empotrado, donde la capacidad de cómputo y la memoria disponible son muy reducidas, es posible que no convenga utilizar sistemas de archivos tan avanzados como ext4 y funcionen mejor sistemas como FAT, pero todo dependerá de las características y requisitos del sistema a desarrollar.

Hay que tener en cuenta, además, que la arquitectura de la memoria embebida en el sis-

tema puede hacer inviable la utilización de cualquiera de los estándares definidos para la implementación de un Sistema de Archivos.

Otro aspecto a tener en cuenta para el desarrollo de un Sistema de Archivos es la arquitectura física de la memoria y su tecnología de fabricación. Los sistemas de archivos, aunque su funcionamiento básico sea el mismo, deben diferenciar el tipo de memoria sobre el que trabajan para proporcionar un mayor rendimiento y fiabilidad.

Actualmente el desarrollo de los nuevos Sistemas de Archivos viene impuesto por el avance, tanto en costes como en rendimiento, de las memorias FLASH utilizadas en discos SSD, de varios TBytes. Esta tecnología, además presenta dos variantes que se deben tener en cuenta a la hora de implementar los drivers de acceso, y son la fabricación mediante NAND o mediante NOR.

Los sistemas empotrados, generalmente, llevan memoria FLASH embebida, por lo que es relevante para la realización de este TFG hacer un repaso sobre los sistemas de archivos desarrollados para esta tecnología, aunque suelen ser modificaciones sobre sistemas de archivos de propósito general.

Para memorias FLASH en general tenemos como ejemplo de sistema de archivos específico JFFS2 [JFF14], que soporta el formato de metadatos de POSIX y comprime los datos durante las escrituras.

Para memorias FLASH fabricadas con tecnología NAND tenemos YAFFS2 [YAF14], una modificación de JFFS2 diseñadas específicamente para este tipo de memorias.

También hay alternativas de sistemas de archivos de solo lectura que pueden ser útiles en determinados entornos, como pueden ser ROMFS [rom14] o SQUASHFS [squ14].

## 3.7 Dispositivos GPS

El Sistema de Posicionamiento Global (GPS) es un sistema propietario y privativo del gobierno de Estados Unidos, construido pensando en el uso militar, que es capaz de determinar la posición de un dispositivo compatible proporcionándole la información sobre la latitud, longitud y altitud a la que se encuentra [USA14].

El Sistema GPS funciona gracias a 24 satélites en órbita sobre el planeta Tierra, con tra-

yectorias sincronizadas, que envían constantemente señales hacia la Tierra. El método para determinar la posición exacta sobre la Tierra es mediante triangulación inversa de varias de estas señales. Esto exige que el sistema se encuentre al alcance de la señal de al menos tres de estos satélites, lo que implica que la mayoría de los dispositivos no funcionen de forma correcta en interiores y necesiten ayuda de una antena exterior. [USA14].

La Unión Soviética empezó a construir un sistema similar, llamado GLONASS, que no consiguió terminar hasta 2007. También pensado para uso exclusivamente militar pero liberado posteriormente para uso civil.

La Unión Europea está desarrollando otro sistema similar, llamado Galileo. Este sí será el primer sistema de geoposicionamiento motivado desde el principio para uso civil. Este sistema, además, implementa una mejora de precisión sobre los sistemas GPS y GLONASS cuando se consulta la posición en los polos.

China está desarrollando también una alternativa a los sistemas existentes, llamado BEIDOU. Este sistema difiere bastante de los anteriores ya que sus satélites se encuentran en zona geoestacionaria (los satélites del resto de sistemas se encuentran más bajos) y para el cálculo de la posición sobre la Tierra hace uno de dos de esos satélites y de una estación en tierra. La posición geoestacionaria de sus satélites garantiza un correcto funcionamiento con muy pocos satélites, pero limita su cobertura sobre la tierra a la visible por los satélites, Actualmente este sistema solo tiene una cobertura sobre toda China y algunos países circundantes.

Actualmente podemos encontrar dispositivos GPS en casi cualquier dispositivo electrónico, desde el más sencillo smartphone hasta vehículos como coches, barcos, trenes, aviones, etc., y en dispositivos específicos hechos para la lectura de datos GPS y la ayuda a la realización de rutas como pueden ser los dispositivos comercializados por la empresa TOMTOM.

Recientemente, además, están apareciendo dispositivos compatibles con varios de los sistemas de geoposicionamiento global existentes.

El estándar de transmisión de datos GPS es el NMEA 0183, como se verá más concretamente en el apartado 5.2.2.

Capítulo 4

# Método de trabajo

## 4.1 Metodología de desarrollo

En el apartado 3.2 se ha hablado de las metodologías de diseño de un sistema empotrado. Para realizar el diseño del sistema empotrado objetivo de este TFG se debe diseñar el componente software de control, o firmware.

### 4.1.1 Diseño basado en plataforma

La metodología de desarrollo empleada para el desarrollo del sistema empotrado es el llamado «Diseño basado en plataforma», donde los equipos de desarrollo del hardware y del software pueden trabajar de forma independiente y separados en el tiempo, o puede haber varios equipos de desarrollo software realizando diferentes sistemas sobre la misma placa hardware. En el artículo [TWS06] se hace un pequeño repaso a cómo la utilización de los mismos componentes hardware reduce el coste global de desarrollo de varios sistemas empotrados con diferentes funcionalidades. Esta metodología tiene también sus inconvenientes ya que, al desarrollar el software de forma independiente, se pueden topar con restricciones que impidan continuar con el desarrollo y obliguen a rediseñar el sistema.

La metodología de desarrollo basada en plataforma permite de forma sencilla y barata desarrollar una «familia de productos» con componentes, módulos o subsistemas similares pero con tareas diferentes, pero el coste de organización empresarial y del portfolio del negocio se incrementa notablemente. A finales de los '90 IBM reorganizó sus equipos de desarrollo en torno a la idea de coordinar esfuerzos y poder reutilizar componentes desarrollados en una familia de productos bien planificada. Esto condujo a la reducción de entre un 70 % y 80 % de los módulos desarrollados y un ahorro de unos 700 Millones de $ del coste de su estructura empresarial. Además de esto se incrementó el número de productos comercializados en un 270 %. En la Figura 4.1 podemos ver la evolución del ahorro de costes de IBM, el incremento del grado de reutilización y la opinión de sus trabajadores sobre el nuevo método de trabajo [TWS06].

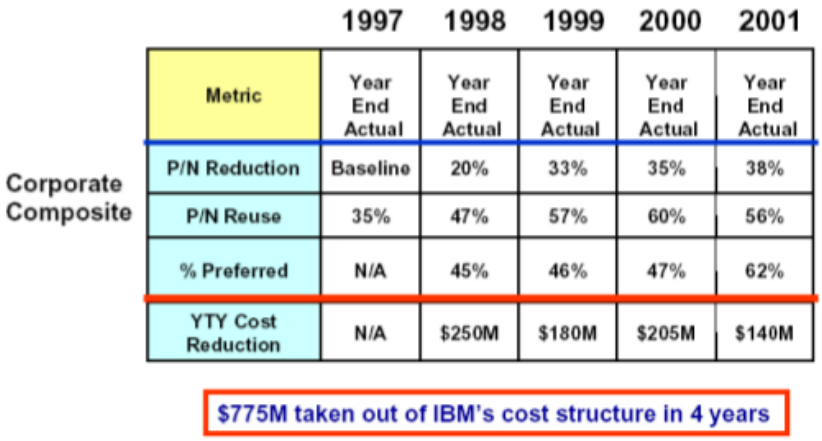



Figura 4.1: Ahorro de costes en IBM después de aplicar el desarrollo basado en plataforma

### 4.1.2 Desarrollo guiado por pruebas

Para la programación del firmware del sistema objetivo de este TFG se utilizará la metodología de desarrollo guiado por pruebas, o TDD por sus siglas en inglés (Test-Driven Development).

El TDD involucra dos técnicas de programación: escribir las pruebas primero y refactorización después [Bec02].

El proceso que se sigue es el siguiente:

- En primer lugar se escribe una prueba y se verifica que falla.
- A continuación, se implementa el código que hace que la prueba pase satisfactoriamente.
- Seguidamente se refactoriza el código escrito.

Podemos ver el flujo de trabajo en la figura 4.2.

El propósito del desarrollo guiado por pruebas es lograr un código limpio que cumpla con la funcionalidad exigida. La idea es que los requisitos sean traducidos a pruebas, de este modo, cuando las pruebas pasen se garantizará que el software cumple con los requisitos que se han establecido [Ble10].

El motivo principal para utilizar TDD es la obligación de definir antes que nada la interfaz de comunicación entre componentes y su funcionalidad, además de tener un completo y creciente set de pruebas que permite asegurar que los componentes y funcionalidades testadas

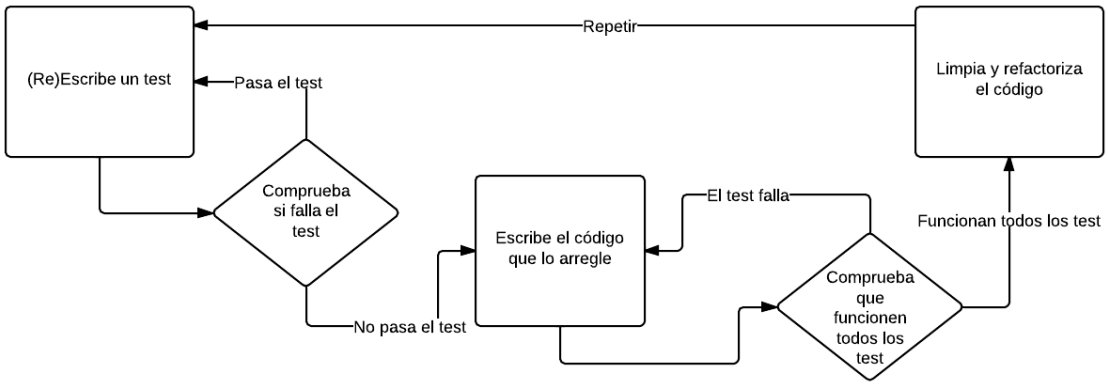



Figura 4.2: Diagrama de flujo que define el funcionamiento del proceso de desarrollo orientado a test

se realizan de forma correcta, y que siguen correctas en todo momento [Ble10].

Otro de los motivos es que al implementar única y exclusivamente el código necesario para hacer pasar los test se evita el escribir código innecesario, se reduce la redundancia de código y se eliminan los «códigos muertos» [Ble10], siendo esto último de especial importancia en sistemas con una fuerte restricción de memoria como es el sistema objetivo del proyecto.

Tener un completo set de pruebas en el desarrollo del software de un sistema empotrado es muy importante debido a la relativa inestabilidad de los prototipos utilizados, que pueden fallar total o parcialmente en un determinado momento, además de la práctica imposibilidad de aplicar parches al sistema una vez esté en producción.

La utilización del TDD en el desarrollo del sistema objetivo de este TFG ha sido difícil teniendo en cuenta que no existe ningún entorno de desarrollo o *framework* que facilite la tarea, así como tampoco existe ningún tipo de emulador de pruebas ni la capacidad del prototipo hardware de interactuar con el usuario fuera de la interfaz de programación.

La realización de las pruebas requeridas por la metodología de desarrollo se han realizado mediante funciones ejecutadas como si del sistema final se tratase. Para verificar si las pruebas han sido exitosas éstas funciones devuelven un número entero con un significado predefinido que indique el estado de éxito. Una vez las funciones han devuelto su estado de éxito la forma de comprobar cuáles son las pruebas que han tenido éxito y cuáles no s eha tenido que abrir una interfaz de depuración con *gdb* y comprobar a mano los valores obtenidos.

## 4.2 Tecnologías y herramientas utilizadas

En esta sección se comentan las herramientas hardware y software utilizadas durante la realización de este TFG.

Para la realización del sistema propuesto se ha utilizado un prototipo hardware basado en la placa «CPU ARM Cortex M3 STMF205VGT6» y con el dispositivo GPS «GlobalTop FGPMMOPA6H» empotrado en la misma. Este prototipo no tiene ningún tipo de interfaz con el exterior que ayude a la interacción con el usuario, salvo la interfaz de programación y depuración.

La interfaz de programación y depuración utilizada para realizar este sistema ha sido un dispositivo comercial que ofrece la empresa que vende la placa, en concreto el dispositivo «ST-Link/v2», que se comunica con la placa mediante comandos JTAG, un estándar de testing para placas definido en el estándar IEEE 1149.1.

En cuanto a los medios software empleados se ha utilizado el sistema operativo «Ubuntu 12.04», el entorno de desarrollo «Eclipse Kepler C/C++» con los complementos «GNU ARM toolchain», para la comunicación de Eclipse con el compilador cruzado «Sourcery CodeBench», y «Zylin Embedded CDT» para comunicar Eclipse con el entorno de programación y depuración a través del programa «OpenOCD».

Sobre las ayudas a la programación se disponen de las bibliotecas de funciones proporcionadas por el fabricante, el código fuente del sistema operativo FreeRTOS y mercurial para el control de versiones y la utilización de un repositorio en Bitbucket.

### 4.2.1 Lenguaje de programación

El lenguaje de programación con el que se realizará este TFG es C, un lenguaje de alto nivel que nos permite también un control a muy bajo nivel e incluso programar algunas funciones en ensamblador si fuera necesario. Es el lenguaje en el que los fabricantes proporcionan las bibliotecas de funciones de sus componentes hardware.

El lenguaje C apareció en el mercado por primera vez en 1972 por Dennis Ritchie en los Laboratorios Bell y, pese a su antigüedad, sigue siendo uno de los lenguajes de programación más utilizados, pese a haber sufridos escasos cambios desde su nacimiento. Según un estudio de la compañía TIOBE Software el lenguaje C se sitúa en primera posición en cuanto al número de sistemas desarrollados en 2014 con un total del 16 % del total [TIO14].

Capítulo 5

# Desarrollo del sistema

En este capítulo se describen cada uno de los objetivos definidos en el punto 2.1, así como los retos y dificultades que han surgido y las diferentes decisiones de diseño que se han tomado.

El desarrollo del sistema, en cuanto al desarrollo del software, solo abarca desde el apartado 5.4 hasta el final de este capítulo. El resto de apartados se refieren al estudio del problema.

El proceso de desarrollo del software objetivo de este TFG ha sido mediante la definición de varios test que definieran la funcionalidad primero y después el desarrollo de las funciones que hicieran cumplir los test definidos. A lo largo de este capítulo se documentan los test definidos y las funciones que implementan las funcionalidades exigidas.

## 5.1 Captura de requisitos

En esta sección se explicarán los requisitos funcionales del proyecto y algunas consideraciones a tener en cuenta.

En la Tabla 5.1 podemos ver un resumen de los requisitos pertenecientes a la gestión de la memoria.

En la Tabla 5.2 podemos ver un resumen de los requisitos pertenecientes a la gestión del dispositivo GPS.

En la Tabla 5.3 podemos ver un resumen de los requisitos pertenecientes al sistema de archivos.

En la Tabla 5.4 podemos ver un resumen de los requisitos pertenecientes al manejador del dispositivo GPS.

| Atributo | Descripción |
|---|---|
| Nombre: | FLASH Driver |
| Responsabilidades: | Abstraer del hardware del componente de almacenamiento en memoria FLASH, ofreciendo una interfaz de más alto nivel para operar con el mismo |
| Colaboradores: | FreeRTOS, FS Manager |
| Operaciones | - Init –>Inicializar el hardware<br>- Reset –>Reinicia el hardware<br>- ReadFile –>Lee un archivos desde la memoria FLASH<br>- WriteFile –>Escribe un archivo en la memoria FLASH<br>- DeleteFile –>Elimina un archivo de la memoria FLASH |

Tabla 5.1: Tabla resumen de requisitos de gestión de la memoria FLASH

| Atributo | Descripción |
|---|---|
| Nombre: | GPS Driver |
| Responsabilidades: | Abstraer del hardware del componente de geolocalización por GPS, ofreciendo una interfaz de más alto nivel para operar con el mismo |
| Colaboradores: | FreeRTOS, GPS Manager |
| Operaciones | - Init –>Inicializar el hardware<br>- ReadGPS –>lee la posición del GPS |

Tabla 5.2: Tabla resumen de requisitos de gestión del dispositivo GPS

| Atributo | Descripción |
|---|---|
| Nombre: | FS Manager |
| Responsabilidades: | Gestionar las operaciones de persistencia con el sistema de archivos en memoria FLASH del sistema |
| Colaboradores: | FreeRTOS, FLASH Driver, LOG module |
| Operaciones | - Init –>Inicializar el hardware<br>- Reset –>Reinicia el FS<br>- OpenFile –>Abre un archivo del FS<br>- CloseFile –>Cierra un archivo del FS<br>- ReadFile –>Lee N bytes de un archivo del FS<br>- WriteFile –>Escribe N bytes en un archivo del FS<br>- FeofFile –>Indica el final de un archivo del FS<br>- DeleteFile –>Elimina un archivo del FS<br>- NewFile –>Crea un archivo en el FS |

Tabla 5.3: Tabla resumen de requisitos del manejador del sistema de archivos

En la Tabla 5.5 podemos ver un resumen de los requisitos pertenecientes al módulo de gestión del LOG.

| Atributo | Descripción |
|---|---|
| Nombre: | GPS Manager |
| Responsabilidades: | Gestionar las operaciones de geolocalización del sistema |
| Colaboradores: | FreeRTOS, GPS Driver, LOG module |
| Operaciones | - Init –>Inicializar el hardware<br>- ReadGPS –>Lectura de las coordenadas GPS<br>- Send2Log –>Envío de datos al log |

Tabla 5.4: Tabla resumen de requisitos del manejador del dispositivo GPS

| Atributo | Descripción |
|---|---|
| Nombre: | LOG Module |
| Responsabilidades: | Realizar un log de las operaciones realizadas a través dela API del sistema |
| Colaboradores: | FreeRTOS, FS Manager |
| Operaciones | - Init –>Inicializar el módulo LOG<br>- Reset –>Reinicia el módulo LOG<br>- NewLog –>Crea un nuevo archivo de log<br>- DeleteLog –>Elimina un archivo de log<br>- WriteLog –>añade una entrada al archivo de log correspondiente |

Tabla 5.5: Tabla resumen de requisitos del módulo de LOG

Además de los requisitos anteriormente citados, se deben aplicar los siguientes principios de diseño:

- Estructura del sistema basada en un modelo de capas. Muy común en el desarrollo de sistemas empotrados, ya que con una capa de abstracción del hardware se permite la adaptabilidad del sistema a otras plataformas con poco esfuerzo.
- Uso de un sistema operativo robusto y confiable para sistemas empotrados. Debido a la complejidad del sistema y las capacidades de comunicación que debe ofrecer, se ha optado por el uso de un sistema operativo en lugar de una aplicación standalone, lo que mejora la robustez, flexibilidad, seguridad y capacidad de recuperación.

Se deben incluir también los requisitos no funcionales especificados en el capítulo 2: que el software sea extensible, robusto, flexible, con capacidad de recuperación ante errores y adaptable.

## 5.2 Comprensión de la plataforma hardware

El conocimiento de la plataforma hardware se centra especialmente es tres componentes: el dispositivo GPS, la memoria FLASH que tiene la placa y la placa en su conjunto para conocer cómo funciona la programación del dispositivo y la interacción con los otros dos

componentes.

En la Figura 5.1 podemos ver el diagrama de bloques de la placa utilizada.

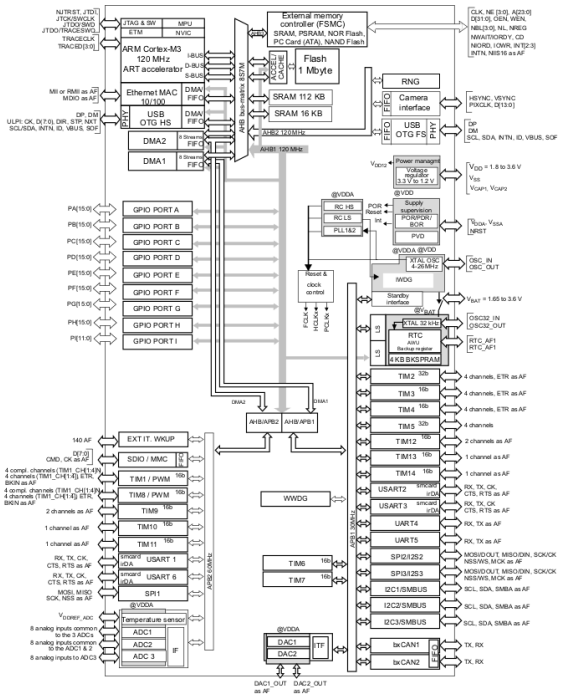



Figura 5.1: Diagrama de bloques de la placa STMF205VGT6

### 5.2.1 Memoria FLASH

En la Tabla 5.6 podemos ver las características principales de la memoria FLASH [Mic13c].

El funcionamiento de una memoria FLASH tiene sustanciales diferencias con respecto a otro tipo de tecnologías de almacenamiento. Una memoria FLASH después de realizar una operación de *Erase*, *reset* o borrado tiene todos sus bits a 1 y a partir de ahí podemos escribir cualquier dato. Una vez escrito no podemos sobreescribirlo directamente, ya que esta tecnología no es capaz de cambiar un bit de un 0 a un 1 (sí de un 1 a un 0) mediante una operación de escritura. El cambio de cualquier bit de 0 a 1 se produce con la función Erase, cuya granularidad mínima es un sector físico completo y borra toda la información del mismo.

En la Tabla 5.7 se puede ver la estructura interna de la memoria FLASH.

Para hacernos una idea de la restricción que supone tener sectores de hasta 128 KBytes, que funcionan como un ente indivisible para la función de borrado/erase, el disco duro de

| Atributo | Descripción |
|---|---|
| Nombre: | Memoria FLASH integrada |
| Definido por: | ST Microelectronics |
| Documentación: | `http://www.st.com/web/en/catalog/mmc/FM141/SC1169/SS1575/LN1433/PF245091` |
| Características: | - Capacidad de 1 MByte.<br>- Soporta lecturas de hasta 128 bits en una única operación.<br>- Soporta escrituras de media, una o dos palabras.<br>- El tamaño de palabra es de 32 bits.<br>- Las operaciones de borrado o erase se pueden realizar a nivel de sector físico o de toda la memoria al completo.<br>- Por defecto la escritura en memoria está deshabilitada.<br>- El estado de la FLASH se especifica en tiempo real en el registro FLASH_SR.<br>- La organización de los módulos de la memoria FLASH se puede ver en la Tabla 5.7.<br>- La memoria de sistema es la que contiene el boot-loader.<br>- La memoria FLASH puede funcionar con voltaje entre 1.8V y 3.6V, dividido en varios rangos y configurable.<br>- El voltaje de la FLASH puede afectar a la velocidad de la misma.<br>- Se puede configurar la latencia de la memoria FLASH y si queremos o no prefetch de instrucciones. |

Tabla 5.6: Tabla descripción de las características de la memoria FLASH

estado sólido SSDSA1MH080G1 de Intel, que funciona con la misma tecnología, tiene un tamaño de sector de 512 Bytes [INT13].

Además una memoria FLASH especifica su ciclo de vida en el número de operaciones de borrado que se pueden realizar, típicamente entre 100.000 y 1.000.000.

### 5.2.2 Dispositivo GPS

En la Tabla 5.8 podemos ver las características principales del dispositivo GPS [Glo13a].

Fundamentalmente el factor de mayor importancia del dispositivo GPS es cómo realiza la comunicación con el resto del sistema y cómo podemos comunicarnos con él y cómo configurarlo.

No es necesario centrarse en la configuración de los pines de entrada/salida del dispositivo GPS ya que está integrado en la misma placa donde están integrados el resto de dispositivos.

| Bloque | Nombre | Dirección del bloque | Tamaño |
|---|---|---|---|
| Memoria principal | Sector 0 | 0x0800 0000<br>0x0800 3FFF | 16 KByte |
| | Sector 1 | 0x0800 4000<br>0x0800 7FFF | 16 KByte |
| | Sector 2 | 0x0800 8000<br>0x0800 BFFF | 16 KByte |
| | Sector 3 | 0x0800 C000<br>0x0800 FFFF | 16 KByte |
| | Sector 4 | 0x0801 0000<br>0x0801 FFFF | 64 KByte |
| | Sector 5 | 0x0802 0000<br>0x0803 FFFF | 128 KByte |
| | Sector 6 | 0x0804 0000<br>0x0805 FFFF | 128KByte |
| | ... | ... | ... |
| | Sector 11 | 0x080E 0000<br>0x080F FFFF | 128 KByte |
| Memoria de sistema | | 0x1FFF 0000<br>0x1FFF 77FF | 30 KByte |
| Area OTP (one-time programmable) | | 0x1FFF 7800<br>0x1FFF 7A0F | 528 Bytes |
| Bytes de opciones | | 0x1FFF C000<br>0x1FFF C00F | 16 Bytes |

Tabla 5.7: Organización de los módulos de la memoria FLASH

| Atributo | Descripción |
|---|---|
| Nombre: | PA6H GPS Module |
| Definido por: | GlobalTop Technology |
| Documentación: | `http://www.gtop-tech.com/en/product/PA6H-GPS-Module/MT3339_GPS_Module_04.html` |
| Características: | - Chipset Mediatek MT3339.<br>- Antena GPS integrada con soporte para antena externa.<br>- Alta sensibillidad -165dBm, diseño TCXO.<br>- Soporte DPGS (WAAS/EGNOS/MASA/GAGAN).<br>- Precisión de 3 metros.<br>- Soporte protocolos NMEA 0183 y PMTK.<br>- Conectado a la placa a través de la USART 3. |

Tabla 5.8: Tabla descripción de las características del dispositivo GPS

Sobre la codificación de los datos que nos proporciona el GPS se ha decido utilizar el estándar de la Asociación Nacional de Electrónica Marina de Estado Unidos (NMEA) [Glo13b].

NMEA es un estándar de codificación de datos que proporciona una especificación de cómo

deben codificarse los datos de diversos dispositivos electrónicos como pueden ser el GPS, sónar, radar, giroscopio, anemómetro, etc. Este estándar documenta multitud de diferentes tipos de mensajes para muy diferentes componentes. La forma de normalizar todos estos mensajes es mediante una cadena de control que comienza con el símbolo del dólar ($), seguido de un identificador del dispositivo de dos caracteres(GP en el caso del GPS) al que acompaña un identificador de tres caracteres del dato al que se refiere el mensaje. A partir de ahí, y separados por comas, se suceden todos los datos que correspondan a un paquete de datos. El final del mensaje de este protocolo es un símbolo de asterisco (*) seguido de una suma de control de los datos y los caracteres <CR><LF>de fin de mensaje [NME08].

Con respecto al dispositivo GPS solo se tendrán en cuenta los datos que ofrece, ya que la configuración por defecto sirve para el propósito planteado. De los datos que ofrece solo interesarían los identificados por los caracteres GGA, que hacen referencia a los «Datos Corregidos del Sistema de Posicionamiento GLobal». Este mensaje proporciona la posición, separada en longitud, latitud y altura sobre el nivel del mar; el tiempo UCT en el que se generó el mensaje; y si se ha aplicado algún factor de corrección. Se puede ver un ejemplo de mensaje y el significado de los diversos campos en la Tabla 5.9.

| **Nombre** | **Ejemplo** | **Descripción** |
|---|---|---|
| Identificador de mensaje | $GPGGA | Cabecera del protocolo GGA |
| Tiempo UTC | 064951.000 | hhmmss.sss |
| Latitud | 2307.1256 | ddmm.mmmm |
| Indicador N/S | N | N=norte, S=sur |
| Longitud | 12016.4438 | dddmm.mmmm |
| Indicador E/O | E | E=este, W=oeste |
| Indicador de posición | 1 | 0=no disponible, 1=ajustado, 2=diferente ajuste del GPS |
| Nº de satélites usados | 8 | Entre 0 y 14 |
| HDOP | 0.95 | Dilución de la precisión horizontal |
| Altitud sobre el mar | 39.9 | Altitud, en metros, de la antena sobre el nivel del mar |
| Unidades | M | Metros de altitud de la antena |
| Separación geoidal | 17.8 | |
| Unidades | M | Metros de la separación geoidal |
| Age of Diff. Corr. | | Nulo cuando no se utiliza DGPS en segundos |
| Checksum | *65 | |
| CR LF | | Fin de mensaje |

Tabla 5.9: Estructura de los datos porporcionados por el GPS

### 5.2.3 Placa STMF205VGT6

Sobre la placa objetivo, al igual que se ha hecho con el GPS, solo se describirán aquellos aspectos que son relevantes para la realización de este TFG, se puede ver en la Tabla 5.10 [Mic13a].

| Atributo | Descripción |
|---|---|
| Nombre: | STMF205VGT6 |
| Definido por: | ST Microelectronics |
| Documentación: | `http://www.st.com/web/en/catalog/mmc/FM141/SC1169/SS1575/LN1433/PF245091` |
| Características: | - Core ARM 32-bit Cortex-M3 CPU.<br>- 1024 KB FLASH Memory.<br>- 128 KB SRAM.<br>- Comunicaciones UART, USART, SPI, I2C, CAN, USB 2.0.<br>- DMAC.<br>- Tanto la memoria FLASH, como el procesador y las UART y USART están conectadas entre sí gracias a una matriz de buses AHB.<br>- Acelerómetro.<br>- 16 canales conversor analógico-digital.<br>- 2 canales conversor digital-analógico.<br>- Timers de 16 y 32 bits.<br>- Oscilador interno de 32 KHz, uno de cristal de 4 a 20 MHz.<br>- Debug serie (SWD), JTAG y Cortex-M3 Trace.<br>- Generador de números aleatorios.<br>- 3 modos de arranque: desde FLASH, desde memoria de sistema o desde SRAM.<br>- El boot-loader está en la memoria de sistema y sirve para reprogramar la FLASH.<br>- Tiene un Controlador de interrupciones apiladas (NVIC). |

Tabla 5.10: Cuadro descripción de las características de la placa STMF205VGT6

## 5.3 Conocimiento software de acceso al hardware

Una vez conocidos los componentes hardware a utilizar y sus principales características, a continuación se describirán las bibliotecas de funciones proporcionadas por el fabricante de los mismos.

En la Figura 5.2 se puede ver el diagrama estructural de la biblioteca del fabricante dividida en las diferentes capas que la componen.

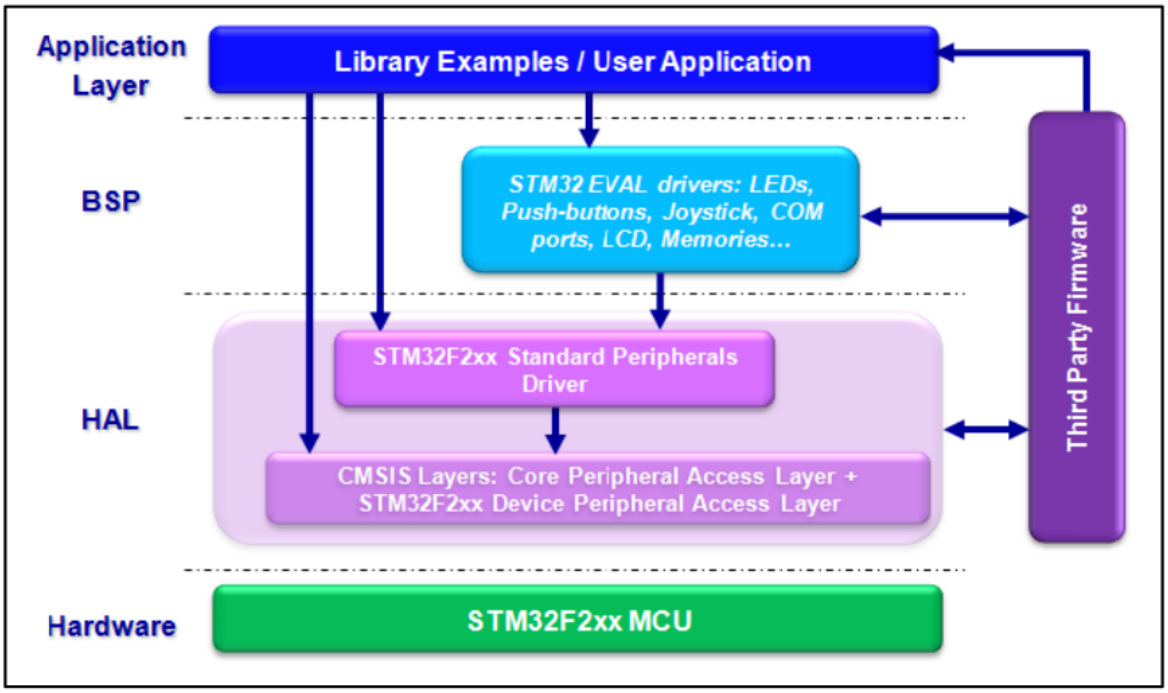



Figura 5.2: Diagrama en capas de la organización de la biblioteca de funciones de la placa

### 5.3.1 Biblioteca de funciones de la Placa STMF205VGT6

La biblioteca de funciones proporcionada por el fabricante de la placa, ST Microelectronics, y consta de los siguientes archivos [Mic13b]:

- misc.h: Agrupa funciones del Controlador del Vector de Interrupciones Apiladas (NVIC).
- stm32f2xx_adc.h: Agrupa funciones del conversor Analógico/Digital.
- stm32f2xx_can.h: Agrupa funciones de control del bus CAN.
- stm32f2xx_crc.h: Agrupa funciones de control de redundancia cíclica.
- stm32f2xx_cryp.h: Agrupa funciones de cifrado de datos.
- stm32f2xx_dac.h: Agrupa funciones del conversor Digital/Analógico.
- stm32f2xx_dma.h: Agrupa funciones relacionadas con la configuración y control del DMA.
- stm32f2xx_exti.h: Agrupa funciones de configuración de interrupciones externas.
- stm32f2xx_flash.h: Agrupa funciones de configuración y uso de la memoria FLASH.
- stm32f2xx_fsmc.h: Agrupa funciones de configuración y uso de memoria externa a través del FSMC.
- stm32f2xx_gpio.h: Agrupa funciones de configuración y uso de los puertos de Entrada/Salida de propósito general.

- stm32f2xx_hash.h: Agrupa funciones de control de integridad de datos.
- stm32f2xx_i2c.h: Agrupa funciones de control del bus I2C.
- stm32f2xx_iwdg.h: Agrupa funciones de control del Watchdog.
- stm32f2xx_pwr.h: Agrupa funciones de control del voltaje.
- stm32f2xx_rcc.h: Agrupa funciones de control de los timers.
- stm32f2xx_rng.h: Agrupa funciones relacionadas con la generación de números aleatorios.
- stm32f2xx_rtc.h: Agrupa funciones relacionadas con el control y configuración del reloj de tiempo real.
- stm32f2xx_sdio.h: Agrupa funciones relacionadas con el control del *Secure Digital Input/Output*.
- stm32f2xx_spi.h: Agrupa funciones de configuración del bus SPI.
- stm32f2xx_syscfg.h: Agrupa funciones relacionadas con la configuración del sistema.
- stm32f2xx_tim.h: Agrupa funciones relacionadas con la configuración de los timers y los preescaler.
- stm32f2xx_usart.h: Agrupa funciones relacionadas con la configuración de las Receptor/Transmisor Universal Asíncrono (USART).
- stm32f2xx_wwdg.h: Agrupa funciones relacionadas con la configuración de la ventana de tiempos que maneja el *watchdog*.

### 5.3.2 Biblioteca de funciones de la Memoria FLASH

A continuación se listan de la biblioteca de funciones de acceso a la memoria FLASH los métodos que han sido utilizados durante el desarrollo de este TFG. Esta biblioteca ha sido proporcionada por el fabricante de la placa y aparece nombrada en la sección 5.3.1. Consta de los siguientes archivos: *stm32f2xx_flash.h* y *stm32f2xx_flash.c*. El listado de funciones se puede ver en el Listado 5.1.

```
/* Desbloquea la FLASH y permite la escritura */
void FLASH_Unlock(void);

/* Bloquea la FLASH e impide la escritura */
void FLASH_Lock(void);

/* Resetea el sector especificado y lo configura con un determinado
   voltaje */
FLASH_Status FLASH_EraseSector(uint32_t FLASH_Sector, uint8_t
   VoltageRange);
```

```
/* Escribe en la direccion proporcionada un dato del tamanyo de una
   palabra (32 bits) */
FLASH_Status FLASH_ProgramWord(uint32_t Address, uint32_t Data);

/* Limpia todos los flags de estado de la FLASH. Es la unica forma de
   recuperar el control de la FLASH si suceden determinados estados
   de error. */
void FLASH_ClearFlag(uint32_t FLASH_FLAG);

/* Nos proporciona el estado actual de la FLASH */
FLASH_Status FLASH_GetStatus(void);

/* Operacion que hace una espera mientras terminan todas las
   operaciones pendientes de la FLASH */
FLASH_Status FLASH_WaitForLastOperation(void);
```

Listado 5.1: Biblioteca de acceso a la memoria FLASH

### 5.3.3 Biblioteca de funciones del dispositivo GPS

El fabricante del dispositivo GPS no proporciona ninguna biblioteca de funciones para facilitar su uso. El dispositivo GPS se conecta a la placa a través de la USART 3, por lo que se deben utilizar las funciones proporcionadas por el fabricante de la placa para la comunicación y gestión de los dispositivos conectados a estos puertos.

Una USART, es un componente hardware que controla los puertos y sirve de interfaz estándar de comunicación con otros dispositivos. El funcionamiento básico de una USART es recibir un stream de datos y enviarlos secuencialmente por los puertos de entrada/salida de los que disponga a otra USART que se conectará directamente a esos mismos puertos.

## 5.4 Análisis del acceso a la memoria FLASH

Para la programación de la memoria FLASH, una vez estudiadas las bibliotecas de funciones proporcionadas y su funcionamiento interno se procedió a programar una serie de test que demostraron que se adquirieron los conocimientos suficientes para afrontar la tarea de la programación de un sistema de ficheros y que la memoria funciona correctamente.

Las operaciones sobre la memoria FLASH se pueden resumir en tres operaciones: lectura, escritura y borrado. Las de escritura y borrado se pueden agrupar en una única operación ya que el flujo de ejecución es el mismo hasta que se especifica a la función de escritura o borrado. En la Figura 5.3 podemos ver el flujo de ejecución de los tres tipos de operaciones.

La memoria FLASH, al estar integrada dentro de la placa, no necesita ningún tipo de inicialización, aunque es recomendable limpiar el estado de la última operación realizada la

última vez que se utilizó.

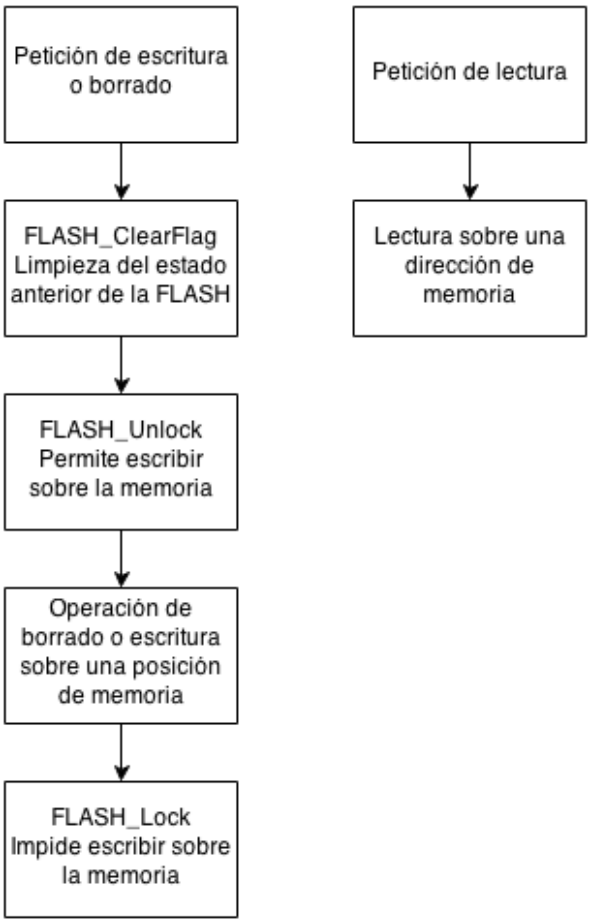



Figura 5.3: Flujo de ejecución de las operaciones sobre la memoria FLASH

A continuación se muestran los test realizados junto a un diagrama de flujo que nos proporcione una guía visual sobre las funciones que realiza.

- int test_unlock_lock(): Este test prueba a desbloquear la memoria y volver a bloquearla, controlando en todo momento el estado de la memoria por si surge algún error. Podemos ver el diagrama de secuencia e interacción con la biblioteca de la memoria FLASH en la Figura 5.4.
- int test_flash_program_example(): Este test prueba a desbloquear la memoria, resetear el sector, escribir y leer de memoria, controlando en todo momento el estado de la memoria por si sirge algun error. Podemos ver el diagrama de secuencia e interacción con la biblioteca de la memoria FLASH en la Figura 5.4. Este test es un ejemplo proporcionado por el fabricante.
- int diskRead(uint32_t address, BYTE *buff, UINT count): Este test prueba a leer de memoria, controlando en todo momento el estado de la memoria por si surge algún

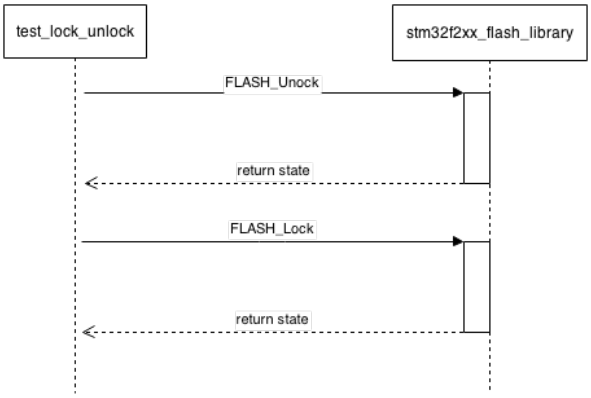



Figura 5.4: Diagrama de secuencia e interacción entre el test de *Unlock/Lock* y la biblioteca de acceso a la FLASH

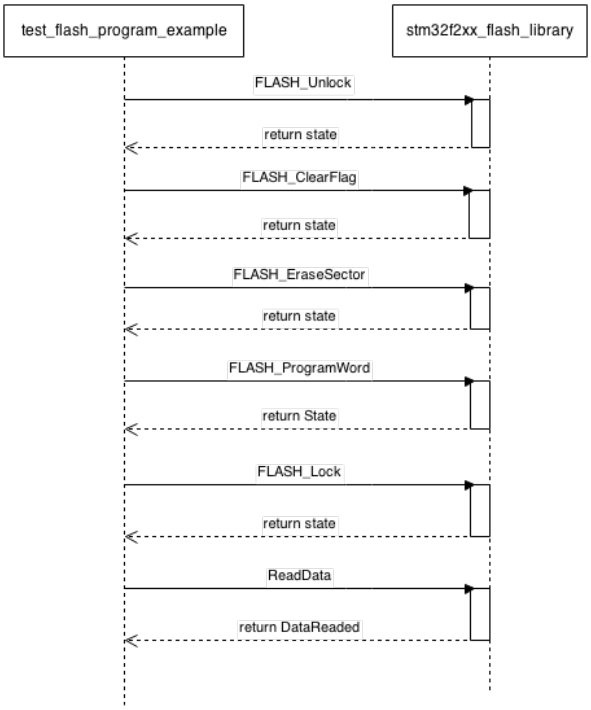



Figura 5.5: Diagrama de secuencia e interacción entre el test de ejemplo y la biblioteca de acceso a la FLASH

error. Podemos ver el diagrama de secuencia e interacción con la biblioteca de la memoria FLASH en la Figura 5.6.

- int diskWrite(uint32_t address, const BYTE *buff, UINT count): Este test prueba a escribir en memoria, controlando en todo momento el estado de la memoria por si surge algún error. Podemos ver el diagrama de secuencia e interacción con la biblioteca de la memoria FLASH en la Figura 5.7.
- int test_write_read(): Este test es una combinación de los test de escritura y lectura y comprobando que se escriben y leen los mismos datos. En este test no se detalla el

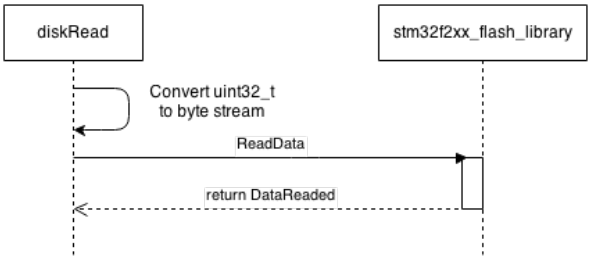



Figura 5.6: Diagrama de secuencia e interacción entre el test de lectura y la biblioteca de acceso a la FLASH

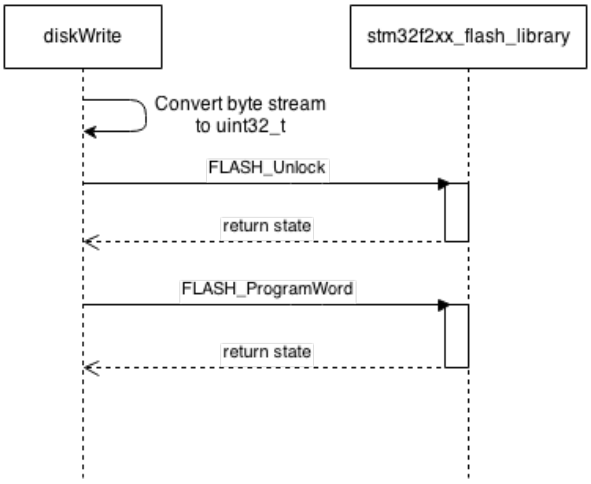



Figura 5.7: Diagrama de secuencia e interacción entre el test de escritura y la biblioteca de acceso a la FLASH

diagrama de secuencia ya que las interacciones con la biblioteca de acceso a la FLASH son las mismas que los test citados.

## 5.5 Análisis del acceso al GPS

Como se ha visto antes, para la comunicación y gestión del GPS se utilizan los puertos correspondientes a la USART 3, a la que está conectado. En este apartado se definen los métodos utilizados para la inicialización y lectura de datos del dispositivo GPS.

En esta tarea también se ha utilizado uno de los leds, para tener certeza que se está ejecutando correctamente, que se apagará antes de parsear un mensaje y se encenderá cuando se haya parseado correctamente.

El dispositivo GPS necesita una inicialización mediante la configuración de la USART a la que está conectado para poder realizar la comunicación con él correctamente. Además necesita que se haya configurado previamente el reloj del sistema ya que es uno de los requi-

sitos de la USART para establecer la comunicación. En la Figura 5.8 se puede ver el flujo de ejecución del acceso al dispositivo GPS.

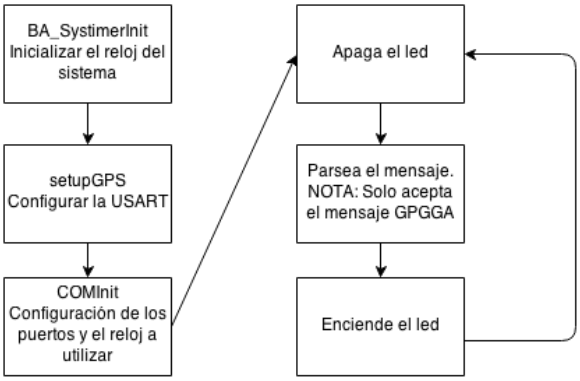



Figura 5.8: Flujo de ejecución de las operaciones de acceso al GPS

No se dispone de forma de comprobar que los datos del GPS son correctos salvo teniendo otro dispositivo GPS cercano y comprobando los datos, o utilizando alguna herramienta de cartografía como puede ser Google Maps. Por este motivo en este punto no se realizan test de programación.

A continuación se va listar los métodos necesarios para inicializar el GPS y lectura de los datos, junto con un diagrama de secuencia para entender cómo se hace y con qué componentes interacciona.

- Inicializar Hardware: En este método se inicializa el hardware del GPS, se configura el reloj del sistema, necesario para que se establezca correctamente la comunicación con el dispositivo y se configuran los puertos de la USART utilizada. Se puede ver un diagrama de secuencia e interacción con las bibliotecas de funciones del sistema en la Figura 5.9.

- ParsearDatosGPS: En este método se parsean los datos del GPS para que reconozca únicamente los mensajes definidos en el apartado 5.2. Este método se ejecuta en bucle, cuando se integre la parte del Sistema de Archivos se escribirán los mensajes válidos en un fichero. Para la lectura de datos utiliza la función implementada en los drivers del sistema *BA_GPS_Receive*. Se puede ver un diagrama de secuencia e interacción con las bibliotecas de funciones del sistema en la Figura 5.10.

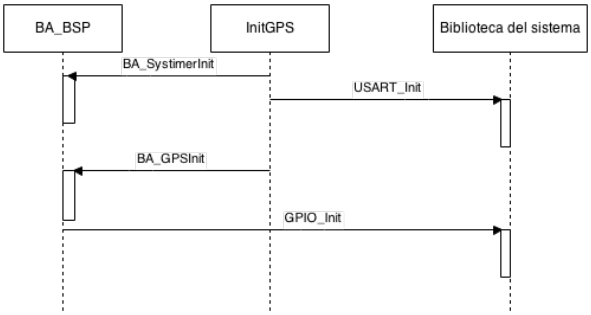



Figura 5.9: Diagrama de secuencia e interacción de las operaciones de inicialización del dispositivo GPS

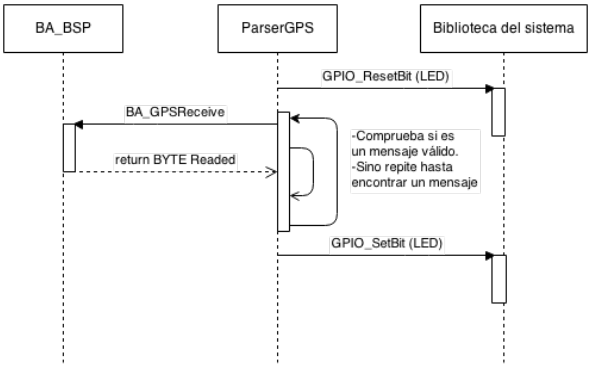



Figura 5.10: Diagrama de secuencia e interacción de las operaciones de lectura y parseo de datos del GPS

## 5.6 Implementación de un Sistema de Ficheros

### 5.6.1 Introducción y evaluación de alternativas

Un sistema de archivos es necesario para proporcionar una capa de abstracción en el acceso a la memoria y el almacenamiento de datos.

La implementación de un sistema de ficheros en un sistema empotrado debe ser, como el resto de componentes, lo más eficiente posible. Como los sistemas de archivos llevan muchos tiempo entre nosotros y hay muchas y diferentes alternativas se ha pensado como primera opción utilizar un sistema FAT por su simplicidad.

Es necesario considerar los requisitos de la memoria FLASH:

- 1 MByte de memoria.
- La memoria está dividida en 11 sectores de diferentes tamaños, desde 16 KBytes hasta 128 KBytes.
- La granularidad mínima del borrado de la memoria, necesario para poder sobreescribir,

es a nivel de sector físico y se pierde toda la información almacenada.

- La memoria del programa está escrita en la memoria FLASH, por lo que no tenemos disponible toda la capacidad de la memoria para el sistema de archivos.

Después de estudiar el tema se descartó el uso de un sistema FAT o cualquiera conocido ya que todos funcionan con una granularidad mínima de un sector físico y en este sistema no sirven ya que, como se ha dicho anteriormente, solo se dispone de 11 sectores físicos y además con un tamaño demasiado grande (128 KBytes) en comparación con el tamaño de sector habitual de este tipo de memorias (512 Bytes).

### 5.6.2 Diseño del sistema de archivos

Para crear un sistema de archivos es necesaria una tabla con los descriptores de archivos que nos proporcionen cierta información como puede ser el nombre, necesario para realizar la búsqueda, y la dirección de inicio.

Se ha decidido que, por simplicidad y para evitar fragmentación interna, el almacenamiento de los archivos en memoria sea de manera secuencial. Con esto surge un problema debido a que tener varios archivos en memoria implica que cada archivo tiene un límite máximo. Para solucionar el problema mencionado se ha definido un número máximo de ficheros, ya que para este sistema se necesita únicamente uno para datos de configuración y otro para almacenamiento de log, o varios si se desea discriminar por fuentes de datos.

La tabla de descriptores se ha definido de la siguiente forma:

- Un byte de validez, puesto todo a 1 por defecto ya que solo se puede cambiar el valor a 0. Para ver si una entrada es válida basta con comprobar el byte de validez y que la entrada en la tabla no tenga vacío el campo del nombre.
- El nombre es una cadena de bytes que codifica el nombre.
- La dirección de inicio será la dirección donde empieza el fichero.
- La dirección de fin será la siguiente posición de memoria consecutiva a la última posición escrita.

A partir de estos elementos es necesario ahora definir el tamaño de un sector lógico para facilitar las tareas de lectura y escritura. Partiendo de lo anterior, y de las posibilidades que ofrece la biblioteca de funciones de acceso a la memoria FLASH, es necesario definir un sector lógico de un tamaño múltiplo de una palabra de datos (32 bits o 4 bytes) y del tamaño

suficiente para albergar una entrada completa de la tabla de descriptores. El tamaño de cada sector lógico se ha establecido a 16 bytes, pero es configurable.

Con un sector lógico de 16 bytes cada entrada de la tabla de descripciones tiene la siguiente estructura: 1 Byte de validez, 7 Bytes de codificación del nombre, 4 Bytes para codificar la dirección de inicio y 4 Bytes para codificar la dirección de fin.

Para facilitar la tarea de reconocer que existe un sistema de archivos escrito en memoria cuando montamos el sistema, el primer sector lógico contiene una cadena de texto fácilmente identificable: "FS CORRECTO".

En la Tabla 5.11 [1] se puede ver un ejemplo de los datos que podría contener la tabla de descriptores de archivos.

| Validez | Nombre | Dirección de inicio | Dirección de fin |
|---|---|---|---|
| 255 | archivo | 2048 | 2184 |
| 255 | ex1.txt | 4112 | 4140 |
| 255 | 1111111 | 4294967295 | 4294967295 |

Tabla 5.11: Ejemplo de datos de la tabla de descriptores de archivos

Una entrada en la tabla de descriptores de archivo es válida si el byte de validez tiene un valor de 255 y la dirección de inicio y el nombre tienen valores válidos.

Para solucionar el problema de que poder escribir un dato en una posición ya utilizada sin antes realizar una operación de Erase en todo el sector, se creará una nueva tarea que comprobará cada cierto tiempo el uso del sistema de archivos utilizado y, si está lo suficientemente completo, creará un nuevo sistema de archivos vacío en otro sector y será el que se utilice desde ese momento en adelante hasta que sea necesario volver a cambiar el sector físico donde tenemos el sistema de archivos.

### 5.6.3 Definición de la biblioteca de acceso al Sistema de Archivos

La biblioteca de funciones del Sistema de Archivos desarrollado se estructura en dos capas bien diferenciadas: 1)la capa del sistema de archivos propiamente dicha y 2) la capa de abstracción del hardware que ofrece simplicidad a la hora de realizar diversas operaciones

[1] El byte de validez con valor 255 corresponde con el número en binario 1111, el número 4294967295 corresponde con el número binario 1111 1111 1111 1111.

sobre la memoria FLASH.

A continuación se listan las funciones del Sistema de Archivos desarrollado y las tareas que realiza cada una, así como un diagrama de secuencia de cada una para explicar cómo funcionan internamente y las interacciones con las diferentes capas de abstracción del sistema.

- f_mount: Crea la estructura de datos necesaria para almacenar los datos del sistema de archivos y le los datos del sistema de archivos del disco. Comprueba si hay un sistema de archivos correcto y extrae los ficheros de la tabla de descriptores. Se puede ver el diagrama de secuencia de esta función en la Figura 5.11.

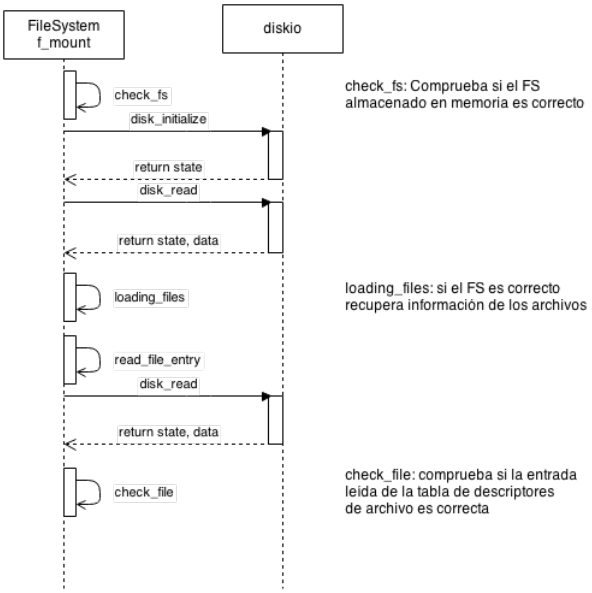



Figura 5.11: Diagrama de secuencia e interacción de la operación f_mount del Sistema de Archivos

- f_mkfs: Crea el sistema de archivos, lo configura y escribe en el disco lo que necesite. Se puede ver el diagrama de secuencia de esta función en la Figura 5.12.

- f_open: Dependiendo del modo de apertura abrirá un archivo existente o creará uno nuevo. Se puede ver el diagrama de secuencia de esta función en la Figura 5.13.

- f_close: Cerrará el archivo. En este momento se vuelcan en memoria todos los datos no guardados que se encuentren en el buffer de escritura y se actualizará la tabla de descriptores de archivo. Se puede ver el diagrama de secuencia de esta función en la

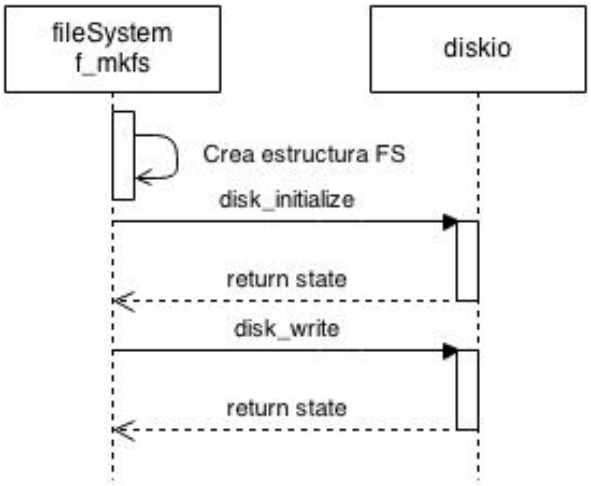



Figura 5.12: Diagrama de secuencia e interacción de la operación f_mkfs del Sistema de Archivos

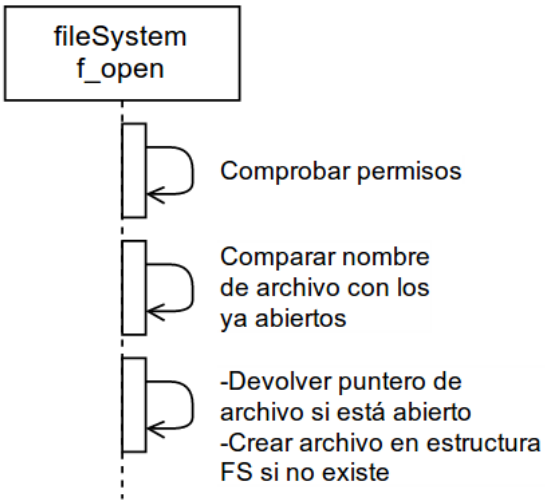



Figura 5.13: Diagrama de secuencia e interacción de la operación f_open del Sistema de Archivos

Figura 5.14.

- f_read: Se leerá de disco el tamaño en bytes especificado al invocar la función. Se puede ver el diagrama de secuencia de esta función en la Figura 5.15.

- f_write: Se escribirá en disco el tamaño en bytes especificado al invocar la función. Si el tamaño especificado es menor que el tamaño del buffer (el mismo tamaño que el sector) entonces no se escribirá hasta que el buffer de escritura se llene. Se puede ver el diagrama de secuencia de esta función en la Figura 5.16.

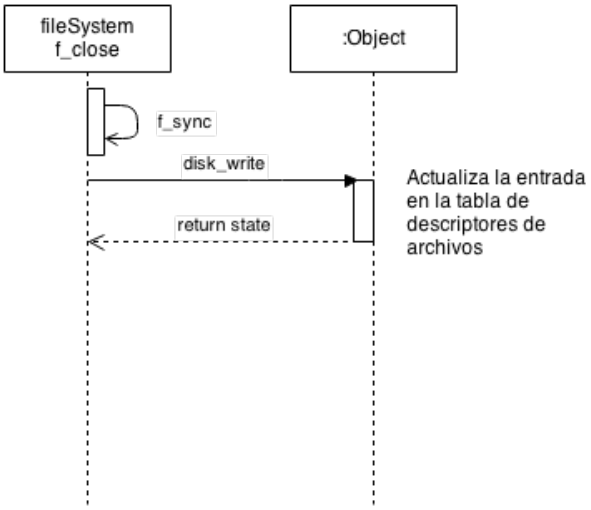



Figura 5.14: Diagrama de secuencia e interacción de la operación f_close del Sistema de Archivos

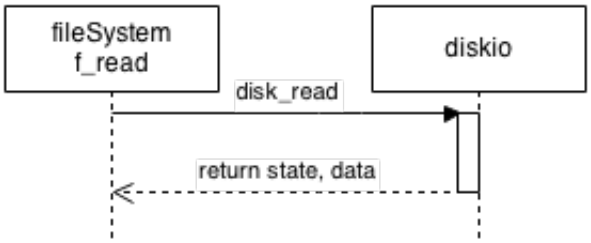



Figura 5.15: Diagrama de secuencia e interacción de la operación f_read del Sistema de Archivos

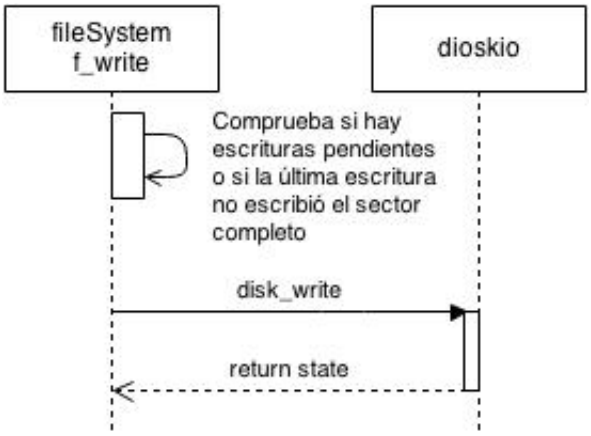



Figura 5.16: Diagrama de secuencia e interacción de la operación f_write del Sistema de Archivos

- f_lseek: Cambia el puntero de lectura a la dirección especificada. Cambia el valor del campo correspondiente en la estructura de datos que representa a un archivo.

- f_sync: Fuerza las escrituras pendientes que haya. Se puede ver el diagrama de secuencia de esta función en la Figura 5.17.

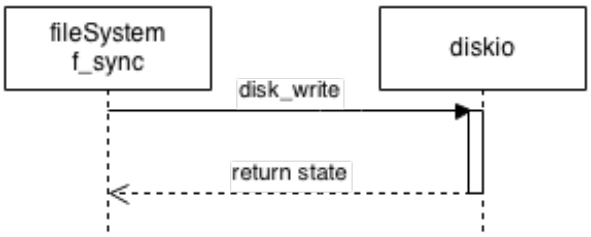



Figura 5.17: Diagrama de secuencia e interacción de la operación f_sync del Sistema de Archivos

- f_truncateStart: Modificará la posición de inicio de un archivo. En la práctica consiste en el borrado de la parte más antigua del fichero. Cambia el valor del campo correspondiente en la estructura de datos que representa a un archivo y se actualizará en la tabla de descriptores de archivo cuando éste se cierre.

- f_getfree: Obtiene el mínimo de los tamaños disponibles para escritura de archivos. Se puede ver el diagrama de secuencia de esta función en la Figura 5.18.

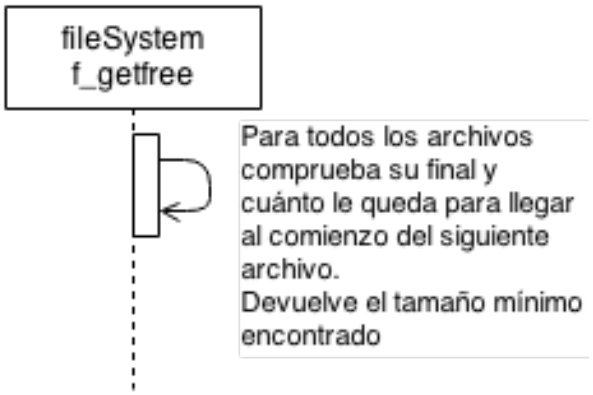



Figura 5.18: Diagrama de secuencia e interacción de la operación f_getfree del Sistema de Archivos

- reset_sector: Realiza una operación de *Reset* (o *Erase*) en el sector, dejándolo preparado para nuevas escrituras. Se puede ver el diagrama de secuencia de esta función en la Figura 5.19.
- change_sector: Cambia el sector físico en el que se encuentra el sistema de archivos. Realiza todos los pasos necesarios para crear un sistema de archivos en otro sector especificado en el archivo de configuración. Resetea el sector, crea el sistema, lo monta

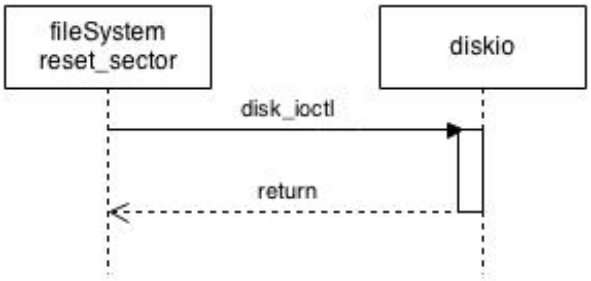



Figura 5.19: Diagrama de secuencia e interacción de la operación reset_sector del Sistema de Archivos

y, si se le indicamos, hace un *backup* de los archivos de un sistema en el otro. Se puede ver el diagrama de secuencia de esta función en la Figura 5.20.

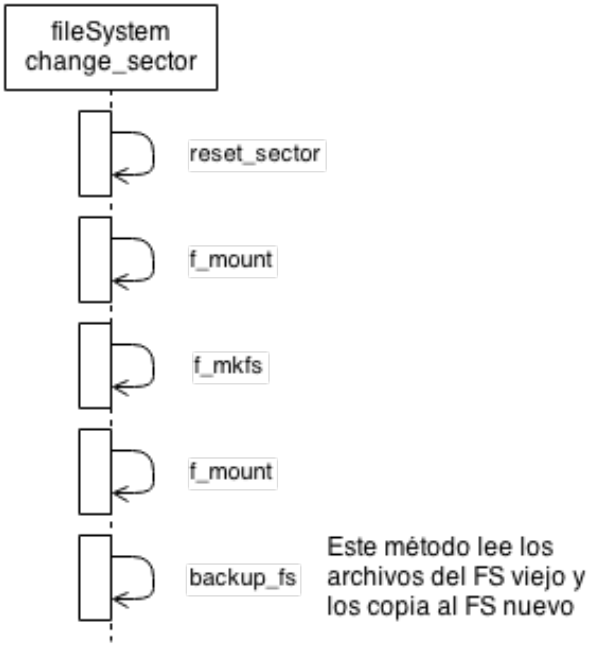



Figura 5.20: Diagrama de secuencia e interacción de la operación change_sector del Sistema de Archivos

Para separar la gestión del sistema de archivos de los accesos a disco y simplificar la programación, el sistema de archivos propuesto ha sido programado sobre una biblioteca de acceso a la memoria FLASH que tiene las siguientes funciones:

- disk_initialize: Inicializará la memoria FLASH. En la práctica la operación consiste en limpiar todos los registros de estado. Se puede ver el diagrama de secuencia de esta función en la Figura 5.21.

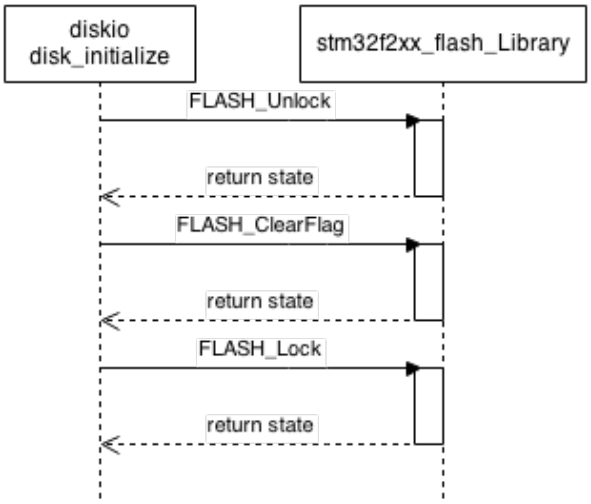



Figura 5.21: Diagrama de secuencia e interacción de la operación disk_initialize de la capa de abstracción de la memoria FLASH

- disk_status: Devuelve el estado actual de la memoria FLASH. Se puede ver el diagrama de secuencia de esta función en la Figura 5.22.

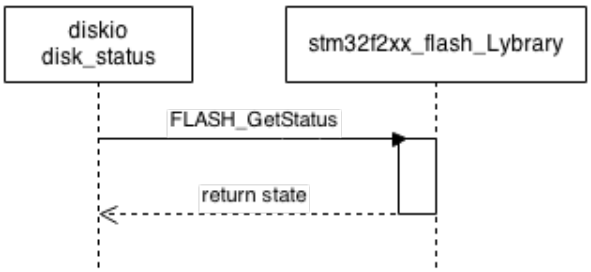



Figura 5.22: Diagrama de secuencia e interacción de la operación disk_status de la capa de abstracción de la memoria FLASH

- disk_read: Lee, a partir de una dirección de memoria dada, la cantidad de sectores especificados. Se puede ver el diagrama de secuencia de esta función en la Figura 5.6.

- disk_write: Escribe, a partir de una dirección de memoria dada, la cantidad de setores especificados. Se puede ver el diagrama de secuencia de esta función en la Figura 5.7.

- disk_ioctl: Agrupa algunas funciones de control, se especificará en uno de los parámetros de entrada qué queremos hacer. Las operaciones soportadas serán:
  - CTRL_SYNC: Esperará hasta que todas las operaciones sobre la FLASH se hayan completado.

- GET_SECTOR_COUNT: Devolverá el número total de sectores presentes en el sistema de archivos.
- GET_SECTOR_SIZE: Devolverá el tamaño de los sectores del sistema de archivos.
- CTRL_ERASE_SECTOR: Realizará la operación de *Reset/Erase* sobre el sector especificado.

Se puede ver el diagrama de secuencia de esta función en la Figura 5.23.

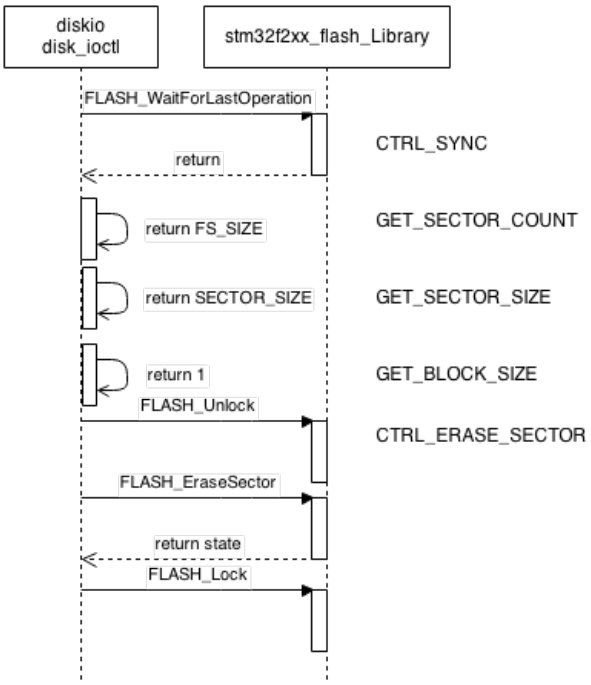



Figura 5.23: Diagrama de secuencia e interacción de la operación disk_ioctl de la capa de abstracción de la memoria FLASH

## 5.7 Programación y configuración del Sistema Operativo en Tiempo Real FreeRTOS

En un sistema empotrado el uso de un sistema operativo es algo crítico ya que, aunque ayuda con la programación del sistema proporcionándonos una capa de abstracción para el manejo de tareas y cambios de contexto, los recursos que consume pueden ser considerables.

A continuación se expondrán una serie de características inherentes a todo sistema operativo de tiempo real y algunos de sus inconvenientes.

- Cada tarea tiene su propio estado y se ejecuta como un programa indepentiente.

- La ejecución de las tareas se maneja mediante el planificador del Sistema Operativo.
- Cuando ocurre una interrupción se suspende la ejecución de una tarea y se ejecuta la tarea asociada a la interrupción.
- Los mecanismos de coordinación entre tareas son las interrupciones, los semáforos y las colas de mensajes. Hay que tener especial cuidado ya que pueden darse condiciones de carrera[2].
- Cada tarea tiene una zona reservada de memoria, o pila, y puede desbordarse si no se ha reservado suficiente espacio.

### 5.7.1 Gestión de la memoria RAM

La gestión de la memoria RAM, como se ha comentado anteriormente, es esencial para el correcto funcionamiento del sistema operativo y las tareas. En la Figura 5.24 se puede ver la disposición en memoria de los datos que se describen a continuación.

- En la parte inferior de la memoria se guarda la pila de cada tarea.
- En la parte superior de la memoria se guarda la pila del sistema.
- En la pila del sistema se guarda, principalmente, la pila de interrupciones.
- La pila de datos es donde se guardan todos los datos compartidos.

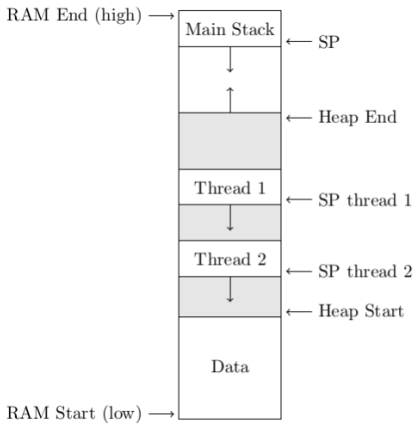



Figura 5.24: Disposición de la memoria de las tareas en memoria RAM

Para la gestión de la memoria RAM que utilizan las tareas se ha optado por la utilización de la opción 2 que proporciona FreeRTOS en el archivo *heap2.c*, que permite liberar bloques usados para volver a ser utilizados posteriormente. La cantidad total de espacio dinámico

[2]Una condición de carrera sucede cuando varios procesos acceden a un mismo recurso y el resultado de las operaciones cambia dependiendo del orden de ejecución de las mismas.

es determinado en el archivo de configuración. Esta configuración tiene el inconveniente de que no es capaz de agrupar en una porción de memoria mayor partes adyacentes que hayan sido liberadas, por lo que solo podrán utilizarse para tareas o estructuras de datos de igual o menor tamaño, incrementando la fragmentación interna. En el caso de la gestión de tareas de este TFG no existe problema con la fragmentación interna, ya que no se eliminan tareas y las estructuras de datos son siempre del mismo tamaño.

### 5.7.2 Gestión de las tareas

Las tareas, o Threads, que maneja FreeRTOS son con prioridad de apropiación, es decir, que cualquier tarea con mayor prioridad va a ejecutarse si está preparada. Las tareas con igual prioridad son ejecutadas alternándose entre sí con un tiempo de *poolling* determinado.

En la Figura 5.25 se puede ver la máquina de estados de cada tarea. A continuación se pasan a describir los estados y cómo pasan de uno a otro.

- Cuando se crea una tarea su estado 0.
- Existe una única una tarea con estado 2.
- Una tarea ejecutándose puede ser pausada y volver al estado 1.
- Una tarea ejecutándose puede ser bloqueada y pasar al estado 3.
- Una tarea bloqueada puede pasar al estado 1.
- El estado suspendido de FreeRTOS se utiliza con la llamada vTaskSuspendAll(). Esta función pasa todas las tareas a estado suspendido excepto aquella que ha invocado la función.

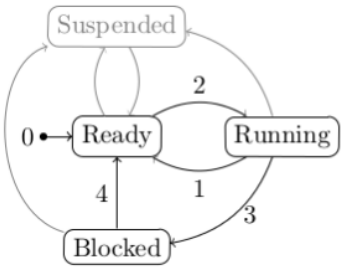



Figura 5.25: Máquina de estados de las tareas

Cada tarea es creada con una función a ejecutar, un nombre, un tamaño de pila, algunos parámetros opcionales, una prioridad y la localización del manejador de la tarea. A partir de aquí el sistema operativo crea la estructura necesaria y la coloca en memoria donde corresponda. El tamaño de la pila es crítico ya que si es demasiado pequeño es probable que haya

desbordamiento y si es demasiado grande puede que no exista espacio para otras tareas. Como dato, utilizar bibliotecas de funciones incrementa significativamente el tamaño necesario en la pila al tener que almacenar funciones que nunca van a ser utilizadas.

Para la gestión del tiempo en que todas las tareas puedan estar bloqueadas esperando recursos, FreeRTOS crea una tarea vacía con la menor prioridad posible.

Los mecanismos de sincronización entre tareas proporcionan una interfaz segura para administrar los recursos compartidos, de manera que cuando una tarea pide un recurso el planificador éste es asignado si está libre o es bloqueada hasta que otra tarea lo libere.

FreeRTOS soporta también colas de mensajes para pasar información de una tarea a otra. Para crearla es necesario especificar un identificador, el tamaño máximo de un mensaje y el máximo de mensajes que puede almacenar. Los mensajes son añadidos a una cola de mensajes copiando la referencia, por lo que consumen memoria de la tarea que los envía, y deben especificar un tiempo máximo de espera para ser atendidos.

Sobre las interrupciones FreeRTOS las trata añadiéndolas a una cola de mensajes, que será tratada por la tarea correspondiente.

### 5.7.3 Funciones necesarias para construir una aplicación con FreeRTOS

En el Listado 5.2 podemos ver las funciones de FreeRTOS que se necesitan utilizar para la realización de este TFG, junto con una pequeña explicación de cada una.

```
/* Creacion de una tarea, necesitamos especificar la funcion   *
 * a lanzar, el nombre de la tarea, el tamanyo de la pila,     *
 * parametros extra si los hubiese, la prioridad y el puntero  *
 * donde quieras que se devuelva el manejador de la tarea.     */
BaseType_t xTaskCreate(
                          TaskFunction_t pvTaskCode,
                          const char * const pcName,
                          unsigned short usStackDepth,
                          void *pvParameters,
                          UBaseType_t uxPriority,
                          TaskHandle_t *pvCreatedTask
                        );

/* Creacion de una cola de mensajes, necesitamos especificar la *
 * longitud de la cola y el tamanyo maximo de mensaje.          */
QueueHandle_t xQueueCreate(
                          UBaseType_t uxQueueLength,
```

```
                            UBaseType_t uxItemSize
                          );

/* Envio de un mensaje a una cola, se debe especificar la cola  *
 * de mensajes objetivo, la variable o estructura a enviar y    *
 * el tiempo maximo de bloqueo esperando sincronizacion.        */
BaseType_t xQueueSend(
                            QueueHandle_t xQueue,
                            const void * pvItemToQueue,
                            TickType_t xTicksToWait
                          );

/* Recepcion de un mensaje a una cola, se debe especificar la   *
 * cola de mensajes objetivo, la variable o estructura a enviar *
 * y el tiempo maximo de bloqueo esperando sincronizacion.      */
BaseType_t xQueueReceive(
                            QueueHandle_t xQueue,
                            void *pvBuffer,
                            TickType_t xTicksToWait
                          );
```

Listado 5.2: Funciones necesarias para construir una aplicación con FreeRTOS

## 5.8 Diseño y programación del sistema final

En las secciones anteriores ya hemos definido el sistema de archivos y cómo utilizarlo (Sección 5.6), el acceso al GPS y cómo utilizarlo (Sección 5.5) y cómo funciona FreeRTOS (Sección 5.7). En esta sección se unifican los módulos individuales ya creados e integran en el sistema final objetivo de este TFG.

En la Figura 5.26 podemos ver el diagrama de componentes del Sistema final, dividido en capas según la funcionalidad de los componentes.

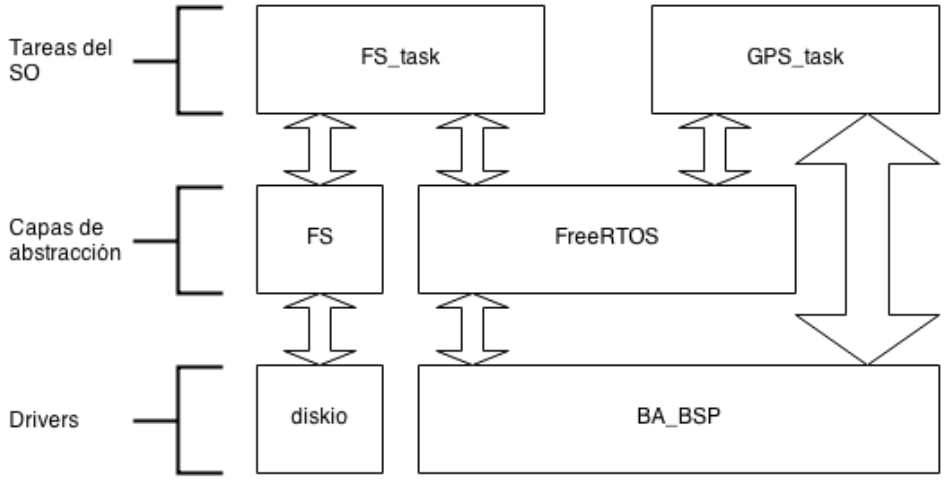



Figura 5.26: Diagrama de componentes del Sistema final

El sistema final constará de dos tareas definidas en FreeRTOS, una para la gestión de la memoria y otra para la gestión del GPS. La comunicación y sincronización entre ambas será realizada a través de una cola de mensajes de FreeRTOS. El problema de concurrencia a afrontar entre estas dos tareas es el problema del productor-consumidor. Para tener alguna interacción con el usuario el sistema va a encender un led verde cada vez que el GPS ofrezca un dato válido y un led rojo cada vez que consuma ese dato y lo escriba en el sistema de archivos.

### 5.8.1 Tarea encargada de acceso al GPS

La tarea encargada de gestionar el GPS va directamente sobre funciones de la biblioteca de funciones proporcionada por el fabricante.

La función que ejecuta la tarea es GPSTaskFunc, que nada más ejecutarse inicializa el componente hardware del GPS y a continuación ejecuta permanentemente la tarea de parsear los datos y enviarlos a la cola de mensajes.

Las funciones de inicialización, comunicación e identificación de los mensajes del dispositivo GPS son los descritos en el apartado 5.5, salvo con la diferencia que la función *parserGPS* envía los mensajes válidos a la cola de mensajes con la que se comunica con la tarea encargada de gestionar la memoria.

El código fuente de la tarea encargada de gestionar el GPS se puede encontrar en el Anexo .

### 5.8.2 Tarea encargada de acceso a la memoria FLASH

La tarea encargada de gestionar la memoria se encuentra sobre una capa de abstracción del sistema de archivos, a su vez sobre una capa de abstracción de acceso al hardware y sobre la biblioteca de funciones proporcionada por el fabricante.

La función que ejecuta la tarea es FSTaskFunc. Nada más ejecutarse inicializa el componente hardware del sistema de archivos, que básicamente consiste en realizar una función de *reset*, creación y montaje de un sistema de archivos sobre el sector especificado en los archivos de configuración. A continuación ejecuta permanentemente la tarea que consume datos de la cola de mensajes y los inserta en un fichero.

Capítulo 6

# Resultados

En este capítulo se evaluarán los resultados obtenidos y se valorará el grado de cumplimiento de los requisitos funcionales y no funcionales especificados en el capítulo 2 y en el punto 5.1.

A continuación voy a listar los requisitos funcionales del sistemas y voy a evaluar el grado de cumplimiento:

- FLASH Driver: Este componente está desarrollado en los archivos *diskio.h* y *diskio.c*. Forma parte de una capa de abstracción del hardware para completar el sistema de archivos. Tiene funciones de inicialización (*disk_initialize*), reset(*disk_ioctl(CTRL_ERASE_SECTOR)*), lectura (*disk_read*), escritura (*disk_write*) y borrado (se puede borrar mediante la función de reset o escribiendo encima de los datos viejos).
  **Grado de cumplimiento**: Total.
- FS Manager: Este componente está implementado en los archivos *fileSystem.h* y *fileSystem.h*. Este componente se trata de la capa de gestión del sistema de archivos y depende de la capa definida como FLASH Driver. Tiene funciones de inicialización (*f_mount*), reset (*f_mkfs* y *reset_sector*), apertura (*f_open*) y cierre de archivos (*f_close*), lectura (*f_read*) y escritura (*f_write*) sobre archivos y creación de archivos (*f_open* con permiso de creación).
  **Grado de cumplimiento**: Total.
- LOG Module: Este componente está desarrollado como una tarea del sistema operativo en los archivos *fs_task.h* y *fs_task.c*. Como acabo de mencionar, este componente se trata de una tarea del sistema operativo y delega en este su creación, eliminación y reseteo. Las funciones de creación, eliminación, lectura y escritura de archivos se realizan utilizando las funciones de la capa del sistema de archivos descrita anteriormente.
  **Grado de cumplimiento**: Total.
- GPS Driver: Esta capa de abstracción se encuentra definida en el archivo *BA_Board.h* y *BA_Board.c*. En estos archivos se definen los métodos de inicialización y lectura de datos de todos los periféricos del sistema, incluyendo el dispositivo GPS.

**Grado de cumplimiento**: Total.

- GPS Manager: Este componente está desarrollado como una tarea del sistema operativo en los archivos *gps_task.h* y *gps_task.c*. Como acabo de mencionar, este componente se trata de una tarea del sistema operativo y delega en este su creación, eliminación y reseteo. Las funciones de lectura de datos del GPS está implementada como *parser_gps* y dentro de la misma se mandan los mensajes leídos a una cola de mensajes de la que lee la tarea LOG Module.
  **Grado de cumplimiento**: Total.

Sobre los requisitos no funcionales especificados en el punto 5.1 considero que están cumplidos ya que se ha desarrollado el sistema dividido en diferentes capas de abstracción y la robustez del sistema operativo elegido está más que probada gracias a la multitud de proyectos que lo han utilizado y que aún se mantiene y mejora con nuevas versiones.

Sobre los requisitos no funcionales especificados en el capítulo 2 considero que también están cumplidos ya que:

- el sistema es fácilmente extensible, ya que solo basta con utilizar las bibliotecas de funciones desarrolladas o implementar unas si la funcionalidad es diferente;
- es robusto, ya que de no recibir datos del GPS se quedaría el sistema en espera y de llenarse el sistema de archivos automáticamente se crearía uno nuevo en otra parte de la memoria;
- es flexible, ya que la funcionalidad desarrollada no impide que se puedan desarrollar nuevas o mejorarlas;
- permite recuperación ante errores, ya que a la tarea del GPS no le afecta en absoluto porque no modifica la configuración del hardware y la tarea del Sistema de Archivos es capaz de leer correctamente un sistema de archivos almacenado previamente;
- y es adaptable a otras plataformas gracias a su diseño en varias capas de abstracción, siendo posible utilizarlo en otra plataforma cambiando únicamente la capa de abstracción del hardware.

Capítulo 7

# Conclusiones

En este TFG se ha desarrollado un Sistema de Logging Geoposicional en Tiempo Real para sistemas Empotrados, consistente en dos tareas que leen datos del GPS y los insertan en memoria respectivamente.

El primero de los retos afrontados ha sido la instalación y configuración de las herramientas de desarrollo, debido a que no existen herramientas que integren todas las funcionalidades requeridas. Como se ha visto en el apartado 4.2 se han utilizado varias herramientas para poder trabajar con el sistema.

Como las herramientas utilizadas son independientes entre sí y desarrolladas y mantenidas por equipos diferentes surgen diversas incompatibilidades cuando aparecen versiones nuevas, por lo que se ha tenido que comprobar cuáles eran las versiones adecuadas de cada una de las diversas herramientas.

El mayor reto afrontado, sin embargo, ha sido el tener que implementar un Sistema de Archivos de desarrollo propio para poder evitar las restricciones de la memoria FLASH del sistema.

Se puede observar que la complejidad del desarrollo del sistema de control de un sistema empotrado no viene tanto del hecho de que realice una tarea compleja sino de las restricciones inherentes al mismo, como pueden ser la capacidad de la memoria, su arquitectura física interna, o la capacidad de cómputo.

Durante este trabajo se ha visto que la utilización de sistemas ampliamente probados y aceptados en la industria, como puede ser el sistema de archivos FAT, no siempre es lo más conveniente y, de hecho, puede no ser posible aplicarlo, teniendo que diseñar alternativas más adecuadas aunque se alejen de cualquier estándar definido.

Como se ha visto las restricciones del sistema hacen único el problema y complican la reutilización del sistema desarrollado en proyectos diferentes.

El sistema desarrollado puede cambiar fácilmente el tamaño de la memoria asignado para el sistema de archivos y el sistema de logging cambiado algunos de sus parámetros de configuración. Esto es destacable ya que la granularidad temporal de los datos recogidos puede hacer inviable el sistema de logging si solo se utiliza uno de los sectores físicos de la memoria FLASH, o 128 KBytes. La memoria utilizada por el sistema de archivos es ampliable hasta la utilización de toda la memoria disponible después de almacenar el binario de la aplicación, que será aproximadamente la mitad del tamaño total.

El sistema de archivos tal y como está planteado funciona en realidad con dos particiones, para evitar la restricción de que solo se pueda borrar la memoria a nivel de sector físico completo cambiando de partición en la que escribir cuando se llena la anterior, pero si fuera necesario, y sin ningún cambio en la implementación del sistema de archivos, se podría utilizar una única partición, aunque con la restricción de que los últimos datos guardados se perderían si no los hemos exportado antes.

Tras la realización del presente TFG se ha demostrado que se han adquirido las competencias necesarias definidas por el Grado en Ingeniería Informática y la intensificación de Ingeniería de Computadores.

Daniel Aguado Araujo

Capítulo 8

# Trabajo futuro

Durante la realización de este TFG se han visto restricciones de diseño que podrían impedir el correcto funcionamiento del sistema tal y como está planteado, tomando especialmente interés las restricciones relacionadas con la memoria del sistema.

El problema principal de la memoria del sistema es la capacidad de la misma y, en menor medida, su estructura física. La capacidad de la memoria, de solo 1 MByte y del cual cerca de la mitad lo ocupa la aplicación desarrollada, puede impedir realizar un sistema de logging de otros componentes diferentes del GPS, o al menos no tener una granularidad temporal suficiente para que tenga sentido realizar el seguimiento de los datos.

Como trabajo futuro se puede ampliar este TFG para utilizar una memoria FLASH externa a través del puerto SPI del que dispone. A partir de ahí sería necesario evaluar de nuevo el diseño del sistema de archivos a utilizar, ya que al existir diferentes restricciones puede que sea mejor opción utilizar otro.

# ANEXOS

Anexo A

# Imágenes del Sistema Hardware

En la Figura A.1 se puede ver la placa para la que se ha realizado este TFG, conectada a la interfaz de programación y depuración.

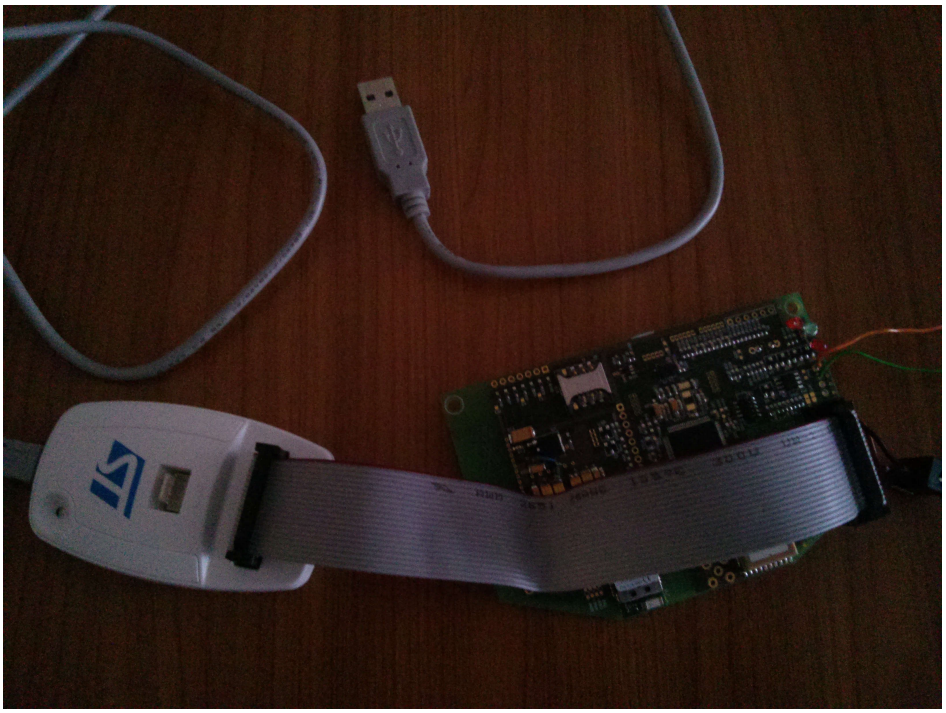


Figura A.1: Imagen que muestran todos los componentes hardware del sistema

En la Figura A.2 se puede ver un diagrama de los componentes más importantes de la placa sobre una imagen de la misma. Normalmente este tipo de diagramas se realiza sobre un dibujo de la placa, pero de esta forma queda más claro dónde está cada componente.

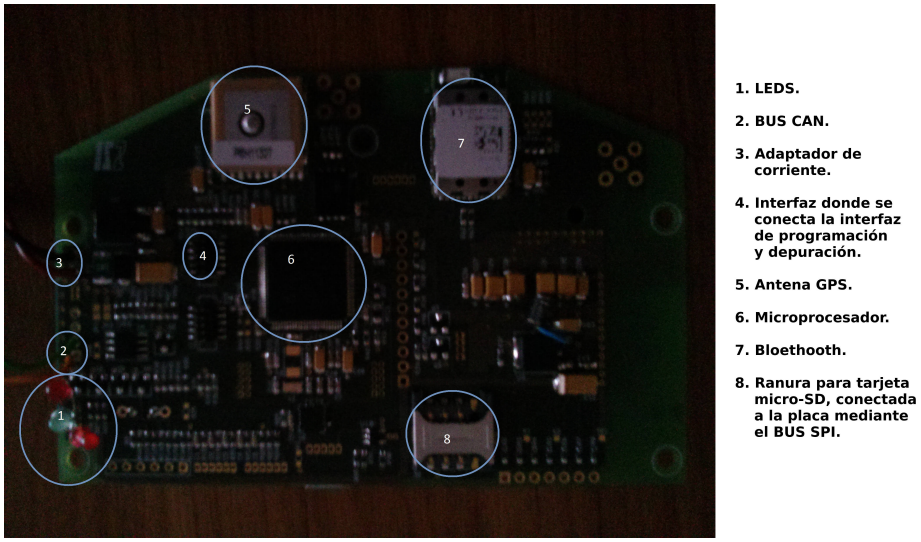



Figura A.2: Diagrama de componentes sobre una imagen de la placa

Anexo B

# Costes del proyecto

Para la realización de este proyecto se han tenido que adquirir varios componentes hardware. A continuación se listan los componentes adquiridos, al utilidad que han tenido y su coste.

- «ST-Link/v2»: este componente se conecta entre la placa donde se implementa el sistema y el PC donde se desarrolla. Este componente proporciona la interfaz de programación y depuración. Su precio a fecha del 15 de Enero de 2014 es de 25,85 € [FAR14].
- «CPU ARM Cortex M3 STMF205VGT6»: este componente es el modelo de la placa donde se implementa el sistema desarrollado. El coste de esta placa es difícil de calcular, ya que se trata de una placa modificada por IS2 del grupo POAS y no ha proporcionado el coste de la placa incluyendo componentes, integración y costes de desarrollo.

Para calcular el coste del proyecto se hizo, antes de empezar el desarrollo, una estimación en horas/hombre y coste con un salario diferente para los distintos roles que se han tenido que tomar durante el desarrollo. Los roles tomados básicamente han sido el de aprendizaje, el de arquitecto del sistema y el de desarrollador. El tiempo total estimado fue de unos tres meses a media jornada.

El salario de estos roles está por debajo del valor de mercado debido a que el desarrollador no cuenta con ninguna titulación relacionada ni con la experiencia requerida.

En la Tabla B.1 se puede ver el reparto de horas, precios y totales estimados del proyecto antes de iniciarlo.

El proyecto ha terminado desarrollándose a lo largo de unos seis meses de trabajo a media jornada. El incremento de coste es debido al descubrimiento de nuevas restricciones, que impidieron la utilización de un Sistema de Archivos comercial para el módulo de log y se tuvo

| Tareas | Horas/Hombre | €/Hora | Total (€) |
|---|---|---|---|
| 1. Revisión y estudio del proyecto existente | 40 | 8 | 320 |
| 2. Análisis y estudio del proyecto planteado | 80 | 8 | 640 |
| 3. Desarrollo del sistema | | | |
| 3.1 Diseño de la arquitectura del sistema | | | |
| 3.1.1 Diseño de la arquitectura del sistema de log | 20 | 20 | 400 |
| 3.1.2 Diseño de las comunicaciones con el dispositivo GPS | 10 | 20 | 200 |
| 3.1.3 Diseño del sistema final. De las tareas del SO y sus comunicaciones | 30 | 20 | 600 |
| 3.2 Implementación del sistema | | | |
| 3.2.1 Implementación del sistema de log | 40 | 12 | 480 |
| 3.2.2 Implementación de las comunicaciones con el dispositivo GPS | 20 | 12 | 240 |
| 3.2.3 Implementación del sistema final. De las tareas del SO y sus comunicaciones | 20 | 12 | 240 |
| 4. Documentación del proyecto | 40 | | |
| Total | 300 | | 3120 |

Tabla B.1: Coste estimado del proyecto antes de empezar su desarrollo

que implementar una alternativa de diseño propio. Otro de los motivos que ha incrementado el tiempo de desarrollo fue la inestabilidad de uno de los prototipos hardware que obligó a paralizar el desarrollo mientras la empresa responsable enviaba uno nuevo.

En la Tabla B.2 se puede ver el reparto de horas, precios y totales una vez finalizado el proyecto.

Por tanto se puede decir que el coste total del proyecto, sin contar el coste del prototipo hardware sobre el que se ha implementado el sistema, ha sido de 5785 €.

| Tareas | Horas/Hombre | €/Hora | Total (€) |
|---|---|---|---|
| 1. Revisión y estudio del proyecto existente | 40 | 8 | 320 |
| 2. Análisis y estudio del proyecto planteado | 80 | 8 | 640 |
| 3. Desarrollo del sistema | | | |
| 3.1 Diseño de la arquitectura del sistema | | | |
| 3.1.1 Diseño de la arquitectura del sistema de log | 80 | 20 | 1600 |
| 3.1.2 Diseño de las comunicaciones con el dispositivo GPS | 10 | 20 | 200 |
| 3.1.3 Diseño del sistema final. De las tareas del SO y sus comunicaciones | 30 | 20 | 600 |
| 3.2 Implementación del sistema | | | |
| 3.2.1 Implementación del sistema de log | 150 | 12 | 1800 |
| 3.2.2 Implementación de las comunicaciones con el dispositivo GPS | 20 | 12 | 240 |
| 3.2.3 Implementación del sistema final. De las tareas del SO y sus comunicaciones | 30 | 12 | 360 |
| 4. Documentación del proyecto | 80 | | |
| Total | 520 | | 5760 |

Tabla B.2: Coste real del proyecto a la hora de su finalización

Anexo C

# Formato de logs y almacenamiento

El nombre del archivo de log tiene un tamaño máximo de siete caracteres. El nombre del archivo utilizado es *log.txt*.

El formato de los mensajes almacenados en el log es el mismo que se recibe desde el dispositivo GPS. Está codificado con el estándar NMEA y se trata de una cadena de caracteres, de tamaño variable, con los datos recibidos. Se ha decidido guardar la cadena de caracteres sin ningún preprocesado debido a que en este TFG no se contempla el caso de la extracción de datos del GPS y se desconoce el formato de datos que necesitan tomar los módulos encargados de exportar los datos del log.

A continuación se encuentran los dos únicos tipos de cadenas posibles que se guardan en el log. El primero de ellos se trata de una cadena de caracteres con el tamaño y formato de un mensaje completo, el segundo se trata de una cadena de caracteres con el tamaño y formato de un mensaje sin datos geoposicionales. Los mensajes sin datos geoposicionales se producen debido a un error de sincronización con los satélites GPS, ya sea por entrar en una zona cubierta o por no haber dado suficiente tiempo de espera al iniciar el sistema.

- $GPGGA,064951.000,2307.1256,N,12016.4438,E,1,8,0.95,39.9,M,17.8,M,,*65
- $GPGGA,064951.000,,,N,,E,0,,,,M,,M,,*147

Anexo D

# Documentación del Sistema desarrollado (generada con Doxygen)

## D.1 Índice de estructura de datos

### D.1.1 Estructura de datos

Lista de estructuras con una breve descripción:



## D.2 Indice de archivos

### D.2.1 Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

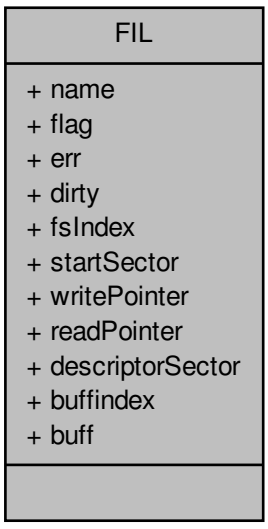
## D.3 Documentación de las estructuras de datos

### D.3.1 Referencia de la Estructura FIL

Estructura que define un archivo.

`#include <fileSystem.h>`

Diagrama de colaboración para FIL:




**Campos de datos**

- BYTE name [7]
- BYTE flag
- BYTE err

- BYTE dirty
- BYTE fsIndex
- DWORD startSector
- DWORD writePointer
- DWORD readPointer
- DWORD descriptorSector
- WORD buffindex
- BYTE buff [SECTOR_SIZE]

**Descripción detallada**

Estructura que define un archivo.

Definición en la línea 47 del archivo fileSystem.h.

**Documentación de los campos**

**BYTE FIL::buff[SECTOR_SIZE]** buffer de escritura

Definición en la línea 58 del archivo fileSystem.h.

**WORD FIL::buffindex** Índice del buffer donde se ha escrito por última vez

Definición en la línea 57 del archivo fileSystem.h.

**DWORD FIL::descriptorSector** Sector del sistema de archivos en el que se encuentra el archivo

Definición en la línea 56 del archivo fileSystem.h.

**BYTE FIL::dirty** Si hay operaciones de escritura pendientes

Definición en la línea 51 del archivo fileSystem.h.

**BYTE FIL::err** Estado de error

Definición en la línea 50 del archivo fileSystem.h.

**BYTE FIL::flag** Estado y permisos del archivo

Definición en la línea 49 del archivo fileSystem.h.

**BYTE FIL::fsIndex** Índice en el sistema de archivos

Definición en la línea 52 del archivo fileSystem.h.

**BYTE FIL::name[7]** Nombre del archivo

Definición en la línea 48 del archivo fileSystem.h.

**DWORD FIL::readPointer** Puntero de lectura. En Bytes

Definición en la línea 55 del archivo fileSystem.h.

**DWORD FIL::startSector** Sector de inicio del archivo

Definición en la línea 53 del archivo fileSystem.h.

**DWORD FIL::writePointer** Puntero de escritura del archivo. Se encuentra al final del archivo. En bytes

Definición en la línea 54 del archivo fileSystem.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

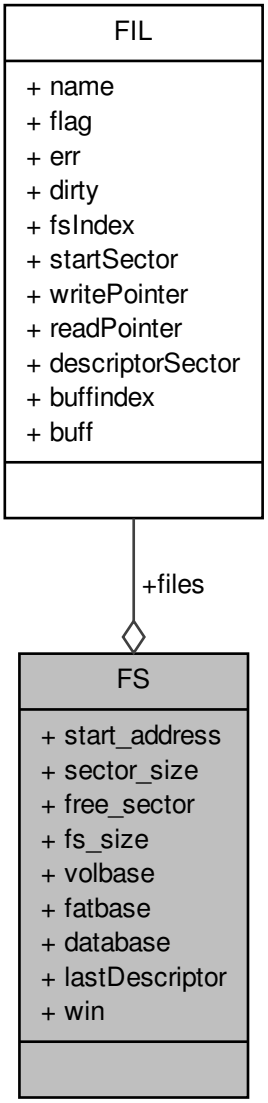
- FS/fileSystem.h

### D.3.2 Referencia de la Estructura FS

Estructura que define el Sistema de Archivos.

`#include <fileSystem.h>`

Diagrama de colaboración para FS:




**Campos de datos**

- uint32_t start_address
- BYTE sector_size
- WORD free_sector
- WORD fs_size
- WORD volbase
- WORD fatbase

- WORD database
- WORD lastDescriptor
- FIL files [_MAX_FILES]
- BYTE win [SECTOR_SIZE]

**Descripción detallada**

Estructura que define el Sistema de Archivos.

Definición en la línea 64 del archivo fileSystem.h.

**Documentación de los campos**

**WORD FS::database** Sector de inicio de la tabla de descriptores

Definición en la línea 71 del archivo fileSystem.h.

**WORD FS::fatbase** Sector de inicio del sistema de archivos

Definición en la línea 70 del archivo fileSystem.h.

**FIL FS::files[_MAX_FILES]** Vector de archivos abiertos

Definición en la línea 73 del archivo fileSystem.h.

**WORD FS::free_sector** Número de sectores libres

Definición en la línea 67 del archivo fileSystem.h.

**WORD FS::fs_size** Tamaño en sectores lógicos del sistema de archivos

Definición en la línea 68 del archivo fileSystem.h.

**WORD FS::lastDescriptor** Sector donde se guardó el último descriptor de archivo

Definición en la línea 72 del archivo fileSystem.h.

**BYTE FS::sector_size** Tamaño de cada sector lógico

Definición en la línea 66 del archivo fileSystem.h.

**uint32_t FS::start_address** Dirección de inicio del sistema de archivos

Definición en la línea 65 del archivo fileSystem.h.

**WORD FS::volbase** Sector de inicio del volúmen de datos

Definición en la línea 69 del archivo fileSystem.h.

**BYTE FS::win[SECTOR_SIZE]** Buffer de lectura del sistema de archivos y de los archivos

Definición en la línea 74 del archivo fileSystem.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

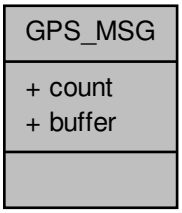
- FS/fileSystem.h

### D.3.3 Referencia de la Estructura GPS_MSG

Estructura que define el mensaje de la cola de mensaje writeQueue.

`#include <common.h>`

Diagrama de colaboración para GPS_MSG:




#### Campos de datos

- uint16_t count
- uint8_t buffer [80]

#### Descripción detallada

Estructura que define el mensaje de la cola de mensaje writeQueue.

Definición en la línea 13 del archivo common.h.

#### Documentación de los campos

**uint8_t GPS_MSG::buffer[80]** Buffer donde se almacena el mensaje

Definición en la línea 15 del archivo common.h.

**uint16_t GPS_MSG::count** Número de caracteres del mensaje enviado

Definición en la línea 14 del archivo common.h.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- src/common.h

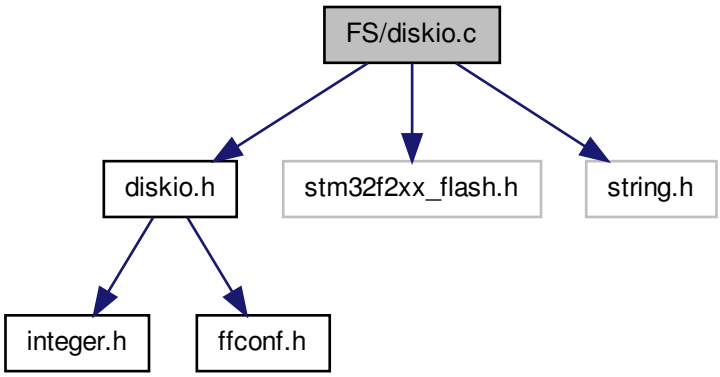
## D.4 Documentación de archivos

### D.4.1 Referencia del Archivo FS/diskio.c

Módulo que implementa las funciones del archivo diskio.h.

```
#include "diskio.h"
#include <stm32f2xx_flash.h>
#include <string.h>
```

Dependencia gráfica adjunta para diskio.c:




**'defines'**

- #define **ADDR_FLASH_SECTOR_0** ((uint32_t)0x08000000)
- #define **ADDR_FLASH_SECTOR_1** ((uint32_t)0x08004000)
- #define **ADDR_FLASH_SECTOR_2** ((uint32_t)0x08008000)
- #define **ADDR_FLASH_SECTOR_3** ((uint32_t)0x0800C000)
- #define **ADDR_FLASH_SECTOR_4** ((uint32_t)0x08010000)
- #define **ADDR_FLASH_SECTOR_5** ((uint32_t)0x08020000)
- #define **ADDR_FLASH_SECTOR_6** ((uint32_t)0x08040000)
- #define **ADDR_FLASH_SECTOR_7** ((uint32_t)0x08060000)
- #define **ADDR_FLASH_SECTOR_8** ((uint32_t)0x08080000)

- #define **ADDR_FLASH_SECTOR_9** ((uint32_t)0x080A0000)
- #define **ADDR_FLASH_SECTOR_10** ((uint32_t)0x080C0000)
- #define **ADDR_FLASH_SECTOR_11** ((uint32_t)0x080E0000)

### Funciones

- uint32_t GetSector (uint32_t Address)

  *Función que devuelve el sector al que hace referencia la dirección dada.*

- uint32_t byte_to_uint32 (const BYTE ∗src)

  *Convierte un dato BYTE en un dato uint32_t.*

- BYTE ∗ uint32_to_byte (uint32_t src, BYTE ∗dst)

  *Convierte un dato uint32_t en un dato BYTE.*

- DSTATUS get_start_end_address (int pdrv, int count, int sector, uint32_t ∗startAddress, uint32_t ∗endAddress)

  *Convierte un número de sector en una dirección física y calcula la dirección de fin a partir del contador.*

- uint32_t get_address (int pdrv, DWORD sector)

  *Convierte un número de sector en una dirección física.*

- DSTATUS disk_initialize (int pdrv)

  *Función que incializa la memoria FLASH.*

- DSTATUS flashState2FSState (FLASH_Status status)

  *Convierte el estado de operación de la flash al estado de operación del sistema de archivos.*

- DSTATUS disk_status (int pdrv)

  *Función que devuelve el estado de la FLASH.*

- DRESULT disk_read (int pdrv, BYTE ∗buff, DWORD sector, UINT count)

  *Función que lee datos desde la memoria FLASH.*

- DRESULT disk_write (int pdrv, const BYTE ∗buff, DWORD sector, UINT count)

  *Función que escribe en la memoria FLASH.*

- DRESULT disk_ioctl (int pdrv, BYTE cmd, void ∗buff)

  *Función de control de entrada/salida, varias funciones.*

### Descripción detallada

Módulo que implementa las funciones del archivo diskio.h.

Definición en el archivo diskio.c.

**Documentación de las funciones**

**uint32_t byte_to_uint32 ( const BYTE * *src* )** Convierte un dato BYTE en un dato uint32_t.

Parámetros

| | |
|---|---|
| *src* | Vector de 4 BYTES a convertir |

Devuelve

Devuelve el valor correspondiente en uint32_t

Definición en la línea 75 del archivo diskio.c.

**DSTATUS disk_initialize ( int *pdrv* )** Función que incializa la memoria FLASH.

Parámetros

| | |
|---|---|
| *pdrv* | Identificador del sector físico utilizado |

Devuelve

Código de estado de la operación

Definición en la línea 139 del archivo diskio.c.

**DRESULT disk_ioctl ( int *pdrv*, BYTE *cmd*, void * *buff* )** Función de control de entrada/salida, varias funciones.

Parámetros

| | |
|---|---|
| *pdrv* | Identificador del sector físico utilizado |
| *cmd* | Comando que especifica qué función realizar |
| *buff* | Parámetro de entrada o salida dependiendo de la función a realizar |

Devuelve

Código de estado de la operación

```
A continuación se especifican todos los comandos disponibles:
 CTRL_SYNC espera a que termine la última operación de la FLASH
 GET_SECTOR_COUNT Devuelve el número de sectores del sistema de archivos
 GET_SECTOR_SIZE Devuelve el tamaño de cada sector lógico
 GET_BLOCK_SIZE Devuelve el número de sectores físicos del sistema de archivos
 CTRL_ERASE_SECTOR Resetea el sector especificado en buff
```

Definición en la línea 262 del archivo diskio.c.

**DRESULT disk_read ( int *pdrv*, BYTE * *buff*, DWORD *sector*, UINT *count* )** Función que lee datos desde la memoria FLASH.

Parámetros

| | |
|---|---|
| *pdrv* | Identificador del sector físico utilizado |
| *buff* | Buffer donde devolver los datos leídos |
| *sector* | Número de sector lógico donde empezar a leer |
| *count* | Número de sectores lógicos a leer |

Devuelve

Código de estado de la operación

Definición en la línea 181 del archivo diskio.c.

**DSTATUS disk_status ( int *pdrv* )** Función que devuelve el estado de la FLASH.

Parámetros

| | |
|---|---|
| *pdrv* | Identificador del sector físico utilizado |

Devuelve

Estado de la FLASH

Definición en la línea 168 del archivo diskio.c.

**DRESULT disk_write ( int *pdrv*, const BYTE * *buff*, DWORD *sector*, UINT *count* )** Función que escribe en la memoria FLASH.

Parámetros

| | |
|---|---|
| *pdrv* | Identificador del sector físico utilizado |
| *buff* | Stream de datos que se van a escribir |
| *sector* | Número de sector lógico donde escribir |
| *count* | Número de sectores lógicos a escribir |

Devuelve

Código de estado de la operación

Definición en la línea 222 del archivo diskio.c.

**DSTATUS flashState2FSState ( FLASH_Status *status* )** Convierte el estado de operación de la flash al estado de operación del sistema de archivos.

Parámetros

| | |
|---|---|
| *status* | Estado de operación de la FLASH |

Devuelve

Código de estado de la operación

Definición en la línea 158 del archivo diskio.c.

**uint32_t get_address ( int *pdrv,* DWORD *sector* )** Convierte un número de sector en una dirección física.

Parámetros

| | |
|---|---|
| *pdrv* | Identificador del sistema de archivos sector Número de sector lógico de inicio |

Devuelve

Código de estado de la operación

Definición en la línea 127 del archivo diskio.c.

**DSTATUS get_start_end_address ( int *pdrv,* int *count,* int *sector,* uint32_t * *start*↩ *Address,* uint32_t * *endAddress* )** Convierte un número de sector en una dirección física y calcula la dirección de fin a partir del contador.

Parámetros

| | |
|---|---|
| *pdrv* | Identificador del sistema de archivos |
| *count* | Número de sectores lógicos de diferencia entre la dirección de inicio y de fin sector Número de sector lógico de inicio |
| *startAddress* | Puntero donde se devuelve la dirección de inicio |
| *endAddress* | Puntero donde se devuelve la dirección de fin |

Devuelve

Código de estado de la operación

Definición en la línea 107 del archivo diskio.c.

**uint32_t GetSector ( uint32_t *Address* )** Función que devuelve el sector al que hace referencia la dirección dada.

Parámetros

| | |
|---|---|
| *Address* | Dirección de memoria |

Devuelve

Sector físico al que pertenece la dirección dada

Definición en la línea 28 del archivo diskio.c.

**BYTE∗ uint32_to_byte ( uint32_t *src*, BYTE ∗ *dst* )** Convierte un dato uint32_t en un dato BYTE.

Parámetros

| | |
|---|---|
| *src* | Valor uint32_t a convertir |
| *dst* | Puntero donde se devolverá el valor |

Devuelve

Devuelve el valor correspondiente en BYTE∗

Definición en la línea 90 del archivo diskio.c.

### D.4.2 Referencia del Archivo FS/diskio.h

Cabecera de la capa de abstracción de operaciones a bajo nivel sobre la memoria.

```
#include "integer.h"
#include "ffconf.h"
```

Dependencia gráfica adjunta para diskio.h:

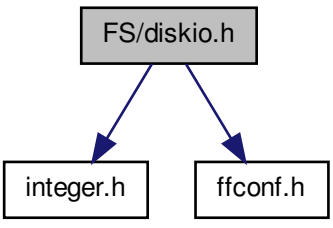
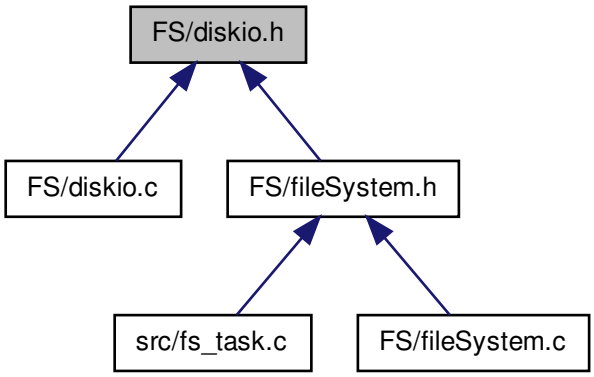


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:




**'defines'**

- #define **STA_NOINIT** 0x01 /∗ Drive not initialized ∗/
- #define **STA_NODISK** 0x02 /∗ No medium in the drive ∗/
- #define **STA_PROTECT** 0x04 /∗ Write protected ∗/
- #define **CTRL_SYNC** 0 /∗ Wait for last operation ∗/
- #define **GET_SECTOR_COUNT** 1 /∗ Get media size (for only f_mkfs()) ∗/
- #define **GET_SECTOR_SIZE** 2 /∗ Get sector size (for multiple sector size (_MA↩ X_SS >= 1024)) ∗/
- #define **GET_BLOCK_SIZE** 3 /∗ Get erase block size (for only f_mkfs()) ∗/
- #define **CTRL_ERASE_SECTOR** 4 /∗ Force erased a block of sectors (for only _↩ USE_ERASE) ∗/

**'typedefs'**

- typedef BYTE DSTATUS

**Enumeraciones**

- enum **DRESULT** {
  **RES_OK** = 0, **RES_ERROR**, **RES_WRPRT**, **RES_NOTRDY**,
  **RES_PARERR** }

**Funciones**

- DSTATUS disk_initialize (int pdrv)

*Función que incializa la memoria FLASH.*

- DSTATUS disk_status (int pdrv)

  *Función que devuelve el estado de la FLASH.*

- DRESULT disk_read (int pdrv, BYTE ∗buff, DWORD sector, UINT count)

  *Función que lee datos desde la memoria FLASH.*

- DRESULT disk_write (int pdrv, const BYTE ∗buff, DWORD sector, UINT count)

  *Función que escribe en la memoria FLASH.*

- DRESULT disk_ioctl (int pdrv, BYTE cmd, void ∗buff)

  *Función de control de entrada/salida, varias funciones.*

- uint32_t GetSector (uint32_t Address)

  *Función que devuelve el sector al que hace referencia la dirección dada.*

**Descripción detallada**

Cabecera de la capa de abstracción de operaciones a bajo nivel sobre la memoria.

Definición en el archivo diskio.h.

**Documentación de los 'typedefs'**

**typedef BYTE DSTATUS** Enumeración de los estados de operación

Definición en la línea 9 del archivo diskio.h.

**Documentación de las funciones**

**DSTATUS disk_initialize ( int *pdrv* )** Función que incializa la memoria FLASH.

Parámetros

| | |
|---|---|
| *pdrv* | Identificador del sector físico utilizado |

Devuelve

Código de estado de la operación

Definición en la línea 139 del archivo diskio.c.

**DRESULT disk_ioctl ( int *pdrv,* BYTE *cmd,* void ∗ *buff* )** Función de control de entrada/salida, varias funciones.

Parámetros

| | |
|---|---|
| *pdrv* | Identificador del sector físico utilizado |
| *cmd* | Comando que especifica qué función realizar |
| *buff* | Parámetro de entrada o salida dependiendo de la función a realizar |

Devuelve

Código de estado de la operación

```
A continuación se especifican todos los comandos disponibles:
 CTRL_SYNC espera a que termine la última operación de la FLASH
 GET_SECTOR_COUNT Devuelve el número de sectores del sistema de archivos
 GET_SECTOR_SIZE Devuelve el tamaño de cada sector lógico
 GET_BLOCK_SIZE Devuelve el número de sectores físicos del sistema de archivos
 CTRL_ERASE_SECTOR Resetea el sector especificado en buff
```

Definición en la línea 262 del archivo diskio.c.

**DRESULT disk_read ( int *pdrv*, BYTE * *buff*, DWORD *sector*, UINT *count* )** Función que lee datos desde la memoria FLASH.

Parámetros

| | |
|---|---|
| *pdrv* | Identificador del sector físico utilizado |
| *buff* | Buffer donde devolver los datos leídos |
| *sector* | Número de sector lógico donde empezar a leer |
| *count* | Número de sectores lógicos a leer |

Devuelve

Código de estado de la operación

Definición en la línea 181 del archivo diskio.c.

**DSTATUS disk_status ( int *pdrv* )** Función que devuelve el estado de la FLASH.

Parámetros

| | |
|---|---|
| *pdrv* | Identificador del sector físico utilizado |

Devuelve

Estado de la FLASH

Definición en la línea 168 del archivo diskio.c.

**DRESULT disk_write ( int *pdrv,* const BYTE * *buff,* DWORD *sector,* UINT *count* )**
Función que escribe en la memoria FLASH.

Parámetros

| | |
|---|---|
| *pdrv* | Identificador del sector físico utilizado |
| *buff* | Stream de datos que se van a escribir |
| *sector* | Número de sector lógico donde escribir |
| *count* | Número de sectores lógicos a escribir |

Devuelve

Código de estado de la operación

Definición en la línea 222 del archivo diskio.c.

**uint32_t GetSector ( uint32_t *Address* )** Función que devuelve el sector al que hace referencia la dirección dada.

Parámetros

| | |
|---|---|
| *Address* | Dirección de memoria |

Devuelve

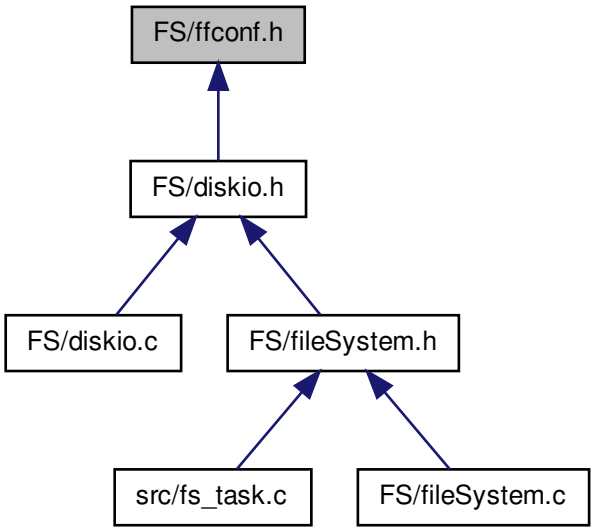
Sector físico al que pertenece la dirección dada

Definición en la línea 28 del archivo diskio.c.

### D.4.3 Referencia del Archivo FS/ffconf.h

Fichero de configuración del sistema de archivos.

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:




**'defines'**

- #define **_MAX_FILES** 3 /∗ Number of maximum files supported ∗/
- #define **SECTOR_SIZE** 16 /∗ Bytes ∗/
- #define **FS_SIZE** 8192 /∗ 128KB/16B= 8192 ∗/
- #define **MAX_FILE_SIZE** FS_SIZE/(_MAX_FILES+1)
- #define **PHYSYCAL_START_ADDRESS2** ((uint32_t)0x080A0000)
- #define **PHYSYCAL_START_ADDRESS** ((uint32_t)0x080C0000)

**'typedefs'**

- typedef unsigned long **uint32_t**

**Descripción detallada**

Fichero de configuración del sistema de archivos.

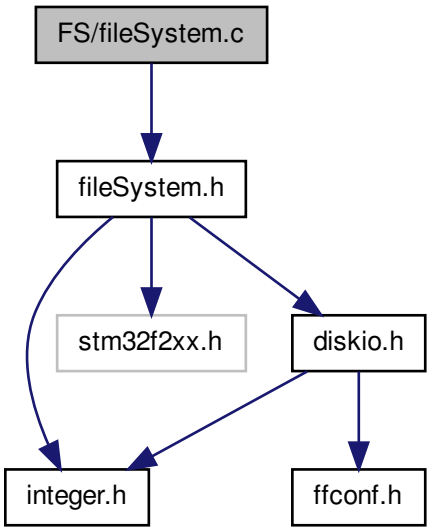
Definición en el archivo ffconf.h.

### D.4.4 Referencia del Archivo FS/fileSystem.c

Módulo que implementa las funciones del archivo fileSystem.h.

```
#include "fileSystem.h"
```

Dependencia gráfica adjunta para fileSystem.c:




**Funciones**

- uint32_t byte_2_uint32 (BYTE ∗src)

  *Convierte un dato BYTE en un dato uint32_t.*

- BYTE ∗ uint32_2_byte (uint32_t src, BYTE ∗dst)

  *Convierte un dato uint32_t en un dato BYTE.*

- FRESULT check_fs (int fsIndex)

  *Comprueba si hay un sistema de archivos en memoria.*

- FRESULT check_file (FIL ∗file)

  *Comprueba si el archivo es válido.*

- FRESULT read_file_entry (FIL ∗file, DWORD sector)

  *Lee una entrada en la tabla de descriptores de archivos y comprueba si es válida.*

- FRESULT loading_files (int fsIndex)

  *Busca todos los archivos del sistema de archivos justo después de montarlo con éxito.*

- int compare (const TCHAR ∗path, TCHAR ∗name)

  *Compara dos cadenas correspondientes al nombre de archivos.*

- int update_file (FIL ∗src, FIL ∗dst)

  *Actualiza un archivo. En la práctica crea una copia del mismo.*

- FRESULT copy_file (FIL ∗src, FIL ∗dst)

  *Copia un archivo en otro.*
- FRESULT backup_fs (int src, int dst)

  *Crea una copia del sistema de archivos en otro sistema de archivos.*
- FRESULT close_all_files ()

  *Cierra todos los archivos del sistema de archivos actual.*
- FRESULT reset_sector (int fsIndex)

  *Función que resetea un sector.*
- FRESULT f_lseek (FIL ∗fp, DWORD ofs)

  *Función que mueve el puntero de lectura de un archivo.*
- FRESULT change_sector (int opt)

  *Cambia el sector físico por defecto en el que se crea el sistema de archivos.*
- FRESULT f_open (FIL ∗fp, const TCHAR ∗path, BYTE mode)

  *Función que abre o crea un archivo.*
- FRESULT f_close (FIL ∗fp)

  *Función que cierra un archivo.*
- FRESULT f_read (FIL ∗fp, void ∗buff, UINT btr, UINT ∗br)

  *Función que lee de un archivo.*
- FRESULT f_write (FIL ∗fp, const void ∗buff, UINT btw, UINT ∗bw)

  *Función que escribe de un archivo.*
- FRESULT f_truncateStart (FIL ∗fp, DWORD ofs)

  *Función que elimina parte de un archivo.*
- FRESULT f_sync (FIL ∗fp)

  *Función que sincroniza un archivo, escribe en memoria los datos que queden pendientes.*
- FRESULT f_getfree (int fsIndex, UINT ∗mnfs)

  *Función que devuelve el número de sectores libres del sistema de archivos.*
- FRESULT f_mount (FS ∗fileSystem, int fsIndex, BYTE opt)

  *Función que crea la estructura necesaria para crear un sistema de archivos.*
- FRESULT f_mkfs (int fsIndex)

  *Función que crea un sistema de archivos.*

**Descripción detallada**

Módulo que implementa las funciones del archivo fileSystem.h.

Definición en el archivo fileSystem.c.

**Documentación de las funciones**

**FRESULT backup_fs ( int *src*, int *dst* )** Crea una copia del sistema de archivos en otro sistema de archivos.

Parámetros

| | |
|---|---|
| *src* | Identificador del sistema de archivos fuente |
| *dst* | Identificador del sistema de archivos destino |

Devuelve

Código de estado de la operación

Definición en la línea 227 del archivo fileSystem.c.

**uint32_t byte_2_uint32 ( BYTE * *src* )** Convierte un dato BYTE en un dato uint32_t.

Parámetros

| | |
|---|---|
| *src* | Vector de 4 BYTES a convertir |

Devuelve

Devuelve el valor correspondiente en uint32_t

Definición en la línea 15 del archivo fileSystem.c.

**FRESULT change_sector ( int *opt* )** Cambia el sector físico por defecto en el que se crea el sistema de archivos.

Parámetros

| | |
|---|---|
| *opt* | Opción que indica si queremos, o no, har un backup de los archivos de un sistema en el otro. Un 1 indica que queremos hacer backup. |

Devuelve

Código de estado de la operación

Definición en la línea 279 del archivo fileSystem.c.

**FRESULT check_file ( FIL * *file* )** Comprueba si el archivo es válido.

Parámetros

| | |
|---|---|
| *file* | Puntero del archivo objetivo |

Devuelve

Código de estado de la operación

Definición en la línea 75 del archivo fileSystem.c.

**FRESULT check_fs ( int *fsIndex* )** Comprueba si hay un sistema de archivos en memoria.

Parámetros

| | |
|---|---|
| *fsIndex* | Índice del sistema de archivos objetivo |

Devuelve

Código de estado de la operación

Definición en la línea 50 del archivo fileSystem.c.

**FRESULT close_all_files ( )** Cierra todos los archivos del sistema de archivos actual.

Devuelve

Código de estado de la operación

Definición en la línea 244 del archivo fileSystem.c.

**int compare ( const TCHAR * *path*, TCHAR * *name* )** Compara dos cadenas correspondientes al nombre de archivos.

Parámetros

| | |
|---|---|
| *path* | Cadena correspondiente al nombre del archivo que se quiere abrir |
| *name* | Cadena correspondiente al nombre de uno de los archivos del sistema de archivos |

Devuelve

Código de estado de la operación

Definición en la línea 169 del archivo fileSystem.c.

**FRESULT copy_file ( FIL * *src*, FIL * *dst* )** Copia un archivo en otro.

Parámetros

| | |
|---|---|
| *src* | Archivo fuente |
| *dst* | Archivo destino |

Devuelve

Código de estado de la operación

Definición en la línea 206 del archivo fileSystem.c.

**FRESULT f_close ( FIL * *fp* )** Función que cierra un archivo.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se especifica el fichero a cerrar |

Devuelve

Código de estado de la operación

Definición en la línea 353 del archivo fileSystem.c.

**FRESULT f_getfree ( int *fsIndex*, UINT * *nclst* )** Función que devuelve el número de sectores libres del sistema de archivos.

Parámetros

| | |
|---|---|
| *fsIndex* | Identificador del sistema de archivos |
| *nclst* | Puntero donde se devuelve el valor |

Devuelve

Código de estado de la operación

Definición en la línea 504 del archivo fileSystem.c.

**FRESULT f_lseek ( FIL * *fp*, DWORD *ofs* )** Función que mueve el puntero de lectura de un archivo.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se especifica el fichero objetivo |
| *ofs* | Cantidad de bytes a mover desde la posición actual del puntero de lectura |

Devuelve

Código de estado de la operación

Definición en la línea 266 del archivo fileSystem.c.

**FRESULT f_mkfs ( int *fsIndex* )** Función que crea un sistema de archivos.

Parámetros

| | |
|---|---|
| *fsIndex* | Identificador del sistema de archivos a crear |

Devuelve

Código de estado de la operación

Definición en la línea 561 del archivo fileSystem.c.

**FRESULT f_mount ( FS * *fileSystem,* int *fsIndex,* BYTE *opt* )** Función que crea la estructura necesaria para crear un sistema de archivos.

Parámetros

| | |
|---|---|
| *fileSystem* | Puntero de Sistema de Archivos donde devolver el valor |
| *fsIndex* | Identificador del sistema de archivos objetivo |
| *opt* | Acepta dos opciones: 0, sólo crea la estructura necesaria para crear un sistema de archivos. 1, intenta montar un sistema de archivos existente |

Devuelve

Código de estado de la operación

Definición en la línea 529 del archivo fileSystem.c.

**FRESULT f_open ( FIL * *fp,* const TCHAR * *path,* BYTE *mode* )** Función que abre o crea un archivo.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se devolverá el puntero del archivo abierto |
| *path* | Nombre del archivo que queremos abrir |
| *mode* | Permisos de apertura del archivo |

Devuelve

Código de estado de la operación

Definición en la línea 305 del archivo fileSystem.c.

**FRESULT f_read ( FIL * *fp*, void * *buff*, UINT *btr*, UINT * *br* )** Función que lee de un archivo.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se especifica el fichero de donde leer |
| *buff* | Buffer donde se devolverán los datos leídos |
| *btr* | Dirección de donde empezar a leer |
| *br* | Cantidad de bytes a leer |

Devuelve

Código de estado de la operación

Definición en la línea 405 del archivo fileSystem.c.

**FRESULT f_sync ( FIL * *fp* )** Función que sincroniza un archivo, escribe en memoria los datos que queden pendientes.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se especifica el fichero objetivo |

Devuelve

Código de estado de la operación

Definición en la línea 490 del archivo fileSystem.c.

**FRESULT f_truncateStart ( FIL * *fp*, DWORD *ofs* )** Función que elimina parte de un archivo.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se especifica el fichero objetivo |
| *ofs* | Cantidad de bytes a eliminar desde la posición cero del archivo |

Devuelve

Código de estado de la operación

Definición en la línea 473 del archivo fileSystem.c.

**FRESULT f_write ( FIL * *fp,* const void * *buff,* UINT *btw,* UINT * *bw* )** Función que escribe de un archivo.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se especifica el fichero de donde leer |
| *buff* | Stream de datos que se quieren escribir |
| *btw* | Dirección de donde empezar a escribir |
| *bw* | Cantidad de bytes a leer |

Devuelve

Código de estado de la operación

Definición en la línea 435 del archivo fileSystem.c.

**FRESULT loading_files ( int *fsIndex* )** Busca todos los archivos del sistema de archivos justo después de montarlo con éxito.

Parámetros

| | |
|---|---|
| *fsIndex* | Identificador del sistema de archivos |

Devuelve

Código de estado de la operación

Definición en la línea 135 del archivo fileSystem.c.

**FRESULT read_file_entry ( FIL * *file,* DWORD *sector* )** Lee una entrada en la tabla de descriptores de archivos y comprueba si es válida.

Parámetros

| | |
|---|---|
| *file* | Puntero de archivo en el que se devolverá el archivo si es válido |
| *sector* | Número de sector donde leer la entrada de la tabla de descriptores |

Devuelve

Código de estado de la operación

Definición en la línea 118 del archivo fileSystem.c.

**FRESULT reset_sector ( int *fsIndex* )** Función que resetea un sector.

Parámetros

| | |
|---|---|
| *fsIndex* | Identificador del sistema de archivos sobre el que resetear sus sectores |

Devuelve

Código de estado de la operación

Definición en la línea 253 del archivo fileSystem.c.

**BYTE∗ uint32_2_byte ( uint32_t *src*, BYTE ∗ *dst* )** Convierte un dato uint32_t en un dato BYTE.

Parámetros

| | |
|---|---|
| *src* | Valor uint32_t a convertir |
| *dst* | Puntero donde se devolverá el valor |

Devuelve

Devuelve el valor correspondiente en BYTE∗

Definición en la línea 36 del archivo fileSystem.c.

**int update_file ( FIL ∗ *src*, FIL ∗ *dst* )** Actualiza un archivo. En la práctica crea una copia del mismo.

Parámetros

| | |
|---|---|
| *src* | Archivo fuente |
| *dst* | Archivo destino |

Devuelve

Código de estado de la operación

Definición en la línea 184 del archivo fileSystem.c.

### D.4.5 Referencia del Archivo FS/fileSystem.h

Cabecera principal del módulo del Sistema de Archivos.

```
  #include "integer.h"
#include "stm32f2xx.h"
#include "diskio.h"
```

Dependencia gráfica adjunta para fileSystem.h:

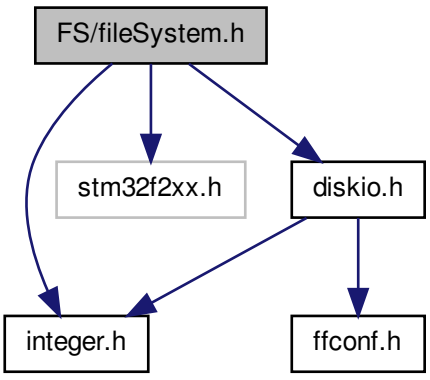
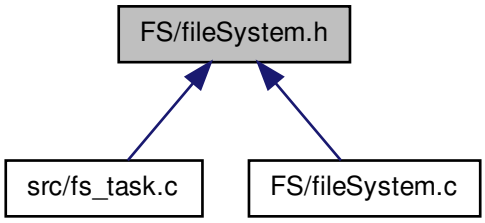


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:




**Estructuras de datos**

- struct FIL

  *Estructura que define un archivo.*

- struct FS

  *Estructura que define el Sistema de Archivos.*

**'defines'**

- #define **FA_READ** 0x01
- #define **FA_OPEN_EXISTING** 0x00
- #define **FA_WRITE** 0x02
- #define **FA_CREATE_NEW** 0x04
- #define **FA_CREATE_ALWAYS** 0x08
- #define **FA_OPEN_ALWAYS** 0x10

**Enumeraciones**

- enum FRESULT {
  FR_OK = 0, FR_DISK_ERR, FR_INT_ERR, FR_NOT_READY,
  FR_NO_FILE, FR_NO_PATH, FR_INVALID_NAME, FR_DENIED,
  FR_EXIST, FR_INVALID_OBJECT, FR_WRITE_PROTECTED, FR_INVALID_-
  DRIVE,
  FR_NOT_ENABLED, FR_NO_FILESYSTEM, FR_MKFS_ABORTED, FR_TIM↩
  EOUT,
  FR_LOCKED, FR_NOT_ENOUGH_CORE, FR_TOO_MANY_FILES, FR_INVA↩
  LID_PARAMETER }

  *Enumeración de los estados de operación de la memoria FLASH.*

**Funciones**

- FRESULT f_open (FIL ∗fp, const TCHAR ∗path, BYTE mode)

  *Función que abre o crea un archivo.*

- FRESULT f_close (FIL ∗fp)

  *Función que cierra un archivo.*

- FRESULT f_read (FIL ∗fp, void ∗buff, UINT btr, UINT ∗br)

  *Función que lee de un archivo.*

- FRESULT f_write (FIL ∗fp, const void ∗buff, UINT btw, UINT ∗bw)

  *Función que escribe de un archivo.*

- FRESULT f_truncateStart (FIL ∗fp, DWORD ofs)

  *Función que elimina parte de un archivo.*

- FRESULT f_lseek (FIL ∗fp, DWORD ofs)

  *Función que mueve el puntero de lectura de un archivo.*

- FRESULT f_sync (FIL ∗fp)

  *Función que sincroniza un archivo, escribe en memoria los datos que queden pendientes.*

- FRESULT f_getfree (int fsIndex, UINT ∗nclst)

  *Función que devuelve el número de sectores libres del sistema de archivos.*

- FRESULT f_mount (FS ∗fileSystem, int fsIndex, BYTE opt)

  *Función que crea la estructura necesaria para crear un sistema de archivos.*

- FRESULT f_mkfs (int fsIndex)

  *Función que crea un sistema de archivos.*

- FRESULT reset_sector (int fsIndex)

  *Función que resetea un sector.*

**Descripción detallada**

Cabecera principal del módulo del Sistema de Archivos.

Definición en el archivo fileSystem.h.

**Documentación de las enumeraciones**

**enum FRESULT** Enumeración de los estados de operación de la memoria FLASH.

Valores de enumeraciones

***FR_OK*** (0) Éxito

***FR_DISK_ERR*** (1) Ha ocurrido un error en la capa de abstracción de E/S

***FR_INT_ERR*** (2) Parámetros erróneos

***FR_NOT_READY*** (3) Memoria ocupada

***FR_NO_FILE*** (4) No se encuentra el archivo

***FR_NO_PATH*** (5) No se encuentra la ruta

***FR_INVALID_NAME*** (6) El nombre o la ruta es inválido

***FR_DENIED*** (7) Acceso denegado o prohibido el acceso al directorio

***FR_EXIST*** (8) Acceso denegado o prohibido el acceso

***FR_INVALID_OBJECT*** (9) El archivo o directorio es inválido

***FR_WRITE_PROTECTED*** (10) La memoria está protegida contra escrituras

***FR_INVALID_DRIVE*** (11) El identificador del medio físco es inválido

***FR_NOT_ENABLED*** (12) El volúmen no tiene área de trabajo

***FR_NO_FILESYSTEM*** (13) No hay un Sistema de Archivos válido

***FR_MKFS_ABORTED*** (14) Abortada la creación de un Sistema de Archivos debido a parámetros erróneos

***FR_TIMEOUT*** (15) Tiempo de espera excedido

***FR_LOCKED*** (16) La operación ha sido rechazada debido a las políticas de acceso

***FR_NOT_ENOUGH_CORE*** (17) No hay espacio disponible para guardar

***FR_TOO_MANY_FILES*** (18) Número de archivos máximo excedido

***FR_INVALID_PARAMETER*** (19) El parámetro dado es inválido

Definición en la línea 22 del archivo fileSystem.h.

**Documentación de las funciones**

**FRESULT f_close ( FIL ∗ *fp* )** Función que cierra un archivo.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se especifica el fichero a cerrar |

Devuelve

Código de estado de la operación

Definición en la línea 353 del archivo fileSystem.c.

**FRESULT f_getfree ( int *fsIndex*, UINT ∗ *nclst* )** Función que devuelve el número de sectores libres del sistema de archivos.

Parámetros

| | |
|---|---|
| *fsIndex* | Identificador del sistema de archivos |
| *nclst* | Puntero donde se devuelve el valor |

Devuelve

Código de estado de la operación

Definición en la línea 504 del archivo fileSystem.c.

**FRESULT f_lseek ( FIL * *fp*, DWORD *ofs* )** Función que mueve el puntero de lectura de un archivo.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se especifica el fichero objetivo |
| *ofs* | Cantidad de bytes a mover desde la posición actual del puntero de lectura |

Devuelve

Código de estado de la operación

Definición en la línea 266 del archivo fileSystem.c.

**FRESULT f_mkfs ( int *fsIndex* )** Función que crea un sistema de archivos.

Parámetros

| | |
|---|---|
| *fsIndex* | Identificador del sistema de archivos a crear |

Devuelve

Código de estado de la operación

Definición en la línea 561 del archivo fileSystem.c.

**FRESULT f_mount ( FS * *fileSystem*, int *fsIndex*, BYTE *opt* )** Función que crea la estructura necesaria para crear un sistema de archivos.

Parámetros

| | |
|---|---|
| *fileSystem* | Puntero de Sistema de Archivos donde devolver el valor |
| *fsIndex* | Identificador del sistema de archivos objetivo |
| *opt* | Acepta dos opciones: 0, sólo crea la estructura necesaria para crear un sistema de archivos. 1, intenta montar un sistema de archivos existente |

Devuelve

Código de estado de la operación

Definición en la línea 529 del archivo fileSystem.c.

**FRESULT f_open ( FIL * *fp*, const TCHAR * *path*, BYTE *mode* )** Función que abre o crea un archivo.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se devolverá el puntero del archivo abierto |
| *path* | Nombre del archivo que queremos abrir |
| *mode* | Permisos de apertura del archivo |

Devuelve

Código de estado de la operación

Definición en la línea 305 del archivo fileSystem.c.

**FRESULT f_read ( FIL * *fp*, void * *buff*, UINT *btr*, UINT * *br* )** Función que lee de un archivo.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se especifica el fichero de donde leer |
| *buff* | Buffer donde se devolverán los datos leídos |
| *btr* | Dirección de donde empezar a leer |
| *br* | Cantidad de bytes a leer |

Devuelve

Código de estado de la operación

Definición en la línea 405 del archivo fileSystem.c.

**FRESULT f_sync ( FIL $*$ *fp* )** Función que sincroniza un archivo, escribe en memoria los datos que queden pendientes.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se especifica el fichero objetivo |

Devuelve

Código de estado de la operación

Definición en la línea 490 del archivo fileSystem.c.

**FRESULT f_truncateStart ( FIL * *fp*, DWORD *ofs* )** Función que elimina parte de un archivo.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se especifica el fichero objetivo |
| *ofs* | Cantidad de bytes a eliminar desde la posición cero del archivo |

Devuelve

Código de estado de la operación

Definición en la línea 473 del archivo fileSystem.c.

**FRESULT f_write ( FIL * *fp*, const void * *buff*, UINT *btw*, UINT * *bw* )** Función que escribe de un archivo.

Parámetros

| | |
|---|---|
| *fp* | Parámetro de entrada en el que se especifica el fichero de donde leer |
| *buff* | Stream de datos que se quieren escribir |
| *btw* | Dirección de donde empezar a escribir |
| *bw* | Cantidad de bytes a leer |

Devuelve

Código de estado de la operación

Definición en la línea 435 del archivo fileSystem.c.

**FRESULT reset_sector ( int *fsIndex* )** Función que resetea un sector.

Parámetros

| | |
|---|---|
| *fsIndex* | Identificador del sistema de archivos sobre el que resetear sus sectores |

Devuelve

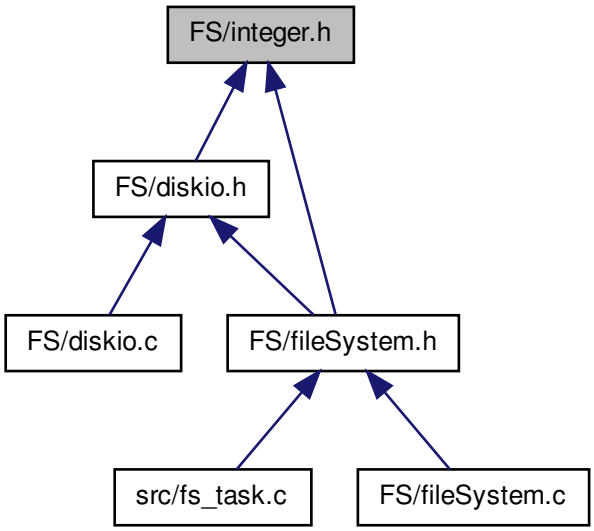
Código de estado de la operación

Definición en la línea 253 del archivo fileSystem.c.

### D.4.6 Referencia del Archivo FS/integer.h

Tipos de datos abreviados.

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:




**'typedefs'**

- typedef unsigned char **BYTE**
- typedef short **SHORT**
- typedef unsigned short **WORD**
- typedef unsigned short **WCHAR**
- typedef int **INT**
- typedef unsigned int **UINT**
- typedef long **LONG**

- typedef unsigned long **DWORD**
- typedef char **TCHAR**

**Descripción detallada**

Tipos de datos abreviados.

Definición en el archivo integer.h.

### D.4.7 Referencia del Archivo src/common.h

En este archivo se definen estructuras de datos comunes para todas las tareas.

```
#include "FreeRTOS.h"
#include "queue.h"
#include "stm32f2xx.h"
```

Dependencia gráfica adjunta para common.h:

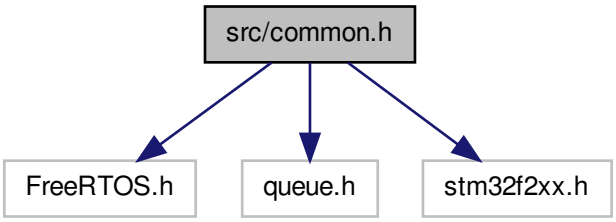
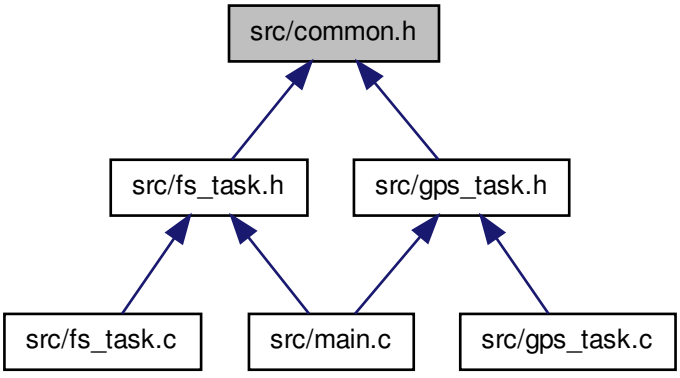



Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



### Estructuras de datos

- struct GPS_MSG

  *Estructura que define el mensaje de la cola de mensaje writeQueue.*

### Variables

- xQueueHandle **writeQueue**

### Descripción detallada

En este archivo se definen estructuras de datos comunes para todas las tareas.

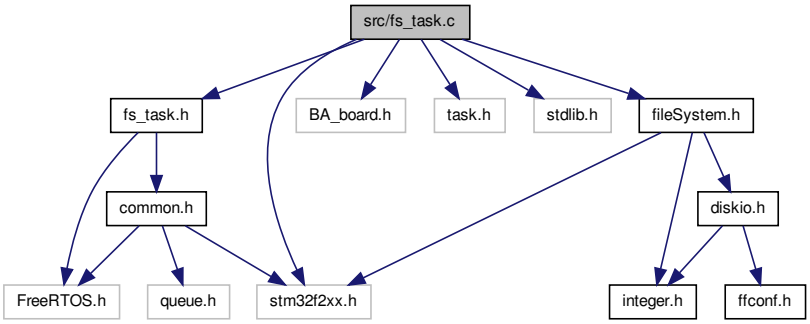
Definición en el archivo common.h.

## D.4.8 Referencia del Archivo src/fs_task.c

Módulo que implementa las funciones del archivo fs_task.h.

```
#include "fs_task.h"
#include "stm32f2xx.h"
#include "BA_board.h"
#include "task.h"
#include <stdlib.h>
#include "fileSystem.h"
```

Dependencia gráfica adjunta para fs_task.c:




### Funciones

- void FSTaskFunc (void ∗pParams)

  *Esta es la función principal de la tarea Esta función inicializa la memoria y el sistema de archivos y lanza el método que se queda a la espera de mensajes desde el GPS.*

- void FSHardwareInit (void ∗pParam)

*Método que inicializa el hardware de la tarea Esta función resetea un sector y crea en él un sistema de archivos.*

- void FSStartTask (unsigned short nStackDepth, unsigned portBASE_TYPE nPriority, void ∗pParams)

  *Método que crea la tarea.*

- void read_file ()

  *Función que lee de un archivo Esta función no se utiliza en la implementación actual, pero se mantiene para poder ser utilizada cuando se integre en el proyecto Biker Assistant.*

- void ReceiveWriteGPS ()

  *Función que escribe en un archivo todos los mensajes recibidos desde la tarea que controla el GPS Esta función, dentro del bucle de ejecución, enciende y apaga un led cada vez que recibe y escribe un dato.*

**Descripción detallada**

Módulo que implementa las funciones del archivo fs_task.h.

Definición en el archivo fs_task.c.

**Documentación de las funciones**

**void FSStartTask ( unsigned short *nStackDepth*, unsigned portBASE_TYPE *nPriority*, void ∗ *pParams* )** Método que crea la tarea.

Parámetros

| | |
|---|---|
| *sStackDepth* | Tamaño de la pila de memoria |
| *nPrioriy* | Prioridad de la tarea |

Definición en la línea 27 del archivo fs_task.c.

### D.4.9 Referencia del Archivo src/fs_task.h

Cabecera de la tarea que gestiona el sistema de archivos.

```
  #include "FreeRTOS.h"
#include "common.h"
```

Dependencia gráfica adjunta para fs_task.h:

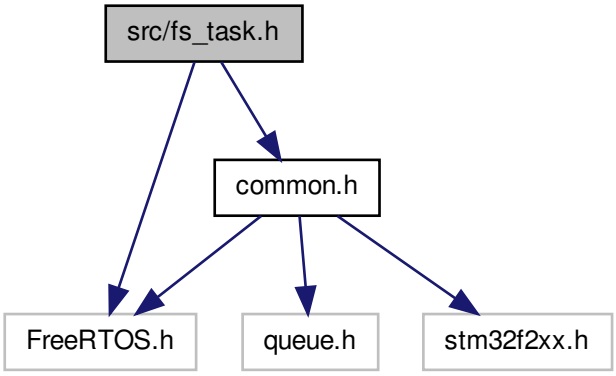
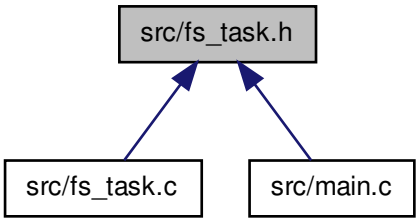



Gráfico de los archivos que directa o indirectamente incluyen a este archivo:




### Funciones

- void FSHardwareInit (void ∗pParam)

  *Método que inicializa el hardware de la tarea Esta función resetea un sector y crea en él un sistema de archivos.*

- void FSStartTask (unsigned short nStackDepth, unsigned portBASE_TYPE nPriority, void ∗pParams)

  *Método que crea la tarea.*

### Descripción detallada

Cabecera de la tarea que gestiona el sistema de archivos.

Definición en el archivo fs_task.h.

### Documentación de las funciones

**void FSStartTask ( unsigned short *nStackDepth*, unsigned portBASE_TYPE *nPriority*, void * *pParams* )** Método que crea la tarea.

Parámetros

| | |
|---|---|
| *sStackDepth* | Tamaño de la pila de memoria |
| *nPrioriy* | Prioridad de la tarea |

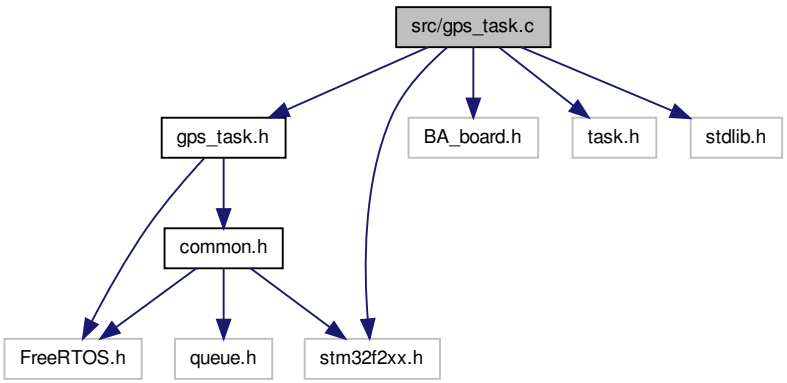
Definición en la línea 27 del archivo fs_task.c.

## D.4.10 Referencia del Archivo src/gps_task.c

Módulo que implementa las funciones del archivo gps_task.h.

```
#include "gps_task.h"
#include "stm32f2xx.h"
#include "BA_board.h"
#include "task.h"
#include <stdlib.h>
```

Dependencia gráfica adjunta para gps_task.c:




### Funciones

- void GPSTaskFunc (void *pParams)

  *Esta es la función principal de la tarea Esta función inicializa el GPS y lanza el método que lee datos del GPS y se los manda a la tarea que gestiona el Sistema de Archivos.*

- void setupGPS ()

*Esta es la función que inicializa el GPS.*

- void GPSHardwareInit (void ∗pParam)

  *Esta es la función que inicializa todo el hardware que necesita la tarea.*

- void GPSStartTask (unsigned short nStackDepth, unsigned portBASE_TYPE nPriority, void ∗pParams)

  *Método que crea la tarea.*

- void parser_GPS ()

  *Esta es la función que lee datos del GPS y, cuando detecta un mensaje correcto, lo envía a la tarea que gestiona el Sistema de Archivos Esta tarea enciende un led cada vez que envía correctamente un mensaje.*

**Descripción detallada**

Módulo que implementa las funciones del archivo gps_task.h.

Definición en el archivo gps_task.c.

**Documentación de las funciones**

**void GPSHardwareInit ( void ∗ *pParam* )** Esta es la función que inicializa todo el hardware que necesita la tarea.

Método que inicializa el hardware de la tarea.

Definición en la línea 39 del archivo gps_task.c.

**void GPSStartTask ( unsigned short *nStackDepth*, unsigned portBASE_TYPE *nPriority*, void ∗ *pParams* )** Método que crea la tarea.

Parámetros

| | |
|---|---|
| *sStackDepth* | Tamaño de la pila de memoria |
| *nPrioriy* | Prioridad de la tarea |

Definición en la línea 43 del archivo gps_task.c.

### D.4.11 Referencia del Archivo src/gps_task.h

Cabecera de la tarea que gestiona el GPS.

```
  #include "FreeRTOS.h"
#include "common.h"
```

Dependencia gráfica adjunta para gps_task.h:

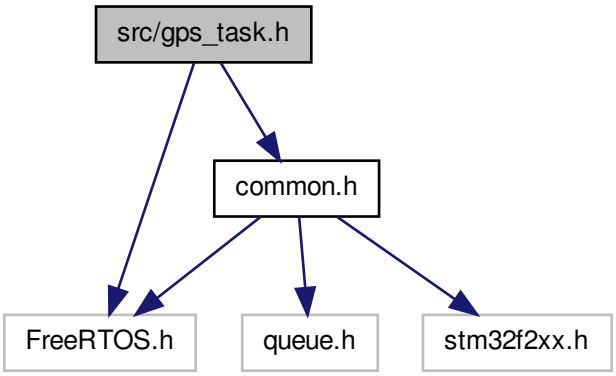
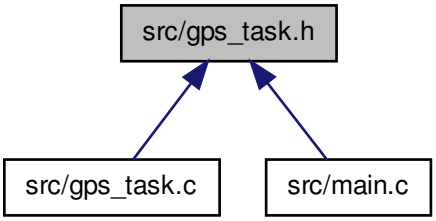



Gráfico de los archivos que directa o indirectamente incluyen a este archivo:




### Funciones

- void GPSHardwareInit (void ∗pParam)

  *Método que inicializa el hardware de la tarea.*

- void GPSStartTask (unsigned short nStackDepth, unsigned portBASE_TYPE nPriority, void ∗pParams)

  *Método que crea la tarea.*

### Descripción detallada

Cabecera de la tarea que gestiona el GPS.

Definición en el archivo gps_task.h.

**Documentación de las funciones**

**void GPSHardwareInit ( void * *pParam* )** Método que inicializa el hardware de la tarea.

Método que inicializa el hardware de la tarea.

Definición en la línea 39 del archivo gps_task.c.

**void GPSStartTask ( unsigned short *nStackDepth*, unsigned portBASE_TYPE *nPriority*, void * *pParams* )** Método que crea la tarea.

Parámetros

| | |
|---|---|
| *sStackDepth* | Tamaño de la pila de memoria |
| *nPrioriy* | Prioridad de la tarea |

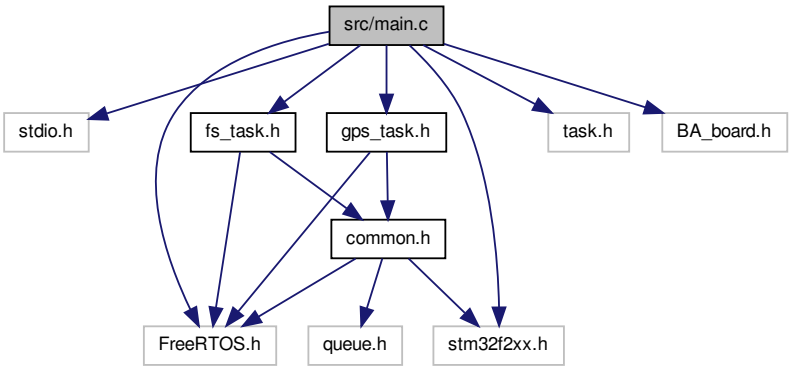
Definición en la línea 43 del archivo gps_task.c.

### D.4.12 Referencia del Archivo src/main.c

Función principal.

```
#include <stdio.h>
#include "FreeRTOS.h"
#include "task.h"
#include "stm32f2xx.h"
#include "BA_board.h"
#include "gps_task.h"
#include "fs_task.h"
```

Dependencia gráfica adjunta para main.c:



### Funciones

- void prvSetupHardware (void)

  *Función que inicializa el hardware.*
- int putChar (int ch)
- void setupTasks ()

  *Función que crea las tareas del Sistemas Operativo FreeRTOS.*
- void setupQueues ()

  *Función que crea las colas de mensajes entre tareas del Sistema Operativo FreeRTOS.*
- int main (void)

  *Función principal del sistema.*
- void vApplicationStackOverflowHook (xTaskHandle ∗pxTask, signed portCHAR ∗pc↩ TaskName)

  *Función que crea la excepción de desbordamiento de la pila de memoria.*
- void vApplicationTickHook (void)

  *Función que crea la excepción del reloj.*
- void Delay (uint32_t t)

  *Función que define la función de espera.*

### Descripción detallada

Función principal.

Definición en el archivo main.c.

### Documentación de las funciones

**void Delay ( uint32_t *t* )** Función que define la función de espera.

Parámetros

| | |
|---|---|
| *t* | Tiempo a esperar en milisegundos |

Definición en la línea 121 del archivo main.c.

**int main ( void )** Función principal del sistema.

Nota

Esta función inicializa el hardware, crea las colas de mensajes, crea las tareas e inicializa el planificador del Sistema Operativo

Definición en la línea 54 del archivo main.c.

**void prvSetupHardware ( void )** Función que inicializa el hardware.

Nota

Esta función inicializa el vector de interrupciones, configura los niveles de priporidad, establece la fuente del reloj del sistema e inicializa los leds

Definición en la línea 72 del archivo main.c.

**int putChar ( int *ch* )** External dependence needed by printf implementation. Write a character to standard out.

Parámetros

| | |
|---|---|
| *c* | Specifies the character to be written. |

Devuelve

Returns the character written. No error conditions are managed.

Definición en la línea 112 del archivo main.c.

**void vApplicationStackOverflowHook ( xTaskHandle * *pxTask*, signed portCHAR * *pcTaskName* )** Función que crea la excepción de desbordamiento de la pila de memoria.

Nota

Esta función la llama internamente el Sistema Operativo

Definición en la línea 90 del archivo main.c.

**void vApplicationTickHook ( void )** Función que crea la excepción del reloj.

Nota

Esta función la llama internamente el Sistema Operativo

Definición en la línea 106 del archivo main.c.

# Referencias


[And14] Android. Página oficial del Sistema Operativo Android, Abril 2014. url: http://www.android.com/.

[App14] Apple. Página oficial del Sistema Operativo iOS de Apple, Abril 2014. url: http://www.apple.com/es/ios/.

[Bec02] K. Beck. *Test-Driven Development: By Example*. Addison Wesley, 2002.

[Ble10] C. Ble. Diseno Agil con TDD, 2010.

[Chi14] ChibiOS. Página oficial de la documentación del Sistema Operativo ChibiOS, Marzo 2014. url: http://www.chibios.org/dokuwiki/doku.php.

[ecO14] ecOS. Página oficial de la documentación del Sistema Operativo ecOS, Marzo 2014. url: http://ecos.sourceware.org/.

[FAR14] FARNELL. STMICROELECTRONICS - ST-LINK/V2 - ICD/PROGRAMMER, FOR STM8, STM32, Enero 2014. url: http://es.farnell.com/jsp/displayProduct.jsp?sku=1892523&gross_price=true&CMP=KNC-GES-FES-GEN-LISTINGS&mckv=RFE1qR8W|pcrid|productlistings.

[Fre14] FreeRTOS. Página oficial de FreeRTOS, Abril 2014. url: http://www.freertos.org/.

[GAGS10] D. Gajsky, S. Abdi, A. Gerslauer, y G. Schirner. *Embedded System Desing: Modelling, Synthesis and Verification*. Springer, 2010.

[Glo13a] GlobalTop. GlobalTop-FGPMMOPA6H-Datasheet-V0A. Datasheet, GlobalTop, 2013.

[Glo13b] GlobalTop. PMTK command packet-Complete-A11. Technical report, GlobalTop, 2013.

[HG13] R. Panthaleon H. Geelen, A. Hoogt. Embedded Systems Roadmap 2014. *HTSM*, December 2013.

[INT13] INTEL. Intel® X18-M/X25-M SATA Solid State Drive. Datasheet, INTEL, 2013.

[JFF14] JFFS2. JFFS2: The Journalling Flash File System, version 2, Mayo 2014. url: https://sourceware.org/jffs2/.

[Lin14a] Linux. Página oficial de la documentación del Sistema de Archivos ext4, Abril 2014. url: https://ext4.wiki.kernel.org/index.php/Main_Page.

[Lin14b] Linux. Página oficial del Sistema Operativo Linux, Abril 2014. url: http://www.linux.org/.

[Mic13a] ST Microelectronics. STM32F205VG Datasheet. Datasheet, ST Microelectronics, 2013.

[Mic13b] ST Microelectronics. STM32F20x LibDescription. Technical report, ST Microelectronics, 2013.

[Mic13c] ST Microelectronics. STMF205 FlashProgramming. Datasheet, ST Microelectronics, 2013.

[Mic14a] Microsoft. Página oficial de la descripción de FAT32, Abril 2014. url: http://support.microsoft.com/kb/154997.

[Mic14b] Microsoft. Página oficial describiendo los sistemas de archivos FAT, HPFS y NTFS, Abril 2014. url: http://support.microsoft.com/kb/100108.

[MIT09] MIT. AeroAstro Magazine Highlight, Marzo 2009. url: http://web.mit.edu/aeroastro/news/magazine/aeroastro6/mit-apollo.html.

[NME08] NMEA. NMEA0183 Standard. Standard, NMEA, 2008. url: http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp.

[rom14] romfs. romfs, Mayo 2014. url: http://romfs.sourceforge.net/.

[squ14] squashfs. squashfs, Mayo 2014. url: http://squashfs.sourceforge.net/.

[Tin14] TinyOS. Página oficial de la documentación del Sistema Operativo TinyOS, Marzo 2014. url: http://www.tinyos.net/.

[TIO14] TIOBE. Index of popularity of programming languages, June 2014. url: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

[TWS06] O.Weck T. W. Simpson, T.Marion. Platform-Based Design and Development: Current Trends and Needs in Industry. En *ASME 2006 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, September 2006.

[USA14] USA. The Global Positioning System, June 2014. url: http://www.gps.gov/systems/gps/.

[Wij14] D. Wijesinghe. Embedded Operating Systems. *University of Ulster*, April 2014.

[Wir10] Wired. Android Phones Can Substitute for Supercomputers, Agosto 2010. url: http://www.wired.com/2010/08/supercomputing-app-android/.

[YAF14] YAFFS2. YAFFS2 Overview, Mayo 2014. url: http://www.yaffs.net/yaffs-overview.

Este documento fue editado y tipografiado con LaTeX
empleando la clase **esi-tfg** que se puede encontrar en:
`https://bitbucket.org/arco_group/esi-tfg`