CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Desarrollo de manejadores y pruebas de integración del computador de a bordo del microsatélite UPMSat-2

Autor: Beatriz Lacruz Alcaraz

Director: Juan Zamorano Flores

MADRID, JUNIO 2015

En muchísimas ocasiones se ha asegurado que algo es imposible, pero una o varias décadas después se ha demostrado su viabilidad.

Michio Kaku.

En primer lugar, quiero agracedecer a toda mi familia, en especial a mi madre, mi padre y mi hermano el apoyo que me han mostrado a lo largo de toda la carrera.
A Héctor, por estar a mi lado en los buenos y malos momentos durante estos años.
También quiero agradecer a mi tutor, Juan, por haberme dado la oportunidad de realizar este proyecto con él y por toda la ayuda prestada durante todo este tiempo.
Por último, agradecer a mis compañeros de laboratorio y amigos que me han acompañado durante estos años ayudándome a mejorar personal y académicamente.

Gracias a todos.

Índice de contenidos

Índice de figuras

Índice de código

Acrónimos

ADC Analog to Digital Converter. Conversor Analógico Digital.

ADCS Control y Determinación de Actitud.

AMBA Advanced Microcontroller Bus Architecture. Arquitectura de Bus de Microcontroladores Avanzados.

BSP Board Support Package.

DAS Digital to Analog Subsystem. Sistema Analógico Digital.

DB Distribution Board. Tarjeta de Distribución.

EBOX Electronic Box.

EEPROM Electrically Erasable Programmable Read-Only Memory. Memoria programable y borrable eléctricamente de sólo lectura.

ESA European Space Agency. Agencia Espacial Europea.

ESTEC European Space Research and Technology Centre. Centro de Tecnología e Investigación Espacial Europeo.

FPGA Field Programmable Gate Array.

I2C Inter-Integrated Circuit. Circuitos Inter-Integrados.

IDR Instituto Ignacio Da Riva.

KB Kilobyte.

LZSS Algoritmo Lempel–Ziv–Storer–Szymanski.

MB Megabyte.

MGM Magnetómetro.

MGT Magnetopares.

MTS Micro Thermal Switch.

OBC On-Board Computer. Computador de a bordo.

ORK Open Ravenscar Kernel.

PDU Power Distribution Unit. Unidad de distribución de la alimentación.

PSU Power Supply Unit. Unidad de fuente de alimentación.

RAM Random Access Memory. Memoria de acceso aleatorio.

RISC Reduced Instruction Set Computing. Juego de Instrucciones Reducidas o Computación de Set de Instrucciones Reducidas.

ROLEU Registro de Objetos Lanzados al Espacio Ultraterrestre.

ROM Read-Only Memory. Memoria de sólo lectura.

RW Reaction Wheel. Rueda de reacción.

SOC System On Chip

SPARC Scalable Processor ARChitecture. Arquitectura de Procesador Escalable.

SPI Serial Peripheral Interface. Interfaz Serie Periférica.

SRAM Synchronous Random Access Memory. Memoria estática de acceso aleatorio.

STRAST Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos.

UART Universal Asynchronous Receiver-Transmitter. Transmisor-Receptor Asíncrono Universal.

UPM Universidad Politécnica de Madrid.

VHDL Very High Speed Integrated Circuit. Circuito Integrado de Alta Velocidad.

WDG Watch Dog Guard.

WDT Watch Dog Timer.

Capítulo 1

Resumen

1.1. Español

Varios grupos de la Universidad Politécnica de Madrid se encuentran actualmente desarrollando un micro-satélite de experimentación bajo el proyecto UPMSat-2, sucesor de otro exitoso proyecto similar, el UPM-Sat 1. Bajo este marco la autora del presente documento ha llevado a cabo la realización de tres tareas fundamentales para hacer posible la puesta en órbita de dicho satélite.

Las tareas principales definidas como alcance de este proyecto pretenden facilitar el uso de la memoria no volátil del computador de a bordo y comprobar el funcionamiento de todos los sistemas del satélite. Por ello se ha realizado el arranque desde la memoria no volátil junto con un manejador para el uso de la misma y un conjunto de pruebas de validación del software e integración del hardware.

La satisfacción con los resultados obtenidos ha hecho posible la inclusión del software y pruebas desarrolladas al conjunto de todo el software del proyecto UPMSat-2, contribuyendo así a la capacidad del satélite para ser puesto en órbita.

1.2. English

UPMSat-2, the successor of UPM-Sat 1, is a joint project for the development of a micro-satellite for experimentation, which is being carried out by various research groups at Universidad Politécnica de Madrid. The author of this document has developed three main tasks to make possible the correct operation of this satellite during the duration of its mission.

The scope of the present work is to enable the use of the on-board computer's non-volatile memory and the development of a software to test that the satellite's subsystems are working properly. To this end, the non-volatile memory's boot sequence has been implemented together with the driver to use such memory, and a series of validation and integration tests for the software and the hardware.

The results of the this work have been satisfactory, therefore they have been included in UPMSat-2's software, contributing this way to the capacity of the satellite to carry out its mission.

Capítulo 2

Introducción

En esta memoria se encuentra una descripción del trabajo realizado por la alumna durante el desarrollo del Trabajo de Fin de Grado.

El trabajo se encuentra enmarcado dentro del proyecto UPMSat-2 [2, 11]. Este proyecto está liderado por el "Instituto Ignacio Da Riva" (IDR) y está desarrollado por varios grupos de investigación y empresas de desarrollo aeroespacial. Dentro de este proyecto, el trabajo ha sido desarrollado dentro del grupo de Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos (STRAST) perteneciente a la Universidad Politécnica de Madrid (UPM).

Este grupo se ha encargado de desarrollar el software para la misión, tanto del segmento de tierra como del software embarcado del computador de a bordo del satélite.

2.1. Antecedentes

El 7 de julio de 1995 fue lanzado el satélite UPM-Sat 1 [15], este satélite desarrollado por un grupo de profesores de la Escuela Técnica Superior de Ingenieros Aeronáuticos de la Universidad Politécnica de Madrid (UPM) es el proyecto predecesor al UPMSat-2. El éxito en el lanzamiento del UPM-Sat 1 demostró la capacidad del grupo perteneciente a la UPM de llevar a cabo proyectos de este tipo.

El UPM-Sat 1 se concibió como un satélite científico y de demostración tecnológica en órbita con un carácter fundamentalmente educativo ya que en el proyecto participaron tanto profesores como alumnos de último curso y doctorado.

El satélite que figura inscrito como UPM- Sat 1/ ROLEU en el Registro de Objetos Lanzados al Espacio Ultraterrestre español (ROLEU) y en el registro de la Organización de Naciones Unidas entra en la categoría de micro-satélite, con un peso de 47 kilogramos. Enviado al espacio por la lanzadera Ariane IV-40 ASAP contó

con una vida operativa de 213 días en la cual siguió una órbita polar heliosíncrona a 670 kilómetros de altitud.

El proyecto UPMSat-2 puede considerarse la continuación del proyecto UPM-Sat 1, en el que se desarrolla un nuevo micro-satélite con la colaboración de profesores y alumnos de distintas escuelas de la UPM.

2.2. Motivación

El computador de a bordo del satélite cuenta con distintos dispositivos de entrada y salida que son diseñados y hechos ad hoc para el proyecto ya que es necesario que, debido a las características del mismo, el nivel de integración y unicidad se realice de forma que la huella en memoria sea la mínima posible así como el riesgo de introducción de errores.

Cabe destacar que al tratarse de un sistema crítico es necesario resaltar la importancia de la realización de pruebas completas de manera que puedan ser detectados posibles errores en el sistema.

Para el correcto funcionamiento del computador de a bordo (OBC) y de los dispositivos asociados a éste se han desarrollado distintos manejadores. Estos han sido implementados sobre la versión preliminar del computador de a bordo y deben ser probados y adaptados a la versión de vuelo del mismo.

A su vez se hace necesario la existencia de un manejador para dar soporte al almacenamiento de datos en una memoria no volátil. Con el uso de este tipo de memorias se asegura poder recuperar la información en caso de una pérdida de suministro eléctrico, por ejemplo que el satélite se quede sin batería.

2.3. Objetivos

Los objetivos a conseguir con la realización de este proyecto son los siguientes:

- Estudio de la documentación del proyecto, así como el estudio de las herramientas de desarrollo utilizadas.
- Estudio de los manuales de usuario de las placas relacionadas con el computador de a bordo del satélite.
- Estudio de las pruebas ya realizadas para el software del computador de a bordo.
- Desarrollo de nuevas pruebas para la integración del software sobre la versión de vuelo del computador de a bordo.

- Estudio de los manejadores ya realizados.
- Adaptación y desarrollo de nuevos manejadores.

2.4. Estructura

A partir de este punto en el presente documento se expone el trabajo realizado por la alumna dentro del proyecto. Para ello se expondrá en el capítulo 3 el proyecto en el que está enmarcado y las tecnologías y herramientas utilizadas.

A continuación, se pasará a explicar las tres tareas principales que han sido llevadas a cabo por la alumna: las pruebas de validación en el capítulo 4, el desarrollo del manejador de la memoria no volátil en el capítulo 5 y cómo se ha realizado el arranque desde la memoria no volátil, desarrollado en el capítulo 6.

Por último, la autora presentará, en los dos últimos capítulos, sus conclusiones acerca del trabajo realizado y lo aprendido durante su desarrollo así como un estudio de las posibles líneas futuras basadas en los resultados obtenidos en este proyecto.

Capítulo 3

Entorno de desarrollo

3.1. UPMSat-2

El UPMSat-2 es un proyecto creado para la realización de un micro-satélite universitario que se utilizará como plataforma de demostración en órbita. El UPMSat-2 se compone por un conjunto de experimentos realizados por diferentes empresas españolas y grupos de investigación de la UPM cuyo objetivo es la comprobación de distintos equipos en condiciones de espacio.

El proyecto está liderado por el IDR asociado a la Escuela Técnica Superior de Ingenieros Aeronáuticos de la UPM. En el proyecto colaboran varios grupos de investigación así como diferentes empresas del sector aeroespacial.

El satélite UPMSat-2, véase figura 3.1, pertenece, al igual que su antecesor UPMSat 1 a la categoría de micro-satélites. Los micro-satélites son satélites con un peso aproximado de 50 kilogramos y unas dimensiones externas de $0{,}5 \times 0{,}5 \times 0{,}6$ metros.

Este satélite seguirá una órbita baja polar entorno a los 600 km de altura. Al realizar esta órbita el satélite tendrá dos períodos de visibilidad desde el segmento de tierra cada 24 horas. Por cada período de visibilidad habrá una conexión máxima de 10 minutos, durante los cuales se llevarán a cabo las comunicaciones con el satélite.

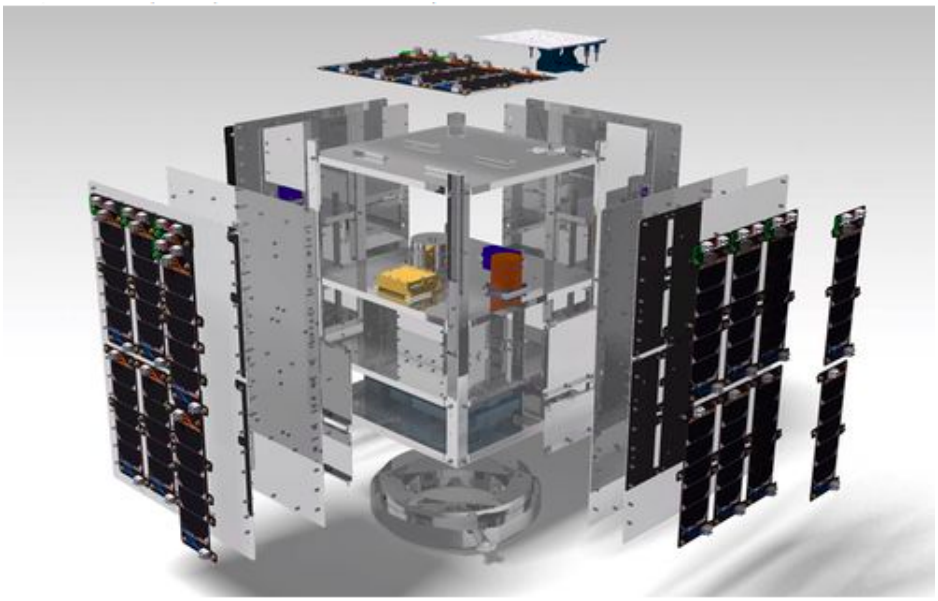

Figura 3.1: Imagen de la estructura del UPMSat-2

El software embarcado en el satélite, realizado por el grupo STRAST, tiene las siguientes funciones principales que pueden verse en la figura 3.2:

- Control y determinación de actitud (ADCS)
- Supervisión y control de la plataforma del satélite (housekeeping).
- Envío de mensajes de telemetría a la estación de tierra.
- Descodificación y procesamiento de órdenes remotas recibidas de la estación de tierra.
- Gestión del tiempo a bordo del satélite.
- Detección de fallos en la plataforma y gestión de modos de funcionamiento.
- Control de la ejecución de los experimentos, adquisición de datos y envío a tierra de la telemetría correspondiente.

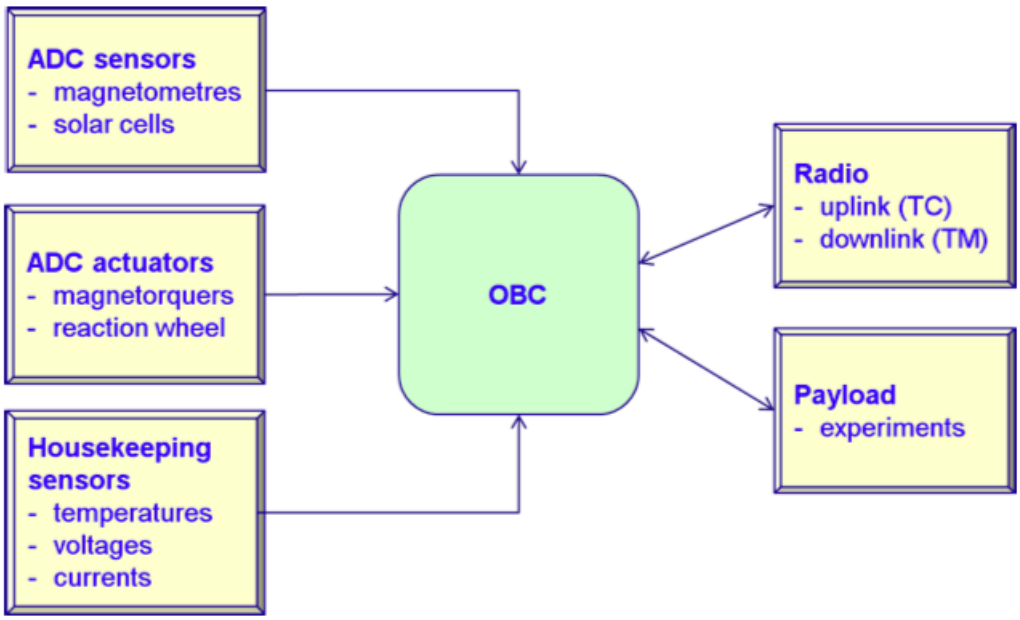

Figura 3.2: Funciones software embarcado

3.1.1. Computador de a bordo

El computador de a bordo es el encargado de realizar el control de todo el equipo del satélite. Está compuesto por un System On a Chip (SOC) que contiene 2 MB de memoria EEPROM y 4 MB de memoria SRAM, una FPGA de Actel y varias entradas y salidas digitales.

A su vez, la FPGA está formada por el procesador LEON3 que trabaja con una frecuencia de 20 MHz, una jerarquía de buses AMBA para el acceso a memoria y buses UART, SPI e I2C para la comunicación con los periféricos. Puede verse la arquitectura del OBC en la figura 3.3.

El procesador LEON3 [9] que contiene el OBC pertenece a la familia de microprocesadores LEON, desarrollado inicialmente por el Centro de Tecnología e Investigación Espacial Europeo (ESTEC), que forma parte de la Agencia Espacial Europea (ESA), y su desarrollo lo continuó la empresa Gaisler Research. Son procesadores de 32 bits basados en la arquitectura SPARC V8 RISC y descritos en VHDL sintetizable. El LEON3, utilizado en este proyecto es la versión más reciente de esta serie de procesadores.

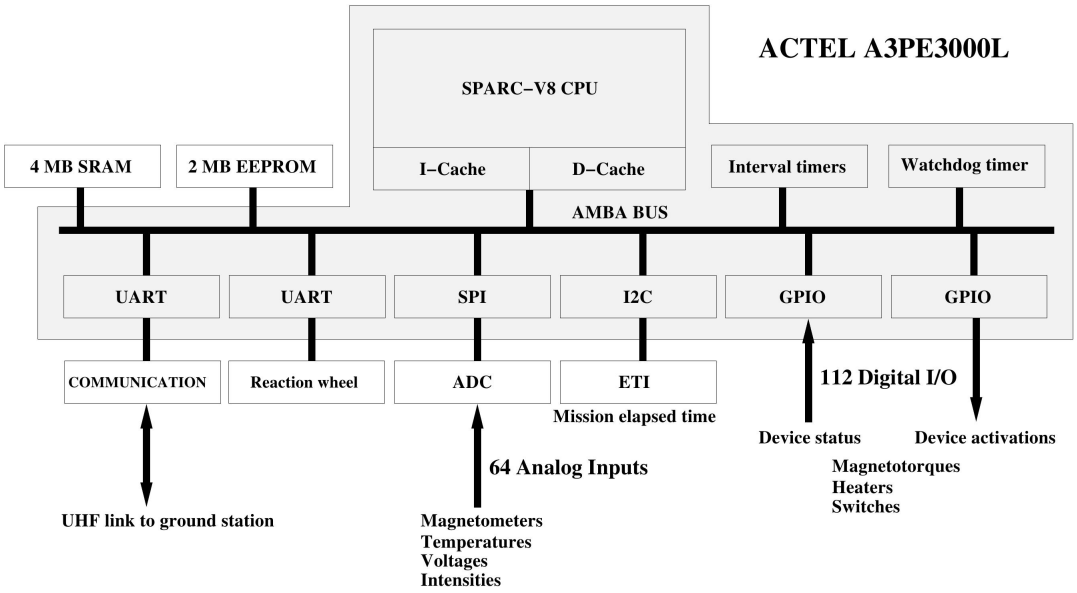

Figura 3.3: Arquitectura hardware del OBC

La versión definitiva del OBC está integrada junto con otras tarjetas en el EBOX (Electronic Box), un pequeño rack robusto cuyo esquema puede verse en la figura 3.4. Está compuesto, además de por el OBC, por las siguientes tarjetas:

- **PSU:** La Power Supply Unit (PSU) es la tarjeta encargada de proporcionar tanto la alimentación al resto de tarjetas del EBOX como de recargar las baterías del satélite cuando sea necesario. La PSU arranca dos horas después de la separación del satélite con la lanzadera, y es la encargada de reiniciar todo el sistema. Existe un temporizador llamado Watch Dog Timer (WDT) que debe refrescarse periódicamente a fin de que el sistema no se reinice.

- **PDU:** La Power Distribution Unit (PDU) se encarga de la gestión de los experimentos del satélite. Recibe del OBC las órdenes correspondientes a la habilitación y deshabilitación de los experimentos así como la activación o desactivación de sus alimentaciones o el cambio de polaridad en el caso de los magnetómetros.

- **DAS:** Digital to Analog Subsystem. Es la tarjeta encargada de realizar la conversión de analógico a digital contando para ello con un conversor de análogico a digital (ADC) de 8 canales y un multiplexor de 64 a 8 canales. Los datos analógicos convertidos en digitales son enviados a través del bus SPI. Es necesario la realización de esta conversión para que estos datos puedan ser interpretados por la FPGA.

- **DB:** Distribution Board. Es la tarjeta encargada de realizar toda la interconexión de las tarjetas para que puedan comunicarse.

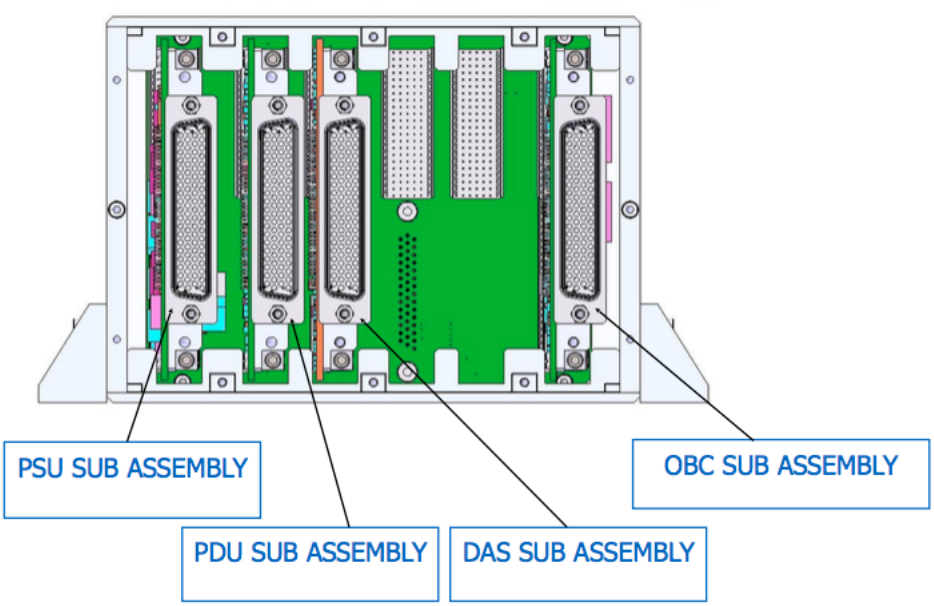

Figura 3.4: Esquema del EBOX

3.2. Sistema operativo del software embarcado del OBC

El software de vuelo se ejecutará sobre una versión del ORK+ [10], el Open Ravenscar real-time Kernel. Este kernel ha sido desarrollado por el mismo grupo STRAST y soporta el perfil de concurrencia Ranvescar, que sigue el estándar de Ada 2005.

3.2.1. Lenguaje Ada

El software se ha desarrollado utilizando el lenguaje de programación Ada [1]. Este lenguaje ofrece numerosas ventajas a la hora de implementar sistemas de tiempo real de alta integridad. A continuación se realiza un pequeño resumen de las ventajas ofrecidas:

- **Legibilidad.** Ada es un lenguaje en el que predomina la legibilidad del código. Con esto se consigue una mayor mantenibilidad.
- **Tipado fuerte.** A través del tipado en Ada se asegura que un objeto nunca contenga un conjunto de valores que no sean los que se han definido en su rango y el compilador detecta errores si se intenta operar con objetos de tipos distintos.

- **Concurrencia.** La necesidad de concurrencia es una de las características de los sistemas de tiempo real, como los satélites, en los que se precisa de la ejecución paralela de varias actividades. El modelo de concurrencia ofrecido por Ada es un modelo muy complejo y potente. Ada permite la definición de perfiles de aplicación a base de restricciones que hacen que su uso sea más simple y preferible para propósitos particulares, uno de estos perfiles es el perfil de Ravenscar.

- **Cláusulas de representación.** Las cláusulas de representación permiten definir dispositivos hardware y proporciona a su vez legibilidad y tipado fuerte.

3.2.2. Perfil de Ravenscar

El perfil de Ravenscar consiste en un subconjunto del lenguaje Ada pensado especialmente para los sistemas de tiempo real. En él se incorporan una serie de restricciones que se realizan al lenguaje y que afecta a algunas de las operaciones que éste permite, de esta forma se asegura que las operaciones permitidas son seguras, sobre todo para los sistemas de tiempo real de gran fiabilidad, ya que al ser un conjunto de directivas más reducido se asegura mayor fiabilidad.

En concreto, el perfil de Ravenscar está compuesto por un conjunto de directivas que se da al compilador para así poder evitar el uso de ciertas estructuras de concurrencia proporcionadas por el lenguaje Ada. Son restricciones que únicamente afectan a programas concurrentes, los programas completamente secuenciales no se ven afectados por estas restricciones. El listado 3.1 muestra las directivas de este perfil:

```
1  pragma Task_Dispatching_Policy (FIFO_Within_Priorities);
2  pragma Locking_Policy (Ceiling_Locking);
3  pragma Detect_Blocking;
4  pragma Restrictions (
5           No_Abort_Statements,
6           No_Dynamic_Attachment,
7           No_Dynamic_Priorities,
8           No_Implicit_Heap_Allocations,
9           No_Local_Protected_Objects,
10          No_Local_Timing_Events,
11          No_Protected_Type_Allocators,
12          No_Relative_Delay,
13          No_Requeue_Statements,
14          No_Select_Statements,
15          No_Specific_Termination_Handlers,
16          No_Task_Allocators,
17          No_Task_Hierarchy,
```

```
18          No_Task_Termination,
19          Simple_Barriers,
20          Max_Entry_Queue_Length => 1,
21          Max_Protected_Entries  => 1,
22          Max_Task_Entries       => 0,
23          No_Dependence => Ada.Asynchronous_Task_Control,
24          No_Dependence => Ada.Calendar,
25          No_Dependence => Ada.Execution_Time.Group_Budget,
26          No_Dependence => Ada.Execution_Time.Timers,
27          No_Dependence => Ada.Task_Attributes);
```

Código 3.1: Directivas del perfil de Ravenscar

3.3. Herramientas utilizadas

Para la compilación del software implementado se utilizará la herramienta de compilación GNAT con el perfil de Ravenscar, el ORK+ está integrado con esta herramienta.

GNAT es un compilador del lenguaje de programación Ada, creado por la Universidad de Nueva York y actualmente es mantenido por la empresa AdaCore, tiene disponible un entorno de desarrollo gráfico llamado GPS (GNAT Programming Studio) que ha sido la herramienta utilizada para el desarrollo del software.

Para la compilación de dicho software ha sido necesaria la utilización de un compilador cruzado ya que el código creado ha de ejecutarse en una plataforma diferente a la plataforma en la que se ha generado. Por este motivo se ha incluido la distribución GNATforLEON, con esta herramienta es posible la compilación cruzada para la ejecución del código sobre procesadores de la familia LEON.

Se ha utilizado la herramienta grmon2 [5] para realizar la depuración del software, esta herramienta permite realizar la carga y depuración de programas en el OBC a través de una línea serie. Está desarrollada por la empresa Aeroflex Gaisler.

También cabe destacar la utilización de la herramienta mkprom2 [4], desarrollada también por Aeroflex Gaisler. Esta herramienta ha sido utilizada con el fin de crear ficheros ejecutables para su carga en memoria no volátil.

Capítulo 4

Pruebas de validación

Al disponer del EBOX con la versión definitiva del computador de a bordo se hace necesaria la realización de varias pruebas para la integración de las diferentes tarjetas contenidas en éste, así como la validación de la funcionalidad del software. Aunque como se verá, también sirven para integrar distintos manejadores como es el ejemplo de la integración del SPI con la DAS.

Por tanto, debe validarse por un lado el correcto funcionamiento del hardware y, por otro lado, que el software cumple con los requisitos de diseño proporcionados en el manual de usuario [12]. Dicho software ha sido desarrollado en una versión preliminar del OBC, por lo que es necesario que sea probado, y en su caso, adaptado a la versión de vuelo del mismo.

Al tratarse de un sistema embarcado esta validación se realiza a través de un BSP (Board Support Package), en él se realiza la implementación del software de bajo nivel que va a ser utilizado para validar las tarjetas. De este modo se realiza al mismo tiempo la comprobación del hardware y software del sistema.

Por estos motivos se ha definido un conjunto de pruebas funcionales, descritas en el documento Pruebas Funcionales EBOX [13] junto con Tecnobit, empresa encargada de la realización de la EBOX. En algunas de estas pruebas se hace necesario simular el modo vuelo del satélite, para ello se dispone de un interruptor que hace posible diferenciar el modo de vuelo del de tierra. A continuación puede verse un listado de las pruebas realizadas con una descripción de éstas:

- **PSU-WDG-001.** En esta prueba se realiza la habilitación de la PSU y se recarga el WDT para comprobar que no se reinicia todo el sistema.

- **DAS-STATUS-001.** Se realiza el arranque de la DAS y se comprueba el estado de sus protecciones.

- **PDU-STATUS-001.** Se realiza el arranque de la PDU y se comprueba el estado de sus protecciones.
- **PSU-OFF-001.** Se habilita la PSU y se comprueba que se puede realizar un apagado completo de todo el equipo.
- **PSU-BAT-001.** En esta prueba se realiza la habilitación de la PSU, se comprueba el estado de la batería y se recarga el WDT para comprobar que no se reinicia todo el sistema.
- **PDU-PROT-001.** Se realiza el arranque de la PDU, se activan las protecciones de ésta y se comprueba su estado, realiza la activación de los distintos dispositivos (experimentos) del satélite y se comprueba su estado.
- **PSU-RTC-001.** Se habilita la PSU, se comprueba el estado del RTC a través del I2C y los datos obtenidos por parte del OBC.
- **PDU-HEAT-001.** Se arranca la PDU, se realiza la activación del latch y la activación de los calentadores.
- **PDU-MGT-001.** Se realiza el arranque de la PDU, la activación del latch y se comprueba el cambio de sentido de los magnetopares (MGT).
- **DAS-SPI-001.** Se arranca la DAS, se comprueba el bit de adquisición de datos, se verifica la transmisión a través del SPI y la lectura de éste.
- **DAS-ADC-001.** Se arranca la DAS, se comprueban los datos analógicos, se verifica la transmisión por el SPI y se comprueba la salida por parte del OBC con la tensión medida.
- **DAS-PDU-001.** Se realiza el arranque de la DAS, se comprueba el bit de adquisición de datos, se verifica la transmisión a través de SPI y su lectura. A su vez se realiza el arranque de la PDU, la activación de las protecciones y la comprobación del estado de éstas.
- **EBOX-001.** Se habilita la PSU, se recarga el WDT para comprobar que no se reinicia todo el sistema. A su vez se arranca la PSU, se activan las protecciones y se realiza la lectura del estado de éstas. También se realiza el arranque de la DAS, la comprobación del bit de adquisición de datos, la transmisión a través del SPI y su lectura. Por último se realiza la habilitación de la PSU y se comprueba el apagado completo del equipo.

La autora del proyecto ha colaborado en mayor o menor medida en la realización de todas las pruebas, habiendo realizado de manera exclusiva las siguientes: PDU-PROT-001, PDU-HEAT-001, PDU-MGT-001, DAS-SPI-001 y DAS-PDU-001.

Después del estudio y comprensión tanto del lenguaje de programación Ada así como del esquema del OBC, la codificación de dichas prueba no ha resultado una tarea complicada, siendo la depuración de estas la fase más laboriosa.

4.1. Depuración

Debido a que se verificó tanto la integración del hardware como la validación del software, los errores obtenidos durante la depuración pueden deberse bien a errores de configuración en la integración del EBOX o bien a errores en la implementación del código. En cualquier caso dichos errores se subsanaron. De las pruebas realizadas los errores más difíciles de resolver se detectaron tras las pruebas PSU-WDG-001, PSU-BAT-001 y PDU-PROT-001.

La prueba PSU-WDG-001 es la encargada de comprobar el arranque del satélite simulando el modo vuelo de éste. En dicha prueba la PSU debía arrancar a las dos horas de desconectar el interruptor de simulación de tierra, pero la PSU no arrancaba. Al realizar la prueba se detectó un problema de hardware causado por una resistencia de la tarjeta en mal estado, esta resistencia que tenía una conexión estropeada causaba un mal funcionamiento de la placa. Una vez cambiada la resistencia se obtuvieron resultados satisfactorios consiguiendo realizar el arranque de la PSU en el estado de vuelo del satélite a las dos horas.

Con la prueba PSU-BAT-001 se comprueba el estado de la batería del satélite, un nivel adecuado de batería oscila entre 22 V y 24 V, en caso de la carga de la batería sea mayor se debe dar un aviso de batería alta, mientras que si es menor deberá darse el aviso de batería baja, si la carga de batería es menor de 21 V deberá mostrarse un aviso de batería crítica. Al realizar esta prueba simulando diferentes niveles de voltaje los mensajes obtenidos no eran los adecuados, los mensajes se obtenían de un modo confuso que no correspondía con el nivel de carga de batería que se estaba dando al EBOX, por ejemplo, cuando se debía mostrar el mensaje de batería baja se obtenía el de batería crítica.

El hecho de recibir mensajes no correspondientes a los estados reales de la batería derivó en una revisión del código del manejador encargado de realizar esta funcionalidad. Se detectó que, mientras en el manual de usuario [12] se especificaba que estos valores trabajan a nivel bajo, en el código estaban implementados a nivel alto. Modificando la parte de código correspondiente se corrigió esta anomalía, comprobando después el correcto funcionamiento de los avisos del estado de la batería.

Tras la realización de la prueba PDU-PROT-001 se detectó un comportamiento anómalo. Esta prueba es la encargada de comprobar la activación de los dispositivos y experimentos del satélite, para ello primero arranca la PDU activando las protecciones de ésta y se recarga el WDT para impedir que se reinicie el sistema. Los dispositivos que se activan son los siguientes:

- Magnetómetros (MGM).
- Sensores de temperatura.

- Rueda de reacción (RW).
- Magnetopares (MGT).
- Experimento BOOM1.
- Experimento BOOM2.
- Micro Thermal Switch (MTS).
- Modem.

Esta prueba daba resultados confusos, ya que algunos dispositivos como el MTS parecía encenderse o no dependiendo del orden en que se activasen todos los dispositivos, y otros, como el BOOM1 se encendía y apagaba. Este resultado dio lugar a plantearse la posibilidad de que la activación de estos niveles estuviese siendo afectada por el resto de dispositivos, ya que se realizaron una pruebas únicamente para estos dispositivos y los resultados de las pruebas parecían mostrar un funcionamiento correcto. A raíz de éste funcionamiento se detecta que una de las resistencias de la placa tiene un "glitch", es decir, un cambio esporádico durante su funcionamiento.

4.2. Test de regresión

Como se ha visto el hardware está bajo depuración, esto quiere decir que el hardware sufre cambios durante la realización de las pruebas de validación al ser detectados y subsanados errores en la implementación. Por este motivo se hace importante destacar la necesidad de realizar un test de regresión durante las pruebas de validación.

Con el test de regresión se asegura que funcionalidades ya validadas del software siguen comportándose correctamente tras las modificaciones del hardware. Para realizar esta comprobación se vuelven a ejecutar todas las pruebas realizadas hasta el momento del cambio en el hardware de las tarjetas del EBOX.

Junto con la realización del test de regresión se detectaron nuevos errores en el comportamiento de algunas pruebas, como es el caso de la anteriormente comentada, la PDU-PROT-001. Después de conseguir el funcionamiento correcto de esta prueba, tras un nuevo cambio en el hardware del equipo se volvió a detectar anomalías en la activación de los dispositivos de dicha prueba. En relación con este nuevo comportamiento, diferente al comportamiento erróneo de la vez anterior, se está a la espera de que Tecnobit confirme que los resultado incorrectos se deben a que no están conectados los dispositivos que trata de activar.

Capítulo 5

Manejador de memoria no volátil

Durante el proyecto se ha llevado a cabo el desarrollo del manejador encargado de dar soporte de almacenamiento a la información en la memoria no volátil del computador de a bordo. La información que se va a almacenar corresponde con los telecomandos, la telemetría y la información recogida por el satélite y sus experimentos. Se considera telemetría a los datos enviados por parte del satélite a la estación de tierra mientras que los telecomandos son los datos que se mandan desde la estación de tierra al satélite. El resto de información corresponde con la información adquirida por el satélite sobre su comportamiento y el de los experimentos que se van a realizar.

La memoria no volátil utilizada por el OBC es una memoria de tipo EEPROM, es decir, una memoria programable que se puede borrar. Existen dos tipos de memoria ROM borrable, la memoria EPROM (Erasable Programmable Read Only Memory) que es una memoria no volátil en la que el borrado se hace mediante impulsos de luz ultravioleta y la memoria EEPROM (Electrically Erasable Progammable Read Only Memory) en la cual el borrado se realiza a través de impulsos eléctricos generados por el propio computador siendo por tanto, la adecuada para este proyecto.

El OBC del UPMSat-2 contiene dos circuitos EEPROM de 1 MB de tamaño cada uno, lo que hace un total de 2 MB de memoria. Podemos ver el mapa de memoria EEPROM con los espacios ya reservados en la figura 5.1.

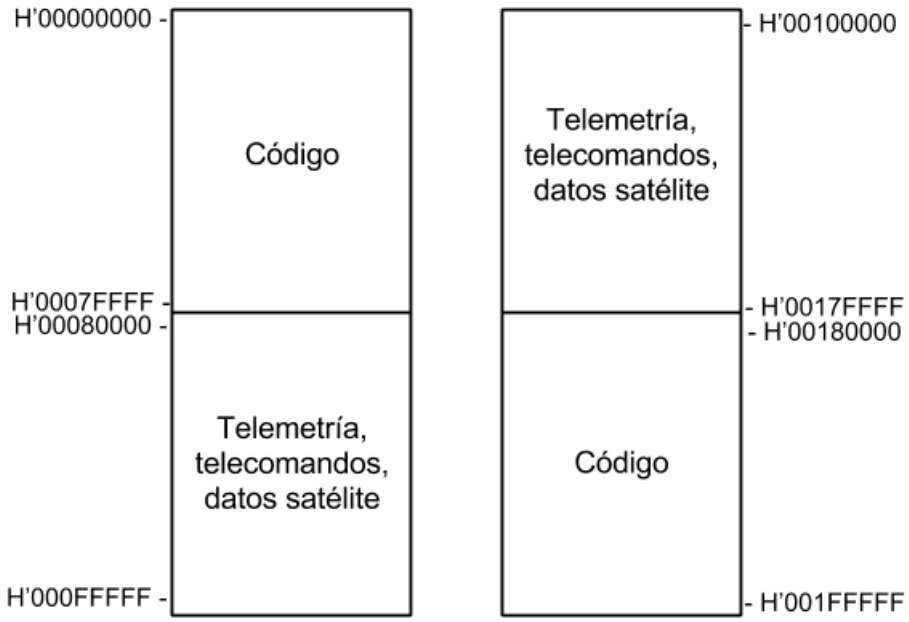

Figura 5.1: Mapa de la memoria EEPROM

Con el fin de almacenar tanto la telemetría, telecomandos e información del satélite se ha reservado un total de 1 MB de memoria. Se ha considerado que es espacio suficiente debido a que, como se ha explicado anteriormente, el satélite tendrá dos periodos de visibilidad (de 10 minutos cada uno) con la estación de tierra cada 24 horas.

Durante el tiempo de cada conexión con la estación de tierra se envía la información a una velocidad de 9600 baudios, lo que hace un total de 0.5 MB de datos aproximadamente a enviar. Además se debe tener en cuenta que durante este tiempo se reciben los telecomandos de la estación de tierra y debido a que la transmisión se realiza de forma semi-dúplex solamente podrá enviarse información en un sentido a la vez. Ésto quiere decir que el tiempo a enviar datos será menor a esos 10 minutos.

Como podemos ver en la figura 5.1 los primeros 512 KB de la memoria se utilizarán para almacenar el código de la misión, a partir de ahí, tomando como dirección base la dirección H'00080000 ocuparemos 1MB de memoria para la telemería, los telecomandos y el resto de información. Quedarán otros 512 KB que se utilizará de nuevo para el almacenamiento del código. Se tendrá duplicado el código por seguridad, de esta forma en caso de que alguna parte del código haya quedado corrupta podremos recuperarlo accediendo al código almacenado en el otro circuito EEPROM.

Debido a las especificaciones técnicas de la memoria EEPROM [3] se debe esperar un tiempo total de 15 milisegundos después de cada escritura, ya sea a nivel de

bloque, de palabra o de byte. Para simplificar el manejo de la memoria no volátil se utilizarán bloques de longitud fija de 512 bytes para almacenar la telemetría, telecomandos e información.

Al no disponer todavía de la tarjeta de radio que se utilizará no ha sido posible definir el tamaño óptimo de los bloques de telemetría, telecomandos o información que serán almacenados en la memoria, por este motivo se ha dejado como un parámetro en la implementación del manejador. Para la realización de pruebas del manejador se ha considerado que, para poder aprovechar todo el ancho de banda en las escrituras los bloques de datos sean del mismo tamaño que el de los bloques en los que está dividida la memoria EEPROM.

Esta igualdad en el tamaño de los bloques nos asegura la ausencia de huecos entre los bloques escritos. Este fenómeno, conocido como segmentación, implica la necesidad de realizar una búsqueda de huecos de tamaño adecuado a los bloques que quieren escribirse.

Las operaciones que se pueden realizar sobre estos bloques de memoria son las encargadas de la escritura, lectura y liberación de bloques en la memoria y por tanto son las que realizará el manejador de la memoria EEPROM.

Escritura:

La operación de escritura deberá ser atómica para poder garantizar la escritura de la totalidad del contenido del bloque, ya que de no ser así la información almacenada en la memoria la memoria no sería correcta.

Por este motivo, la escritura de un bloque en memoria debe hacerse con la máxima prioridad con el objetivo de garantizar que ninguna otra tarea interrumpa la operación antes de que termine. Por tanto la operación de escritura se va a separar en dos operaciones distintas, de esta forma podemos asegurar la exclusión mutua y la sincronización con el resto de operaciones.

La primera operación para la escritura se encargará de encontrar la posición en la que se debe almacenar el bloque y la segunda operación realizará la copia de dicho bloque en la posición de memoria, refiriéndonos a esta última como operación de actualización del bloque y a la primera como operación de escritura.

Para llevar a cabo la escritura es necesario encontrar un bloque cuyo estado nos indique que está disponible. Una vez seleccionado un bloque para ser escrito su estado deberá ser modificado, de esta forma se reconocerá que hay contenido en él y no será sobreescrito.

Lectura:

La lectura se realiza de una forma más sencilla que la escritura, ya que sabiendo el identificador del bloque que se quiere leer simplemente habrá que obtener la dirección de memoria en la que se encuentra dicho bloque. Primero comprobaremos que el bloque a leer está en uso, si es así entonces devolveremos el contenido de dicho bloque de memoria.

Liberación:

La liberación del bloque de memoria se realiza para que este se pueda sobreescribir con nueva información. Simplemente se deberá marcar ese bloque como libre y, por tanto, pueda ser utilizado de nuevo.

5.1. Diseño

Como se ha explicado en la sección anterior es importante realizar un control de acceso a la EEPROM, consiguiendo tanto la exclusión mutua como la sincronización para las operaciones realizadas.

Debido a las características de concurrencia del sistema, el primer diseño del manejador se basa en conseguir la exclusión mutua a través de objetos protegidos y garantizar la sincronización a través de barreras. Ambos recursos de la concurrencia serán usados para proteger las operaciones anteriormente descritas y permitir la sincronización entre las mismas, de manera que cuando una barrera se encuentre cerrada no se pueda acceder a la operación del objeto protegido.

Se han realizados dos diseños para el desarrollo del manejador cumpliendo ambos con los requisitos de forma diferente.

5.1.1. Primer diseño

Como se ha explicado anteriormente el proceso de escritura debe ser un proceso que funcione con máxima prioridad y no debe bloquearse por ningún otro, por tanto la actualización final dentro de la memoria se implementará en un objeto protegido a parte.

Debido a la necesidad, anteriormente descrita, al finalizar la escritura se debe esperar un tiempo de 15 milisegundos antes de realizar otra operación. En Ada se prohibe realizar operaciones bloqueantes, como un retardo, dentro de un objeto protegido. Por tanto, se debe añadir una tarea encargada de llamar al objeto protegido que realiza la actualización final del bloque en memoria.

La complejidad de este diseño reside en, que debido al perfil de Ravenscar, no puede haber más de una entrada por objeto protegido, impidiéndonos por tanto tener todos los subprogramas de las operaciones definidas en el mismo objeto protegido.

Este perfil tampoco autorizada encolar tareas en una operación protegida, por tanto solamente se podrá tener la tarea de escritura y se deberán crear más objetos protegidos para conseguir realizar todas las operaciones. Tendremos un total de cuatro objetos protegidos y una tarea como puede verse en la figura 5.2.

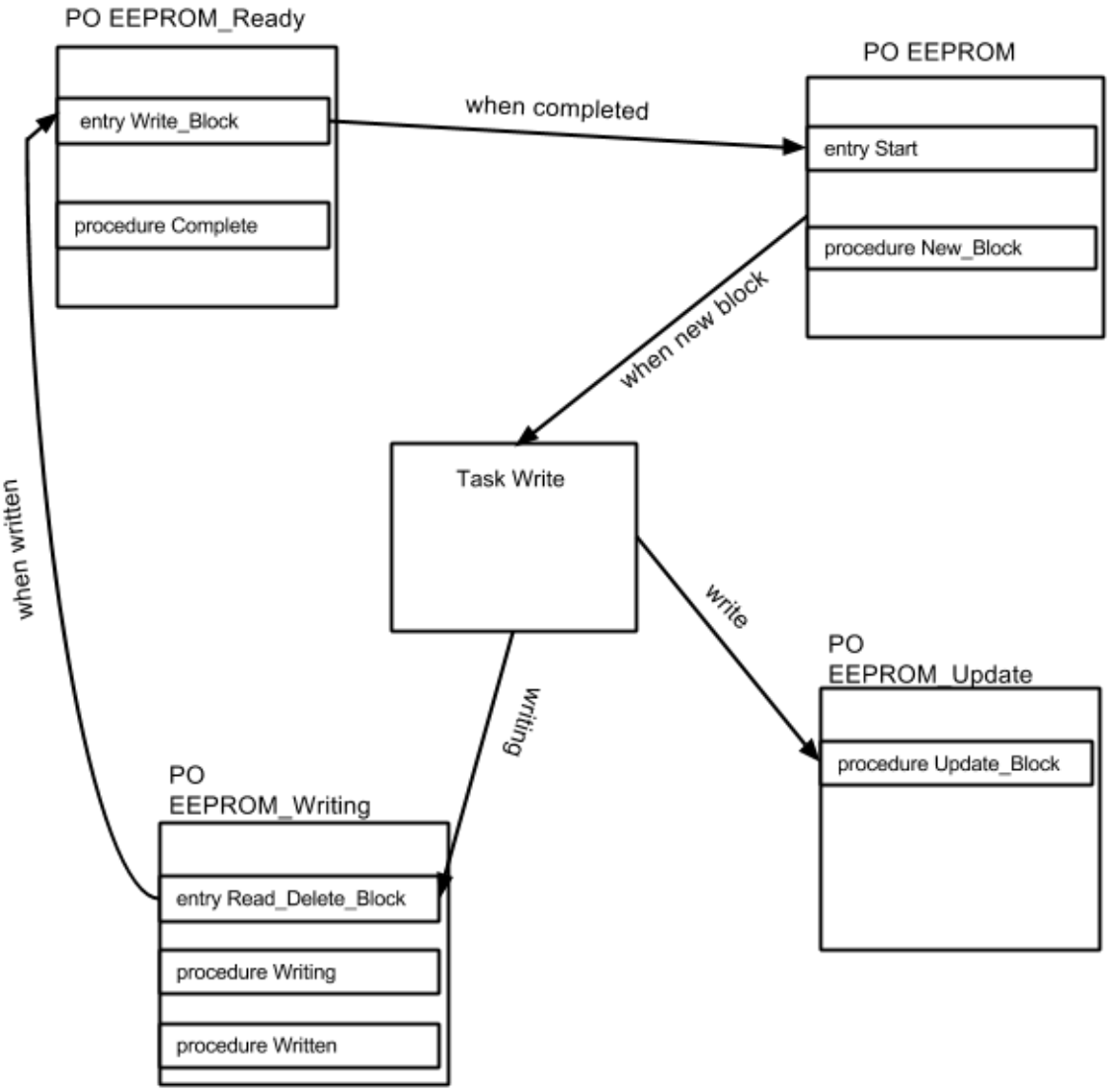

Figura 5.2: Primer diseño del manejador

5.1.2. Segundo diseño

Como una de las premisas del software seguro es mantener un diseño simple, se descarto la implementación del diseño 1 y se decidió usar un diseño más simple basado en el uso de menor número de objetos protegidos. El diseño puede verse en 5.3.

Este diseño consta únicamente de dos objetos protegidos, un objeto protegido EEPROM que contiene los subprogramas correspondientes a la escritura, liberación y lectura de un bloque y un segundo objeto protegido, EEPROM_Update, que contiene el procedimiento encargado de realizar la actualización (o escritura final) del bloque en memoria. Se decidió realizar esta parte de la escritura en un objeto separado porque como hemos visto es necesario que esta operación se realice con máxima prioridad.

Al dar mayor prioridad al objeto encargado de la actualización en memoria del bloque nos aseguramos que ninguna otra tarea interrumpe la escritura. Esto también nos permite suprimir la utilización de la tarea implementada en el diseño anterior, realizando por tanto en este caso el retardo de 15 milisegundos a través de una espera activa al final del procedimiento encargado de escribir el bloque.

Se ha dado por buena la posibilidad de utilizar una espera activa al ser un procedimiento que está dentro de un objeto protegido cuyo techo de prioridad es bajo y, por tanto, no impedirá que tareas prioritarias sean despachadas para su ejecución.

Como puede observarse este diseño cumple con las necesidades de que la escritura se realice sin interrupciones y se proporcione exclusión mutua entre las tres operaciones. Por tanto, al ser más sencillo y usar menor número de recursos computacionales se ha decidido implementar este mismo.

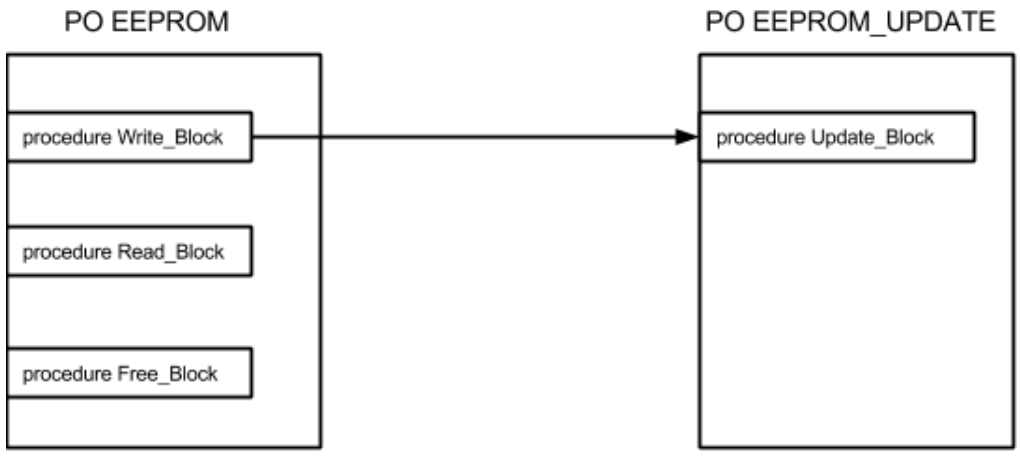

Figura 5.3: Segundo diseño del manejador

5.2. Arquitectura Software del manejador

Por lo general, la arquitectura que sigue un manejador de dispositivos se compone de los siguientes módulos:

- HLInterface: Contiene la interfaz de alto nivel del manejador.
- Registers: Contiene las definiciones de los registros de control así como de otras estructuras que puedan ser útiles.
- Parameters: Contiene la definición de los parámetros necesarios para el desarrollo del manejador.
- Core: Contiene toda el código necesario para la implementación de las operaciones que se van a realizar.

La memoria EEPROM no dispone de registros de control y estado y por lo tanto no es necesaria la codificación de este módulo.

Por tanto, podemos ver en la figura 5.4 como queda el esquema de la arquitectura del manejador.

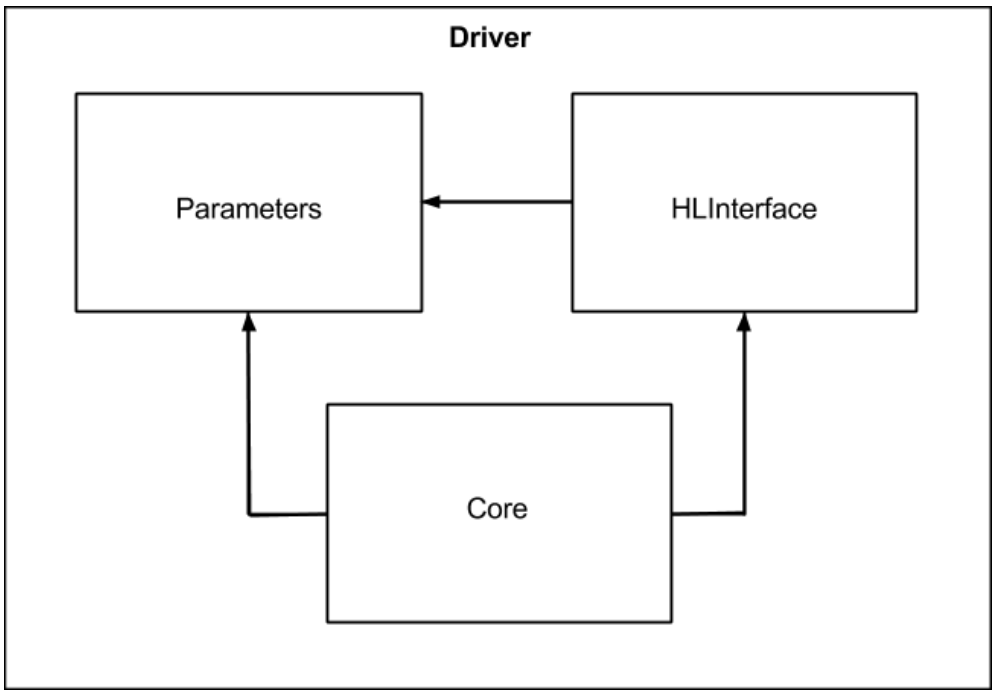

Figura 5.4: Arquitectura software del manejador

5.3. Codificación

En esta sección se presenta la implementación realizada del manejador para el cual, como se ha explicado anteriormente, se ha seguido el diseño de la figura 5.3.

A continuación se explica de una forma detalla la implementación que se ha realizado de los módulos correspondientes a Parameters y a Core, según la figura 5.4.

Es necesario destacar que el módulo HLInterface no se detalla, ya que únicamente consiste en la realización de llamadas a las operaciones implementadas en el módulo Core.

5.3.1. Parámetros:

El módulo Parameters ha sido el primer módulo desarrollado, consta de un único fichero de especificación en el que se ha implementando la lista de parámetros necesarios para la realización del driver. Los parámetros que se han definido pueden verse en el código 5.1 y son:

- **Size_EEPROM:** Este parámetro define el tamaño total de la memoria EEPROM que se dedica, es decir 1MB.
- **Base_Address_EEPROM:** Se ha definido la dirección H'00080000 como dirección base, a partir de la cual se almacenarán los bloques.
- **Block_Size:** Este parámetro contiene el tamaño de los bloques que se van usar en la memoria. Si en un futuro se decide utilizar bloques de distinto tamaño bastará con modificar este parámetro.
- **Ceiling_Priority:** Este es el techo de prioridad del objeto protegido que proporciona exclusión mutua en el acceso a la EEPROM.

```
1  Size_EEPROM : constant := 2**20;
2  -- Size of EEPROM memory.
3
4  Base_Address_EEPROM : Interfaces.Unsigned_32 :=
   16#00008000#;
5  -- Base address for eeprom.
6
7  Block_Size : constant := 128*4;
8
9  Ceiling_Priority : System.Priority := System.
   Default_Priority;
```

Código 5.1: Parámetros definidos

5.3.2. Core:

El módulo Core es el módulo principal del manejador, en él se definen y desarrollan las operaciones realizadas por éste, por tanto a diferencia del módulo anterior el Core estará compuesto por dos ficheros, la especificación y la implementación.

En la especificación, véase el código 5.2, se definen las operaciones que se realizan, así como los tipos abstractos de datos y las excepciones necesarias. Las operaciones definidas son las descritas al inicio del capítulo y corresponden con la escritura, la lectura y la liberación de bloques.

```
1
2     type Type_Block_Id is private;
3
4     subtype Stream_Element_Array_512 is Ada.Streams.
      Stream_Element_Array (1..512);
5
6     -- The size of the block is less than or greater than the
       block size (512B).
7     Block_Size_Error : exception;
8
9     -- The block doesn't exists in EEPROM.
10    Block_Not_Exists_Error : exception;
11
12    -- EEPROM is full.
13    Memory_Full_Error : exception;
14
15    procedure Write_Block (Block_EEPROM : in
      Stream_Element_Array_512 ; Block : out Type_Block_Id);
16    -- Writes a Block into the last address available in
      EEPROM.
17
18    function Read_Block (Block : in Type_Block_Id) return
      Stream_Element_Array_512;
19    -- Reads a Block stored in EEPROM.
20
21    procedure Free_Block (Block: in Type_Block_Id);
22    -- Deletes the block in EEPROM.
23
24 private
25    type Type_Block_Id is new Integer range 0..2047;
```

Código 5.2: Especificación del Core

Para poder manejar los bloques se ha implementado un tipo abstracto de datos Type_Block_Id que nos permitirá acceder a un bloque por su identificador, esto es, el número de bloque de la memoria no volátil. Además esta implementación nos permite asignar un estado a los bloques de la memoria permitiéndonos diferenciar entre los bloques con contenido y los bloques vacíos.

Es importante también destacar las excepciones que se han creado y que son necesarias para avisar al intentar realizar operaciones no permitidas, éstas son:

- **Block_Size_Error:** Esta es la excepción que se mostrará cuando los bloques que se quieren escribir sean de tamaño superior o inferior al tamaño de los bloques de memoria.
- **Block_Not_Exists_Error:** Esta excepción se mostrará al intentar realizar una lectura o una liberación de un bloque vacío.
- **Memory_Full_Error:** Cuando la memoria este llena y se intente la escritura de un nuevo bloque deberá mostrarse esta excepción.

Como se ha explicado, esta implementación consta de dos objetos protegidos, teniendo por tanto un objeto protegido EEPROM cuya implementación puede verse en el código 5.3 y un segundo objeto protegido EEPROM_Update cuya implementación está definida en el código 5.5.

El primer objeto protegido que encontramos, tiene el techo de prioridad asignado que se supone que será una prioridad baja. Este objeto protegido implementa directamente las operaciones definidas en la especificación.

```
1  -- Protected Object EEPROM
2  protected EEPROM is
3     pragma Priority (Parameters.Ceiling_Priority);
4     procedure Write_Block (Block_EEPROM: in
   Stream_Element_Array_512; Block: out Type_Block_Id);
5     function Read_Block (Block : in Type_Block_Id) return
   Stream_Element_Array_512;
6     procedure Free_Block (Block: in Type_Block_Id);
7  end EEPROM;
```

Código 5.3: Operaciones del Objeto Protegido EEPROM

A continuación se explica con más detalle cada una de estas operaciones:

- **Write_Block:** Este procedimiento corresponde a la operación de escritura, recibe como parámetro de entrada el bloque que se quiere almacenar en la memoria y devuelve el identificador que se ha asociado a dicho bloque. También es el encargado de realizar un llamada al procedimiento de actualización,

al que le pasa el bloque a actualizar y la dirección de memoria en la que se escribirá dicho bloque.

La dirección de memoria en la que se realizará la escritura se calcula de una forma sencilla. Cuando se ha encontrado en memoria un bloque disponible se calcula a partir de su identificador asociado, mediante la fórmula 5.1, la dirección en la que está el bloque.

$$dirección = identificador * tamaño\_bloque + dirección\_base \tag{5.1}$$

Una vez que se ha actualizado el bloque, deberá marcarse en la memoria como ocupado, asegurándonos que no será sobreescrito en una nueva operación de escritura.

Puede verse la implementación de este procedimiento en el fragmento de código 5.4.

```
1
2   procedure Write_Block (Block_EEPROM: in
    Stream_Element_Array_512; Block: out Type_Block_Id)
    is
3       Completed : Ada.Real_Time.Time;
4       Block_Address : System.Address;
5
6   begin
7       if (Block_EEPROM'Length = Parameters.Block_Size)
    then
8           for I in Is_Block_Used'Range loop
9               if not Is_Block_Used (I) then
10                  Block_Address := System'To_Address (
    Interfaces.Unsigned_32 (I*512) + Parameters.
    Base_Address_EEPROM);
11                  EEPROM_Update.Update_Block(Block_EEPROM,
    Block_Address);
12                  Is_Block_Used (I) := True;
13                  Block := I;
14                  exit;
15              end if;
16
17              if I = Is_Block_Used'Last then
18                  raise Memory_Full_Error;
19              end if;
```

```
20          end loop;
21
22          Completed := Ada.Real_Time.Clock;
23           -- Waits 15 millisecond before write a new
    block.
24          while (Ada.Real_Time.Clock < (Completed + Ada.
    Real_Time.Milliseconds (15))) loop
25             null;
26          end loop;
27
28       else
29          raise Block_Size_Error;
30       end if;
31    end Write_Block;
```

Código 5.4: Procedimiento de escritura

La parte de la escritura que se realiza de forma atómica, correspondiente a la actualización del bloque, está definida dentro del segundo objeto protegido que, como hemos dicho, es el que trabaja con máxima prioridad. Puede verse este objeto protegido, EEPROM_Update, en el código 5.5.

```
1     protected EEPROM_Update is
2        pragma Priority (System.Interrupt_Priority'Last);
3        procedure Update_Block (Block_EEPROM : in
    Stream_Element_Array_512; Block_Address : System.
    Address);
4     end EEPROM_Update;
```

Código 5.5: Operaciones del objeto protegido EEPROM_Update

Como hemos visto, este objeto protegido únicamente realiza la operación de actualizar el bloque, que está implementada en el procedimiento Update_Block:

- **Update_Block:** Este procedimiento, cuyo código se encuentra en el fragmento de código 5.6, corresponde con la segunda parte de la operación de escritura, es el encargado de la actualización de la dirección de memoria con el bloque que recibe como parámetro, también recibe la dirección final en la que debe copiarse, que se ha calculado en el procedimiento Write_Block.

  La copia del bloque en memoria se hace recorriendo el bloque que se quiere copiar y escribiendo a nivel de Byte su contenido en la dirección correspondiente.

  Si los bloques fueran de tamaño fijo múltiplo de 4, una implementación más adecuada sería realizar la escritura a nivel de palabra. Debido a que aún no se

ha definido el tamaño óptimo para los bloques a copiar se ha preferido realizar la copia a nivel de byte. De esta forma, si en un futuro los bloques son de tamaño variable no habrá que realizar modificaciones en esta parte del código.

```
1    protected EEPROM_Update is
2      pragma Priority (System.Interrupt_Priority'Last);
3      procedure Update_Block (Block_EEPROM : in
   Stream_Element_Array_512; Block_Address : System.
   Address);
4    end EEPROM_Update;
```

Código 5.6: Procedimiento de actualización

- **Read_Block:** La lectura del bloque se ha implementado como una función en vez de un procedimiento, ya que debe devolverse el contenido del bloque asociado al identificador que recibe como parámetro. Como puede verse este parámetro se devuelve por referencia.

  Lo primero que se hace es el cálculo de la dirección de memoria en la que se encuentra dicho bloque, siguiendo la fórmula 5.1.

  Sabiendo ya la dirección de memoria que se quiere leer se procede a comprobar si esta dirección contiene un bloque o por el contrario se encuentra vacía. En el primer caso se procederá a devolver el contenido de ésta y finalizar. En caso contrario se deberá lanzar la excepción que avisa de que el bloque que se intenta leer esta vacío.

  Puede verse la implementación de la lectura a continuación en el fragmento de código 5.7.

```
1      function Read_Block (Block : in Type_Block_Id)
   return Stream_Element_Array_512 is
2         Reading_Address : System.Address := System'
   To_Address (Interfaces.Unsigned_32 (Block*512) +
   Parameters.Base_Address_EEPROM);
3         Block_Readed : Stream_Element_Array_512;
4         for Block_Readed'Address use Reading_Address;
5
6      begin
7         if Is_Block_Used(Block) then
8            return Block_Readed;
9         else
10           raise Block_Not_Exists_Error;
```

```
11          end if;
12       end Read_Block;
```

Código 5.7: Procedimiento de lectura

- **Free_Block:** El último procedimiento que realiza este objeto protegido es la liberación de un bloque de memoria, recibiendo como parámetro de entrada el identificador del bloque a liberar.

  Es un procedimiento sencillo que únicamente ha de comprobar que el bloque esté ocupado, y en caso afirmativo marcarlo como libre. En el caso de que el bloque que se quiera liberar esté libre deberá avisarse con la misma excepción de bloque libre que se realiza en la lectura.

  Puede verse la implementación de este procedimiento en el fragmento de código 5.8.

```
1
2       procedure Free_Block (Block: in Type_Block_Id) is
3       begin
4          if Is_Block_Used (Block) then
5             Is_Block_Used (Block) := False;
6
7          else
8             -- The block does not exists
9             raise Block_Not_Exists_Error;
10         end if;
11      end Free_Block;
```

Código 5.8: Procedimiento de liberación

5.4. Pruebas realizadas

Para poder comprobar el funcionamiento del manejador se han codificado varias pruebas que verifican que las operaciones se realizan sin ningún problema. Estas pruebas también comprueban que las excepciones se activan cuando se intentan realizar operaciones erróneas.

Se han implementado diversas pruebas partiendo todas del mismo diseño y difiriendo únicamente en los casos de prueba a realizar. A continuación se presentan los casos de prueba que se han definido, así como los resultados obtenidos en cada uno de ellos.

5.4.1. Prueba 1: Escritura, lectura y liberación de bloques

- **Descripción:** Esta prueba consiste en la realización de varias escrituras de bloques que a continuación se leen. En este caso se ha decidido mostrar por pantalla el contenido leído, de esta forma podremos comprobar si es correcto. Por último se realiza la liberación de los bloques.

- **Resultado:** Para comprobar el correcto funcionamiento de las escrituras se hace un volcado de la memoria mediante el grmon2, con este volcado se comprueba que lo que está escrito es lo que se pretendía escribir.

  La comprobación de que las lecturas son correctas es más sencilla, con comprobar que el contenido mostrado por pantalla corresponde realmente a los bloques escritos verificamos que se hace de forma correcta.

  Para comprobar que la liberación de un bloque se hace satisfactoriamente se realizan operaciones de lectura en la que se intenta leer un bloque ya borrado. Al intentar leer un bloque marcado como vacío obtenemos la excepción adecuada, Block_Not_Exists_Error, comprobando el correcto funcionamiento tanto de la operación de liberar los bloques como del aviso al intentar leer un bloque vacío.

5.4.2. Prueba 2: Múltiples escrituras y lecturas

- **Descripción:** Esta prueba consiste en la realización de múltiples escrituras y lecturas sobre bloques disintos. Es importante tener una prueba que realice múltiples llamadas a estas operaciones para ver el funcionamiento del manejador en caso de saturación.

También se comprueba el tiempo empleado en las operaciones. Es necesario comprobar que las escrituras realizan la espera de los 15 milisegundos y también que el tiempo empleado en las lecturas sea adecuado.

Nótese que el volcado por pantalla del contenido de los bloques se realiza utilizando la herramienta grmon2, que trabaja sobre una línea serie a 9600 baudios. Por lo tanto, teniendo en cuenta que cada carácter tarda un milisegundo en transmitirse, no basta con comprobar el tiempo que tarda en devolverse el contenido, ya que es mayor que el tiempo en realizar la operación de lectura.

- **Resultado:** Con esta prueba se obtiene tanto el tiempo dedicado en las escrituras como en las lecturas. En las escrituras recibimos un tiempo de 0.304 segundos, teniendo en cuenta que realizamos la escritura de 20 bloques, obtenemos un tiempo de 15,2 milisegundos, es decir, se emplean 200 microsegundos en cada operación de escritura, cumpliéndose la restricción de la espera en dichas operaciones.

  Las operaciones de lectura, al no requerir de esta espera se realizan en un tiempo mucho menor, siendo este tiempo 0.004 segundos para un total de 20 bloques, se tarda por tanto un tiempo de 200 microsegundos para leer un bloque de memoria.

5.4.3. Prueba 3: Memoria llena

- **Descripción:** Es importante comprobar que no se realizan escrituras fuera del espacio de reservado en la memoria.

  Para la realización de esta prueba se ha creado un bucle que realiza llamadas a la operación de escritura hasta saturar la memoria. Como el espacio reservado es 1 MB y los bloques son de 512 bytes se tienen 2048 bloques. Para poder optimizar la realización de ésta prueba se ha optado por hacer uso de los parámetros de configuración reduciendo por tanto la memoria reservada a 10 KB. Con este cambio solo tenemos que escribir más de 20 bloques y ya deberíamos obtener la excepción que nos avisa de que, al intentar escribir el último bloque, la memoria esta llena.

- **Resultado:** Se obtiene un resultado satisfactorio, y la excepción esperada para este caso, Memory_Full_Error. De este modo se demuestra que no se realizan escrituras fuera del espacio reservado.

5.4.4. Prueba 4: Bloques de tamaño mayor al permitido

- **Descripción:** Con la realización de esta prueba se quiere comprobar el funcionamiento del manejador en el caso de que los bloques recibidos sean de mayor tamaño al definido. Para ello se realiza una operación de escritura de un bloque de 600 bytes de tamaño.

- **Resultado:** Se muestra por pantalla la excepción Block_Size_Error comprobando de este modo que no es posible la operación con bloques de mayor tamaño al permitido.

Capítulo 6

Arranque desde memoria no volátil

Todo el software necesario para el funcionamiento del satélite ha de ser cargado en la memoria no volátil del OBC. El código se almacenará en los espacios reservados para ello en la memoria EEPROM tal y como se contempla en la figura 6.1.

Como se ha visto en el capítulo 4, alguna de las pruebas de validación realizadas simulaban el funcionamiento del satélite en modo vuelo, por este motivo, y debido a que será necesario cargar todo el software una vez terminado se hace necesaria la comprobación del correcto funcionamiento del arranque del OBC desde la memoria EEPROM.

Como se ha explicado en el capítulo 3 la herramienta que se ha utilizado en este caso ha sido el mkprom2. Esta herramienta es la encargada de, a partir del código, realizar un fichero comprimido de tipo ejecutable y cargarlo en memoria no volátil. Para ello se deben determinar los parámetros necesarios para que este fichero ejecutable sea el adecuado para el OBC, principalmente se determina el controlador de memoria y el tipo de procesador utilizado, en este caso en LEON3.

Mkprom2 comprime el código utilizando una modificación del algoritmo de compresión LZSS (Lempel–Ziv–Storer–Szymanski). El algoritmo LZSS es un algoritmo de compresión de datos sin pérdida en la que se intenta reemplazar una cadena de símbolos por una referencia a una localización del diccionario de la misma cadena.

Aunque el fichero comprimido se carga en la memoria EEPROM, cuando el satélite arranca este debe transferirse a la memoria RAM (Random Access Memory) del mismo, véase figura 6.1. Ésto es debido a la diferentes velocidades con las que trabajan ambas memorias, la memoria EEPROM es mucho más lenta que la memoria RAM y por tanto el satélite debe usar la memoria RAM a la hora de realizar las operaciones.

El ejecutable cuando está almacenado en la memoria no volátil solamente contiene texto mientras que al ser descomprimido y transferido a la RAM contendrá

tres zonas de memoria: texto, datos y bss. El texto corresponde al código, la zona de datos contiene las variables globales y el bss alberga las pilas de las tareas.

Mkprom2 almacena en la memoria no volátil tanto el ejecutable comprimido como un descompresor para éste. Cuando se produce el arranque se inicia el computador realizando la inicialización del controlador de memoria. A continuación se descomprime el fichero ejecutable y se realiza la transferencia del código desde la EEPROM a la RAM, donde se realizará la ejecución del código.

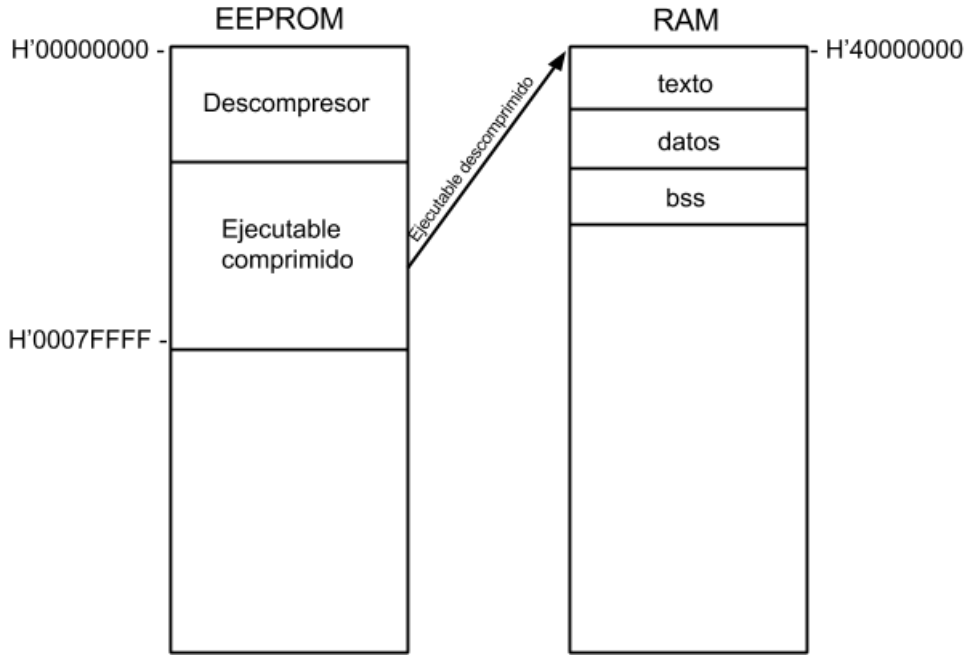

Figura 6.1: Esquema del arranque desde memoria EEPROM

Capítulo 7

Conclusiones

Como se ha podido observar a lo largo de todo el documento se ha realizado un importante trabajo que sin duda aporta valor al proyecto en el que se encuentra enmarcado. Debido al alcance y la importancia del proyecto UPMSat-2 todo el desarrollo del software que se ha hecho se ha acompañado de pruebas de verificación y validación de los subsitemas.

Se han logrado desarrollar satisfactoriamente las tres tareas principales que componen este trabajo: las pruebas de validación, el desarrollo del manejador de la memoria no volátil y el arranque del OBC desde esta memoria.

Todas estas tareas podrán ser integradas en cuanto sea necesario con el resto de sistemas, ya que los resultados obtenidos con las diferentes pruebas realizadas garantizan que se cumplen las especificaciones de las mismas. Así mismo se puede decir que todos los objetivos descritos al principio del presente documento han sido cumplidos.

Es importante destacar que este trabajo pone a disposición del proyecto UPMSat-2 dos piezas fundamentales para el funcionamiento del micro-satélite, puesto que no sería posible realizar un lanzamiento del satélite si el sistema no pudiese arrancar desde la memoria EEPROM y no se facilitase, gracias al manejador, la interacción con la misma.

Por ello, el código desarrollado por la autora ha sido incluido al sistema de control de versiones del proyecto y será licenciado en el futuro, junto con el resto del software del proyecto, bajo una licencia LGPL (Lesser General Public License) [8].

A nivel personal considero que la realización de este trabajo ha sido muy beneficiosa, ya que me ha permitido realizar una aplicación de los conocimientos adquiridos durante mi formación en un entorno profesional de gran relevancia.

El desarrollo de este trabajo me ha permitido desarrollar nuevas capacidades como la programación para sistemas de tiempo real, creyendo muy oportuno el uso del lenguaje de programación Ada para dicho fin, o la realización de pruebas de validación que comprueban el funcionamiento del software y hardware al mismo tiempo.

Por último, tener acceso a un entorno de sistemas aeroespaciales ha hecho posible adquirir nuevos conocimientos sobre este área, y poder aplicar mis conocimientos y experiencia previa a este campo.

Capítulo 8

Líneas futuras

A pesar de que el trabajo realizado es totalmente válido y útil se divisan algunas optimizaciones que permiten hacer mejoras a la implementación realizada.

En primer lugar y aunque el manejador desarrollado funciona correctamente y no se han descubierto errores durante las pruebas realizadas, es conveniente realizar una segunda versión que mejore, no su funcionamiento, sino su comportamiento respecto al uso de recursos. En concreto, uno de los cambios introducidos en esta nueva versión debe ser la optimización del uso de la memoria.

Con la versión actual del manejador desarrollado existe la posibilidad de que no se utilice todo el espacio reservado de la memoria EEPROM, ya que al escribir se busca el primer bloque libre que hay en memoria. Debido a las características de la EEPROM, esta memoria se degrada con las escrituras y por tanto se quiere conseguir un uso equitativo de todo el espacio reservado, consiguiendo que el número de operaciones de escritura se reparta uniformemente a lo largo de toda la memoria.

En un futuro puede ser que asignemos bloques de longitud fija aunque éstos no se usen completamente. Por tanto, habrá que implementar una forma de almacenar, además del estado (ocupado o libre) del bloque, la cantidad real de información que contiene dicho bloque.

Una vez más, el cambio introducido en la nueva versión responde a razones de optimización, puesto que permitirá adaptar las operaciones de escritura al tamaño de los datos sin desperdiciar espacio de memoria.

De momento se ha implementado el arranque de la memoria no volátil con una copia del código ejecutable. Como trabajo futuro se cargarán dos copias del código. Además, se creará una rutina encargada de comprobar la integridad del código cargado en EEPROM antes de su transferencia a RAM. Deberá, por tanto, ejecutarse justo después de que se inicie el computador tras el arranque.

El funcionamiento de esta rutina deberá hacer una comprobación del código, para ello las copias del código se almacenarán con una suma de comprobación. Esta comprobación se realizará mediante una suma de verificación, que en caso de indicar que el código ha sufrido algún cambio pasará a realizar la misma comprobación sobre el código de respaldo cargado al final de la memoria no volátil.

En el caso de que ambas fracciones de código no sean válidas se deberá transferir una de ellas a la memoria RAM para que el satélite puede arrancar y se proceda a la carga de un código válido desde tierra.

Naturalmente una línea futura del UPMSat-2 es completar todo el resto del software de alto nivel y realizar pruebas de unidad y de integración de todo el software en su conjunto.

Bibliografía

[1] John Barnes. *Programming in Ada 2012*. Primera edición edition, 2014.

[2] Instituto de microgravedad Ignacio Da Riva. Proyecto upmsat-2. `http://www.idr.upm.es/tec_espacial/06_UPMSAT.html`.

[3] Renesas Electronics. *Renesas Technology EEPROM HN58V1001 Series*. 2010.

[4] Aeroflex Gaisler. *MKPROM2 Overview*. 2014.

[5] Aeroflex Gaisler. *Grmon2 User's Manual*. 2015.

[6] Juan A. de la Puente Alejandro Alonso Jorge Garrido, Juan Zamorano and Emilio Salazar. *Ada, the programming language of choice for the UPMSat-2 satellite*. 2015.

[7] Jorge López Juan A. de la Puente Juan Zamorano. *Guidelines for integrating device drivers in the assert virtual machine*. ESA/ESTEC, 2009.

[8] GNU Lesser General Public License. Licencia lgpl. `http://www.gnu.org/licenses/lgpl.html`.

[9] Gaisler Research. *LEON3 Product Sheet*. 2008.

[10] STRAST. Open ravenscar kernel. `http://web.dit.upm.es/~str/ork/index.html`.

[11] STRAST. Proyecto upmsat-2. `http://web.dit.upm.es/~str/proyectos/upmsat2/`.

[12] Tecnobit. *Manual de Usuario*. 2015.

[13] Tecnobit. *Pruebas Funcionales EBOX*. 2015.

[14] Alan Burns y Andy Wellings. *Sistemas de Tiempo Real y Lenguajes de Programación*. Tercera edición edition, 2003.

[15] José Meseguer Ruiz y Angel Sanz Andrés. *El satélite UPM-Sat 1*. 1998.

[16] Juan Zamorano and Jorge Garrido. *Schedulability analysis of PWM tasks for the UPMSat-2 ADCS*. 2015.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Jun 05 17:58:45 CEST 2015
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)