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

Diseño de carga de pago para satélite de Misión Alpha

Autor: Enrique Martínez Calvo
Tutores: Fernando Muñoz Chavero
José María Hinojo Montero

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías de Telecomunicación

Diseño de carga de pago para satélite de Misión Alpha

Autor:
Enrique Martínez Calvo

Tutores:
Fernando Muñoz Chavero
Catedrático de Universidad

José María Hinojo Montero
Investigador Postdoctoral

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Grado: Diseño de carga de pago para satélite de Misión Alpha

Autor: Enrique Martínez Calvo

Tutores: Fernando Muñoz Chavero
José María Hinojo Montero

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

A mis amigos

A mis maestros

Agradecimientos

Escribiendo estas líneas doy por finalizada una etapa complicada en mi vida, pero llena de aprendizajes y personas a las que debo muchas cosas, por ello aquí les dedico unas líneas con mi más profundo agradecimiento:

En primer lugar, quiero dar las gracias a Fernando por su gran labor docente e investigadora en nuestra Escuela y por darme la oportunidad de colaborar en un proyecto tan interesante como este.

A José le debo un inmenso agradecimiento, puesto que su ayuda siempre ha estado disponible cada vez que la he necesitado y se ha preocupado por cada detalle de este trabajo, a pesar del escaso tiempo del que dispone. No puedo sentirme más agradecido de haber contado con su tutorización y todo el aprendizaje que ello conlleva.

Al Grupo de Ingeniería Electrónica, por todo lo que he aprendido trabajando aquí y la compañía durante estos meses.

A la Universidad de Sevilla y a la ETSI, por toda la educación y aprendizajes que he recibido durante estos años.

A la Delegación de Estudiantes de la ETSI, que me ha ayudado a formarme en otros ámbitos muy distintos, pero igual de importantes a los vistos en clase, donde he tenido la suerte de conocer a muchas personas implicadas por mejorar su entorno y ofrecer ayuda desinteresada a los demás y donde he forjado amistades que me quedarán para toda la vida.

Al Consejo Estatal de Estudiantes de Telecomunicación, porque me ha brindado la oportunidad de conocer a estudiantes de toda España con mis mismas inquietudes e interés por la representación estudiantil.

A todos los amigos que la carrera me ha ofrecido, en especial a Silvia y Antonio. Ha sido un placer compartir tantas horas de estudio y trabajo con vosotros.

A Luis, porque juntos hemos crecido y hemos aprendido muchísimo el uno del otro, por haber compartido conmigo prácticamente cada día en la ETSI, por todas las comidas en la Fcom llenas de risas y buena compañía, por las semanas de estudio al fallo encerrados con el Tiny Desk del Cigala en bucle, por los viajes del CEET... Gracias por haber vivido este camino conmigo, ahora a por el siguiente.

A María, porque su compañía ha sido imprescindible en estos años, por sufrir conmigo mis agobios y haberme permitido compartir con ella esta etapa tan valiosa para ambos.

A mi hermana Isabel, por todo su cariño y sus ánimos (muchas veces a su manera).

Finalmente, a mis padres, porque sin ellos no estaría escribiendo esto. Me han ofrecido todo lo necesario para que haya podido llegar hasta aquí, siempre han apostado por la educación y me han hecho ser la persona que soy hoy en día. Es a ellos a quienes debo todo esto.

Enrique Martínez Calvo

Sevilla, 2024

Resumen

La llegada de los satélites de pequeño tamaño o ‘CubeSats’ a la industria espacial, ha democratizado el acceso a la órbita terrestre, reduciendo significativamente los costes de desarrollo y despliegue de dispositivos con finalidades educativas, de experimentación científica o de demostración de tecnologías.

En 2022, una serie de empresas andaluzas junto con las universidades de Cádiz y Sevilla, lanzaron la iniciativa: ‘Misión Alpha’, un proyecto para desarrollar y poner en la órbita baja terrestre un satélite genuinamente andaluz, con el objetivo de generar datos científicos de diferentes experimentos realizados en el satélite y, por otro lado, visibilizar el concepto de ‘Newspace’, una industria emergente a nivel global en la que empresas privadas desarrollan tecnologías y servicios espaciales con un enfoque innovador, competitivo y accesible en comparación a grandes agencias espaciales gubernamentales.

Desde la Universidad de Sevilla, el Grupo de Ingeniería Electrónica se ha encargado de diseñar la carga de pago que realizará diversos experimentos para caracterizar la radiación que recibe un satélite a lo largo de su vida útil en órbita baja terrestre. La carga de pago hace uso del microcontrolador ATSAM3X8E de Atmel para gestionar la toma de medidas por parte de los diferentes sensores y posteriormente enviar la información al ordenador de a bordo del satélite, encargado de transmitir la información a la estación terrena.

En este trabajo, se expondrá el proceso de desarrollo de los drivers encargados de gestionar la comunicación a través del bus CAN con el ordenador de a bordo, la gestión de los modos de bajo consumo del microprocesador y el desarrollo de un sistema operativo en tiempo real para la gestión de las distintas tareas llevadas a cabo por el sistema.

Abstract

The arrival of small satellites or 'CubeSats' to the space industry has democratized access to Earth orbit, significantly reducing the costs of development and deployment of devices for educational purposes, scientific experimentation or technology demonstration.

In 2022, a number of Andalusian companies together with the universities of Cadiz and Seville, launched the initiative: 'Alpha Mission', a project to develop and put into low Earth orbit a genuinely Andalusian satellite, with the aim of generating scientific data from different experiments conducted on the satellite and, on the other hand, to make visible the concept of 'Newspace', an emerging industry globally in which private companies develop space technologies and services with an innovative, competitive and accessible approach compared to large government space agencies.

From the University of Seville, the Electronics Engineering Group has been in charge of designing the payload that will perform several experiments to characterize the radiation received by a satellite throughout its lifetime in low Earth orbit. The payload makes use of Atmel's ATSAM3X8E microcontroller to manage the measurements taken by the different sensors and then send the information to the satellite's on-board computer, which is responsible for transmitting the information to the ground station.

In this work, the development process of the drivers in charge of managing the communication through the CAN bus with the on-board computer, the management of the low consumption modes of the microprocessor and the development of a real-time operating system for the management of the different tasks carried out by the system will be presented.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

"Cualquier tecnología lo suficientemente avanzada es indistinguible de la magia."

-Arthur C. Clarke-

Durante este capítulo se hará una breve descripción de distintos términos y conceptos importantes para entender el contexto de la Misión Alpha y la carga útil desarrollada por el Grupo de Ingeniería Electrónica de la Universidad de Sevilla. También, se detallarán los objetivos a cumplir en el presente documento y se explicará la organización de este.

1.1. Contexto del Proyecto

1.1.1. NewSpace

Los grandes logros conseguidos en la industria espacial, como la primera expedición del ser humano al espacio, la Estación Espacial Internacional o la carrera espacial, han sido objetivos motivados por intereses científicos y geopolíticos. Por este motivo, estas misiones, debido a su alto coste económico, fueron financiados en su práctica totalidad por las grandes agencias gubernamentales, por ejemplo, la NASA y/o la ESA. Debido a dichos propósitos, la rentabilidad económica era algo secundario, invirtiéndose grandes cantidades de dinero en desarrollar sistemas robustos, fiables y con garantías de no provocar errores fatales durante el transcurso de las misiones.

El concepto de NewSpace, propone un cambio de paradigma en la financiación de estas misiones, donde inversores privados puedan acceder a nuevos modelos de negocio basados en el espacio, sin tener que enfrentarse a las grandes barreras de entrada que suponen acceder a la industria. Esto no implica la comercialización de servicios espaciales, puesto que es algo que se lleva haciendo desde comienzos de la carrera espacial con satélites empleados para retransmisiones televisivas o sistemas de telecomunicaciones.

NewSpace pretende ser un cambio cultural hacia una mayor participación privada en el sector, pero también un cambio en la búsqueda y explotación de nuevos mercados de servicios, incluso fuera de la industria espacial. En estos mercados, los recursos espaciales no serían los únicos ni principales proveedores de servicios, sino que se convertirían en uno más de los diversos proveedores que ofrecen soluciones para una diversa gama de aplicaciones, como industriales o educativas.

La financiación, como ya se ha mencionado, es un factor clave en el NewSpace. Las empresas que forman parte de esta filosofía trabajan con acuerdos de precio fijo, lo que supone un mayor incentivo para incrementar la eficiencia operacional y, por tanto, maximizar los beneficios contando con recursos finitos. Esto implica buscar nuevos enfoques a nivel de ingeniería, diferentes a los tradicionales en los que la aversión del riesgo es una prioridad. En el caso del NewSpace, la aceptación del fracaso forma parte de la mentalidad de las empresas, que apuestan por un desarrollo más ágil, buscando el mayor rendimiento empresarial.

Esto se consigue gracias al uso de componentes conocidos como COTS (Commercial Off The Shelf), los cuales no están específicamente diseñados para funcionar en entornos hostiles como el espacio y son más propensos a sufrir daños fatales debido a las condiciones extremas en las que operan. No obstante, los beneficios que aporta el disponer de tecnología reciente en el espacio unido a la mayor robustez que presentan los nuevos procesos de fabricación a las condiciones de operación esperadas en el espacio, hacen que haya un gran interés por validar COTS para su uso en misiones de corta duración. Además, se buscan técnicas de redundancia que permitan desarrollar sistemas robustos frente a posibles eventos críticos que afecten a la funcionalidad del satélite.

Asimismo, siempre es importante hacer un balance de lo que supone el uso de este tipo de componentes, teniendo en cuenta los recursos que se emplean en realizar pruebas y el nivel de riesgo que implica el uso de determinados componentes.

En resumen, el NewSpace se podría describir como una nueva filosofía empresarial, en la que empresas que aspiran a buscar un hueco en el sector espacial puedan optar a ello, a base de desarrollos impulsados por el rendimiento económico, haciendo uso de tecnologías ampliamente comercializadas, que les permitan innovar y desarrollar nuevos mercados de servicios basados en el espacio. [1], [2]

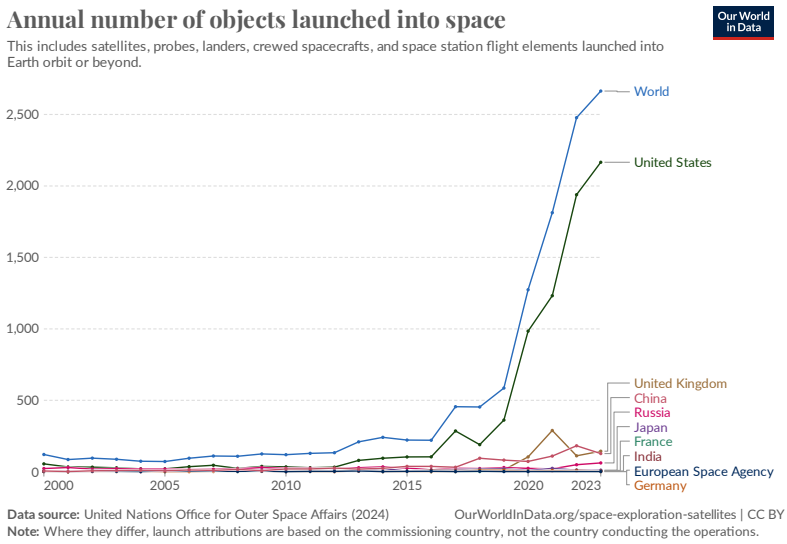

Figura 1-1. Número de objetos lanzados anualmente al espacio desde el año 2000 [3].

1.1.2. Órbita Baja Terrestre

La Órbita Baja Terrestre, en inglés LEO (Low Earth Orbit), es una órbita alrededor de la Tierra que abarca un rango el cual no está acordado, pero que se suele establecer entre los 150 km y los 2000 km de altitud respecto a la superficie terrestre. A esta distancia, los satélites orbitan con un periodo aproximado de 130 minutos o 11,25 vueltas al globo terrestre por día [4].

El situar un satélite en órbita LEO supone diversas ventajas respecto a órbitas de mayores altitudes debido, principalmente, al coste; puesto que se requiere menor energía para situarlo en órbita. Además, los sistemas de teledetección y comunicación son menos complejos, requiriendo una menor infraestructura.

Un aspecto fundamental en las misiones espaciales que operan en esta órbita es la elección de la altitud, puesto que no es una decisión trivial y hay que tener en consideración ciertas restricciones que determinarán la duración de la misión.

1. Efecto de arrastre atmosférico a altitudes bajas.
2. Cinturones de Van Allen. Zonas de acumulación de partículas altamente cargadas que afectan significativamente a la electrónica [5]. Estas zonas se encuentran a altitudes superiores a los 1500 km.

Por lo general, para garantizar la viabilidad económica, se suelen usar trayectorias orbitales superiores a los 500 km, en las que aún con cierta influencia de la atmósfera, se consiguen misiones con duraciones cercanas a los 2 años.

1.1.3. CubeSat

CubeSat, es un estándar para el diseño de satélites de pequeño tamaño, en él se definen entre muchos otros parámetros, la unidad mínima para la construcción de satélites que usen el estándar CubeSat, siendo esta unidad mínima, cubos de 10 cm de arista. A estas unidades se les denomina ‘U’. Se pueden unir varios de estos cubos entre ellos para formar satélites de mayor tamaño que permitan alojar un mayor número de componentes.

El concepto fue ideado en 1999 por el ingeniero español Jordi Puig-Suari y el ingeniero estadounidense Bob Twiggs y surgió como una idea para "democratizar el espacio" [6], haciendo accesible el desarrollo de tecnología espacial en países donde estudiar el espacio era algo por aquel entonces inalcanzable.

Hoy en día, el uso de CubeSats se asocia al concepto de NewSpace; no obstante, cómo ya se ha explicado anteriormente, el NewSpace se debe entender como una filosofía, un cambio de rumbo en la industria del sector espacial. Los satélites de pequeño tamaño son sólo una de las consecuencias de este cambio de paradigma, puesto que permiten a las empresas desarrollar sus tecnologías a precios competitivos y en plazos de tiempo más ajustados.

Figura 1-2. CubeSat SamSat-ION con formato 3U, desarrollado por investigadores de la universidad de Samara [7].

1.1.4. Misión Alpha

La Misión Alpha, [8], nace como un proyecto impulsado por distintas empresas andaluzas junto con dos universidades andaluzas (Universidad de Sevilla y Universidad de Cádiz) para lanzar al espacio el primer satélite genuinamente andaluz en el año 2025. Los objetivos principales de esta iniciativa son:

- Generar todo el conocimiento e infraestructura necesaria para poder desarrollar y poner en órbita un satélite, dotando de los procedimientos necesarios a las empresas involucradas
- Llevar a cabo dos experimentos científicos que ponga de manifiesto la utilidad y versatilidad de estos satélites y pueda ofrecer a la comunidad científica datos relevantes para futuras investigaciones.
- Recaudar información de utilidad para el desarrollo de misiones futuras más eficientes, seguras y fiables.
- Fomentar el sector aeroespacial en la región.

En relación con los experimentos científicos, la Misión Alpha incluye una carga útil para medir con precisión campos magnéticos de muy baja frecuencia. Este experimento es liderado por la Universidad de Cádiz. El segundo experimento corresponde con una carga útil que permite estimar los niveles de radiación en una órbita baja terrestre. Esta segunda carga útil ha sido desarrollada en el seno del Grupo de Ingeniería Electrónica, perteneciente a la Universidad de Sevilla. Cabe destacar que los datos recopilados por sendos experimentos serán puestos a disposición de la comunidad científica de forma gratuita para su uso [6].

Por otro lado, la misión pretende promover la importancia de la nueva economía que plantea seguir la corriente del NewSpace y finalmente, inspirar a futuras generaciones a interesarse por el sector espacial, mostrando las distintas disciplinas desde las que se puede contribuir a la industria.

El satélite en sí usará el estándar CubeSat y será de tamaño 3U, para poder alojar las cargas de pago correspondientes a el ordenador de a bordo, baterías y los experimentos desarrollados por las universidades de

Cádiz y Sevilla. Estos experimentos recaudarán una serie de datos los cuales serán enviados a través del ordenador de a bordo a una estación terrena, para poder procesar los datos e interpretarlos en tierra.

1.2. Objetivos

A continuación, se detallará el alcance del presente Trabajo Fin de Grado, el cual se enmarca en el trabajo desarrollado por el Grupo de Ingeniería Electrónica en la Misión Alpha y se ha centrado en el desarrollo de las comunicaciones entre la carga útil y el ordenador de a bordo. Los objetivos del trabajo son los siguientes:

1.2.1. Definición de las estructuras de datos necesarias

Es necesario que la comunicación entre el ordenador de a bordo y la carga útil siga una determinada estructura en las que se identifique el tipo de datos que contiene cada mensaje, los datos en sí y otra información relevante como la marca de tiempo o un número de secuencia.

1.2.2. Desarrollo de un driver de Comunicaciones CAN

Se desarrollará un controlador CAN para poder implementar una comunicación usado las estructuras de datos previamente definidas.

1.2.3. Definición de la lógica de comunicación

Durante la misión, pueden producirse fallos de funcionamiento que interrumpan la comunicación con la consecuente pérdida de mensajes. Por ello se definirán los pasos a seguir en caso de que se produzca un fallo en la comunicación.

1.2.4. Integración del sistema completo

Usando un sistema operativo de tiempo real, se integrará el controlador CAN desarrollado para poder seguir la lógica de comunicación diseñada y tener un sistema funcional.

1.2.5. Verificación del sistema completo

Se realizarán distintas pruebas para comprobar que el sistema se comporta acorde a lo planteado, haciendo uso de dispositivos que permitan simular un ordenador de a bordo.

1.3. Organización del documento

El presente documento consta de un total de 6 capítulos, además de los anexos.

El primer capítulo, Introducción, muestra una visión general del proyecto y su contexto. Además, recoge los objetivos marcados en el presente Trabajo Fin de Grado.

En el segundo capítulo, se hace una breve descripción de la carga útil desarrollada por el Grupo de Ingeniería Electrónica de la Universidad de Sevilla. Se detallan los efectos de la radiación espacial que serán estudiados y los componentes empleados para llevar a cabo los experimentos.

En el tercer capítulo se explican con mayor detalle la comunicación con el OBC, el funcionamiento del protocolo CAN, la implementación en el microcontrolador y los distintos mensajes que intercambiarán carga útil y OBC.

El cuarto capítulo consiste en la descripción del sistema completo, se explica el concepto de sistema operativo de tiempo real y su utilidad en el proyecto. Una vez explicado dicho concepto, se detalla la arquitectura completa del sistema y se explica el controlador CAN desarrollado.

En el quinto capítulo se hace una recopilación de las pruebas de verificación realizadas.

El último apartado contiene las conclusiones del trabajo realizado y las posibles acciones a seguir para dar continuidad al proyecto.

2. DESCRIPCIÓN DE LA CARGA ÚTIL

De entre las distintas entidades que colaboran el proyecto, el Grupo de Ingeniería Electrónica es el encargado de desarrollar el experimento que consiste en medir los efectos de la radiación en la órbita baja terrestre sobre componentes electrónicos. En este capítulo, se describirán los distintos fenómenos a estudiar por la carga útil y se detallarán los componentes de mayor interés de la carga útil.

2.1. Efectos de la radiación a que estudiará carga útil

2.1.1. Niveles de radiación acumulada por la electrónica (TID o Total Ionizing Dose)

En el espacio, al no estar protegida por una atmósfera, la electrónica es más propensa a sufrir los efectos nocivos de la radiación cósmica. Esta radiación posee elevados niveles de energía y es considerada ionizante, por lo que es capaz de generar pares electrón hueco en la materia y, por tanto, carga eléctrica.

En el caso de tecnología MOS por ejemplo, esta acumulación de carga en las capas de óxido produce una variación de la tensión umbral de los transistores, pudiendo llegar a hacer vulnerables los circuitos que sufren esta radiación [9].

Por ello, el experimento a realizar por la Universidad de Sevilla pretende obtener datos en tiempo real para mejorar predicciones de modelos electrónicos, proponer nuevas estrategias de diseño y optimizar protecciones de los elementos esenciales para el funcionamiento del satélite como el MCU o las memorias [10].

2.1.2. Tasa de generación de eventos singulares (SEE o Single Effect Events)

Al igual que la TID, los SEE están relacionados con la radiación nociva presente en el espacio. La TID es un efecto a largo plazo, cuanto mayor tiempo esté expuesto un componente electrónico a radiación de alta energía, mayor será su degradación.

Un SEE, se puede definir como cualquier efecto medible sobre un circuito que ha sido producido por el impacto de un ion de elevada energía. Estos son eventos singulares que ocurren con cierta probabilidad, por lo que sufrir uno de estos eventos no dependerá del tiempo que el satélite este expuesto a la radiación cósmica [11].

Los SEE engloban varios tipos de estos eventos que pueden producir alteraciones en el funcionamiento de un circuito. En el caso de la carga útil desarrollada por la Universidad de Sevilla, se estudiarán los SEUs o Single Event Upset y los SETs o Single Events Transient.

Un SEU consiste en el cambio de estado de un componente digital por el impacto de radiación ionizante, se considera un efecto ‘suave’, ya que un reinicio del sistema o una reescritura del biestable/celda de memoria reestablecen el correcto funcionamiento del dispositivo.

Los SETs se producen por la inducción de un efecto transitorio no deseado en un circuito analógico, en el caso de la carga útil se analizarán picos de voltaje producidos por el impacto de iones pesados en una unión PN.

2.2. Sensores y componentes

El Grupo de Ingeniería Electrónica de la Universidad de Sevilla ha diseñado una placa con dimensiones específicas para cumplir con las restricciones de espacio del nano satélite y además garantizar que el fallo de un componente no suponga la pérdida de funcionalidad de ninguna de las partes que forman la carga útil. En la Figura 2-1 se puede apreciar la redundancia de componentes, esencial para garantizar que el fallo de uno de ellos no inhabilite el funcionamiento de la carga útil durante la misión.

En este apartado, se comentarán los componentes de mayor interés que forman parte de la carga útil.

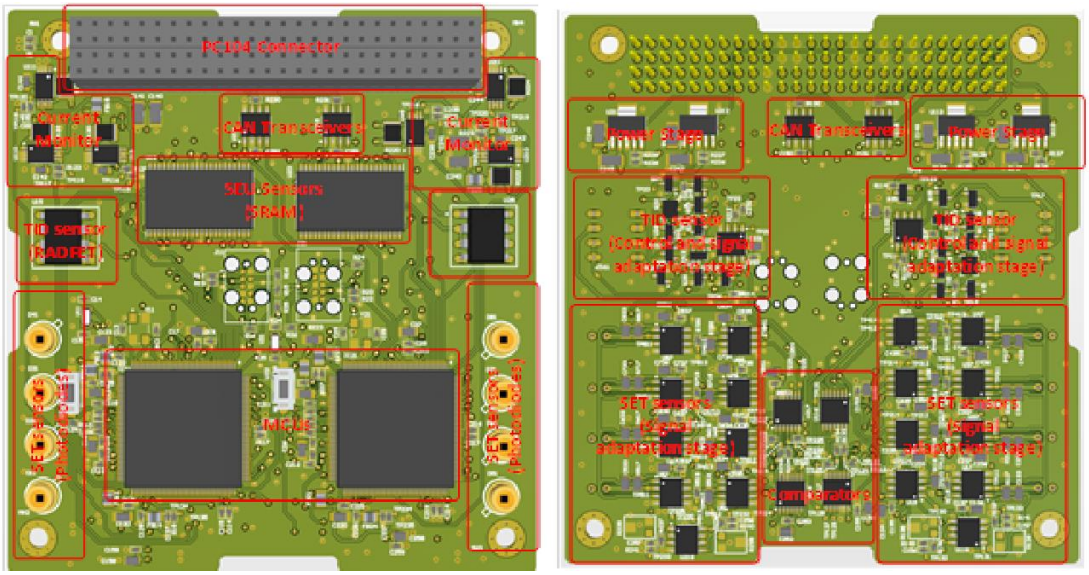

Figura 2-1. Vistas superior e inferior de la carga útil.

2.1.3. Microcontrolador SAM3X8ERT

El SAM3X8ERT es la versión resistente a radiación del microcontrolador SAM3X8E. Es un microcontrolador desarrollado por Atmel basado en tecnología ARM Cortex-M3 con arquitectura de 32 bits. Cuenta con elevada capacidad de procesamiento y bajo consumo de energía, siendo ideal para una amplia variedad de aplicaciones.

Las características principales del microcontrolador son:

- Frecuencia máxima de 84 MHz.
- Memoria flash de 512 KB dividida en dos bancos de 256 KB.
- Memoria SRAM de 96KB (1 bloque de 64KB y otro de 32KB).
- 103 pines de propósito general.
- Convertidor analógico digital de 16 canales de 12 bits.
- Convertidor digital analógico de 2 canales de 12 bits.
- 9 temporizadores de 32 bits.
- Reloj y temporizador de tiempo real (RTC y RTT).
- Múltiples interfaces de comunicación (5 UART, 2 I2C, 4 SPI, 1 CAN y 1 Ethernet MAC).
- Cuenta con 3 modos distintos de bajo consumo.
- Interfaces JTAG/SWD para programación y depuración.

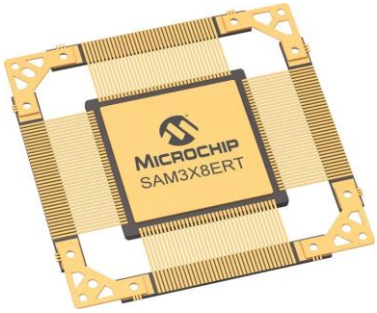

Figura 2-2. Microcontrolador SAM3X8ERT empleado en la carga útil.

2.1.4. Fotodiodos para detección de impactos de partículas

Para detectar eventos del tipo SET que se usarán cuatro fotodiodos del fabricante HAMAMATSU, modelo S1336-18BQ (Figura 2-3). Al impactar una partícula sobre alguno de los fotodiodos, se producirá un pico de tensión.

Esta señal, previamente amplificada, se conecta a uno de los pines del microcontrolador para realizar una medida de la duración del pulso y así poder estimar la energía de la partícula.

Además, junto con la medida se tomará una marca de tiempo para estimar la posición del satélite en el momento del impacto con el propósito de crear un 'mapa de radiación' a lo largo de la órbita del satélite.

Figura 2-3. Fotodiodo HAMAMATSU S1336-18BQ para la detección de SETs.

2.1.5. Sensor para medir la radiación total acumulada

La medida de la radiación total acumulada o TID se realizará usando transistores de efecto de campo sensibles a radiación o RADFETs, concretamente el modelo VT02 del fabricante Varadis (Figura 2-4).

Este sensor, consiste en un transistor MOS de canal p de alta sensibilidad, que se emplea ampliamente para medir radiación ionizante. Al recibir esta radiación, el óxido de la puerta acumula carga positiva, resultando en un aumento de la tensión umbral del transistor.

Para estimar la TID, se medirá la tensión umbral del transistor haciendo uso del convertidor analógico digital del microcontrolador.

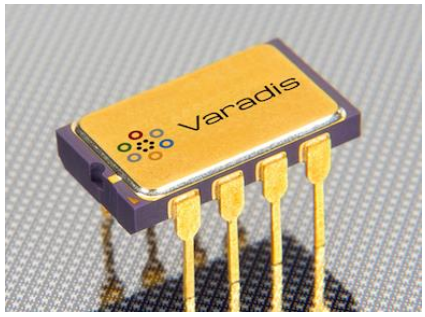

Figura 2-4. Sensor RADFET VT02 de Varadis para la medida de TID. [www.varadis.com]

2.1.6. Memoria SRAM para detección de SEUs

La detección de eventos de tipo SEU se realizará usando la memoria SRAM LY62W20488 del fabricante Lyontek con una capacidad de 2048 Kx8 bits. Es un modelo particularmente sensible a impactos de partículas.

Con esta memoria se tratará de estimar la tasa de SEUs que ocurren en órbita. Se escribirán patrones específicos en la memoria para posteriormente realizar lecturas periódicas y detectar cualquier cambio causado por un SEU.

3. COMUNICACIÓN CON EL ORDENADOR DE A BORDO

En este capítulo se hará una breve introducción al protocolo de comunicación CAN, puesto que es el elegido para la transmisión de datos entre el ordenador de a bordo y la carga útil. Se explicará el funcionamiento del estándar en el microcontrolador y por último, se describirán el tipo de mensajes a enviar y la lógica de envío empleada para evitar la pérdida de tramas.

3.1. Bus CAN

El protocolo elegido para la comunicación es el estándar CAN (Controller Area Network). Fue diseñado por la empresa Bosch en la década de los 80 con la intención de reducir la complejidad y el coste del cableado en automóviles a través de multiplexión. Desde entonces, ha sido adoptado en multitud de contextos distintos al sector automovilístico debido a la fiabilidad que aporta el uso de señales diferenciales para mitigar los efectos del ruido eléctrico que se pueda producir en la línea de transmisión.

El estándar CAN permite asignar distintas prioridades a los mensajes que se envían a través del bus, para así garantizar que los mensajes con mayor prioridad siempre lleguen a su destino. El protocolo se basa en una **comunicación serie**, **asíncrona** y **half-duplex,** lo que quiere decir que la información podrá viajar en ambos sentidos del enlace de comunicación, pero sólo en un sentido en el mismo instante de tiempo.

La comunicación se realiza mediante el uso de dos señales, 'CAN High' y 'CAN_Low'. Ambas transmiten al mismo tiempo y son diferenciales, la lógica del controlador realiza una resta de ambas tensiones y se obtiene el equivalente de una única señal digital, con la ventaja de poseer una gran inmunidad frente al ruido.

Las señales se pueden encontrar en estado **recesivo** (equivalente a un 1 en binario), en este estado ambas están a la misma tensión por lo que la tensión diferencial es 0. Por otro lado, el otro estado es el llamado **dominante** (equivalente a un 0 en binario) y en este caso, suele haber una diferencia de tensión de 2,5 V entre la señal CAN High y CAN Low.

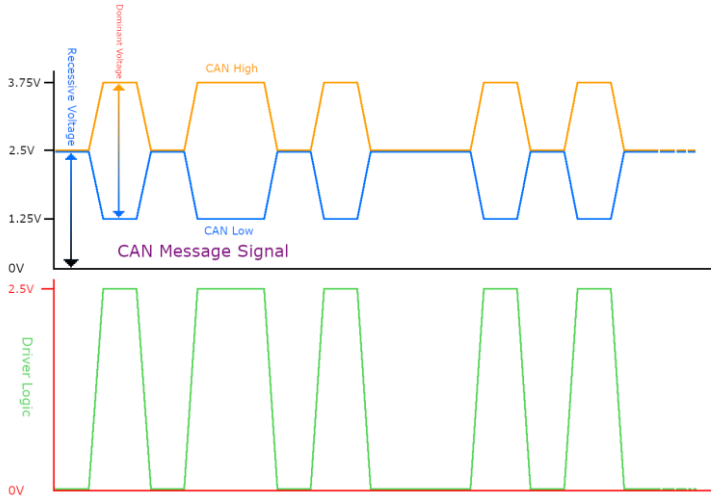

Figura 3-1. Niveles de tensión de las señales CAN High y CAN Low [12].

En el protocolo CAN se establecen prioridades de transmisión en función del identificador asignado a una trama. Estas prioridades entran en juego durante el proceso de **arbitraje** que se da al comienzo de la transmisión de cada trama. En este proceso, todos los nodos que tengan un mensaje que enviar por el bus comenzarán a transmitir los bits del identificador, cuanto menor sea el valor de este identificador, **mayor será**

la prioridad del mensaje debido a que contendrá más ceros y, por tanto, mantendrá el bus en un estado dominante durante más tiempo.

En el arbitraje, los nodos comienzan a transmitir los bits del identificador; cuando uno o varios nodos fijan un estado dominante (transmiten un 0), el resto de los nodos que hayan transmitido un 1 detectarán el estado dominante del bus y cesarán la transmisión. Así ocurrirá en la transmisión de cada bit del identificador hasta que solo quedará un único nodo transmitiendo.

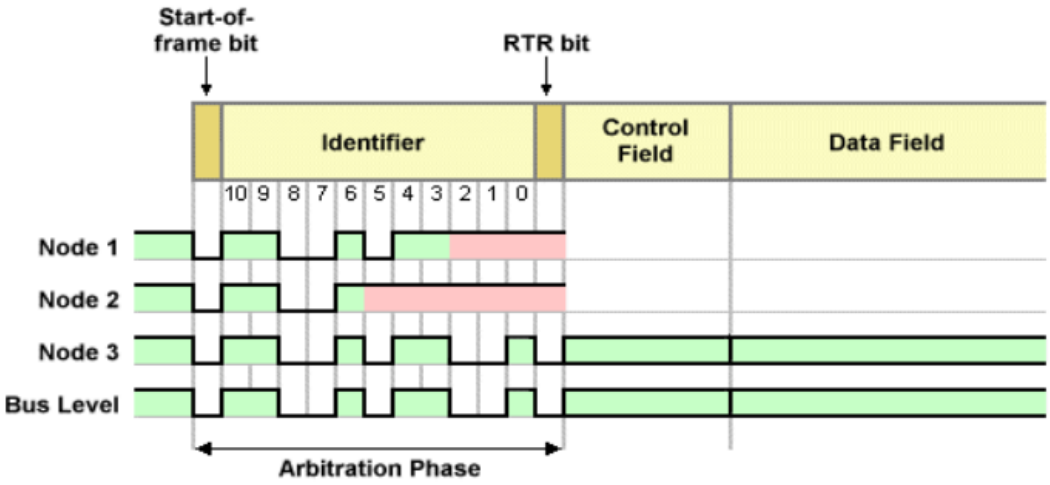

Figura 3-2. Ejemplo de nivel en un bus CAN donde 3 nodos transmiten a la vez, al poseer en nodo 3 un identificador más bajo, su nivel de prioridad es superior al resto de nodos ya que es quién fija el estado del bus [13].

Al ser un protocolo asíncrono, es crucial que todos los nodos funcionen a la misma tasa de bits por segundo. Sin embargo, debido a efectos de ruido, cambios de fase o tolerancias de los osciladores, resulta prácticamente imposible mantener constante la velocidad nominal de bits. Por ello, es necesario realizar una sincronización durante el proceso de arbitraje, ya que los nodos deben de ser capaces de transmitir y, a la vez, detectar el estado del bus.

Hay dos tipos de sincronizaciones:

- **Sincronización forzada.** Se produce en la primera transición de estado recesivo a dominante tras un periodo de inactividad (bit de inicio).
- **Resincronización.** Se produce en cada transición de recesivo a dominante durante la transmisión de la trama.

En caso de que alguna transición no se realice en el instante esperado por el controlador, este puede realizar un **ajuste puntual del tiempo de bit**, evitando errores de sincronización. Para realizar este ajuste, el estándar divide la duración de cada tiempo de bit en un número de segmentos de tiempo denominados **'Time Quanta'** o **cuantos**.

A su vez, estos cuantos se agrupan para formar 4 segmentos de tiempo que serán de utilidad para la sincronización:

- **Segmento de sincronización.** La duración es siempre de 1 cuanto. Como su nombre indica, se emplea para la sincronización de los distintos nodos.
- **Segmento de propagación.** El propósito es compensar el retardo que puede experimentar la señal dentro de la red.
- **Fase 1.** Se puede denominar un búfer de fase para compensar errores por desincronización.
- **Fase 2.** Segundo búfer de fase, al igual que el anterior, empleado para compensar errores.

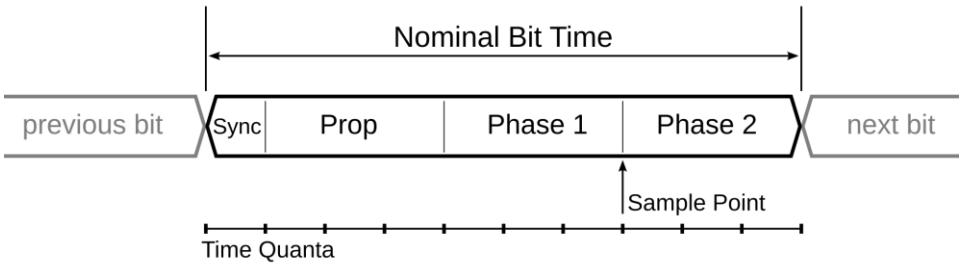

Figura 3-3. Segmentos que forman la duración del tiempo de bit en el bus CAN. [14]

La duración de estos segmentos se puede variar dependiendo de la velocidad de transmisión empleada en el bus y de los distintos nodos que formen la red. Cuando se produce un ajuste de tiempo, el controlador correspondiente podrá aumentar la duración del segmento de fase 1 o disminuir la duración de fase 2.

3.1.1. Configuración en microcontrolador SAM3X8E.

Una vez conocidos el funcionamiento y los conceptos más relevantes sobre el bus CAN, se puede configurar el controlador CAN en el SAM3X8E. Para ello, es necesario establecer la tasa de bits a la que se pretender realizar la comunicación.

Se empleará una tasa de 500 kbps. La configuración de los distintos segmentos de tiempo explicados anteriormente se realiza automáticamente por driver CAN desarrollado por Atmel en función de la velocidad de reloj del microncontrolador.

Los datos valores de configuración obtenidos son los siguientes:

Tabla 3-1. Parámetros de configuración obtenidos para el controlador CAN del SAM3X8E.

Predivisor	Tiempo de bit	Nº de cuantos	Segmento Propagación	Segmento de Fase 1	Segmento de Fase 2
21	250 ns	8 TQ	3 TQ	2 TQ	2 TQ

Cabe destacar que hay dispositivos en los que no se permite configurar la duración del segmento de propagación ya que ese tiempo se incluye en la duración del segmento de Fase 1. En caso de que se pretenda añadir a la red un dispositivo que no permita configurar este segmento, simplemente habrá que asignarle a la duración del segmento de Fase 1 la suma del segmento de propagación y del segmento de Fase 1 del resto de dispositivos que si permitan modificar este valor. A continuación, se muestra un ejemplo de este proceso:

Nodo 1 permite configurar segmento de propagación → PROP_SEG = 3 TQ, PHASE_SEG1 = 2 TQ.

Nodo 2 no permite configurar segmento de propagación → PHASE_SEG1 = 5 TQ.

3.2. Paso de mensajes entre ordenador de a bordo y carga útil

En este apartado se mostrarán los distintos tipos de mensajes que se enviarán entre carga útil y ordenador de a bordo. Posteriormente se explicará la lógica de envío y de retransmisión de datos en caso de pérdida de comunicación y finalmente, se mostrarán diagramas de paso de mensajes.

3.2.1. Tipos de mensajes

Los dos subsistemas intercambiarán diferentes tramas CAN con identificadores que permitan la correcta comunicación.

3.2.1.1. Mensajes enviados por la carga útil hacia el ordenador de a bordo

La Tabla 3-2 recoge los mensajes que se han definido para establecer la comunicación desde la carga útil al OBC. La columna 'ID' representa el identificador de la trama CAN empleado. La columna 'Bytes' marca la información contenida dentro del campo de datos de la trama.

Tabla 3-2. Mensajes que la carga útil enviará al OBC.

ID	Descripción	Bytes							
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0x100	Heart beat / House Keeping	seq	Corriente 0		Corriente 1		Corriente 2		
0x101	Medida Sensor Radiación	Marca de tiempo				seq	Valor		
0x102	Medida del Fotodiodo	Marca de tiempo				seq	ID_{SENSOR} + Valor		
0x103	Medida memoria SRAM	Marca de tiempo				seq	Valor		
0x104	ACK	Nº							
0x10B	Respuesta a NACK	Marca de tiempo				seq			

A continuación, se describe el propósito de los distintos mensajes establecidos:

- **Heart beat (0x100)**: Es un mensaje para indicar al OBC que la carga útil está funcionando correctamente. Este mensaje se envía periódicamente cada 30 segundos. En caso de que el OBC deje de recibir un número determinado de Heart Beats, actuará en consecuencia forzando el reinicio de la carga útil.

  La trama contiene un byte para indicar el número de secuencia y, por otra parte, se usa a modo de mensaje tipo **'House Keeping'**, ya que contiene 3 medidas de corriente que servirán para que el OBC pueda estimar el consumo energético de la carga útil

- **Medidas de sensores (0x101, 0x102, 0x103)**: Son las tramas que se enviarán con datos de mediciones. Incluyen la marca de tiempo en formato UNIX 70, el número de secuencia y el valor medido.

- **ACK (0x104)**: Mensaje de afirmación para indicar al OBC que se ha recibido un mensaje. Contiene el número de mensaje recibido.

- **Respuesta a NACK (0x10B)**: Mensaje que se enviará en caso de que el OBC comunique que no ha recibido un mensaje. Indica el número de secuencia del siguiente mensaje que el OBC va a recibir.

3.2.1.2. Mensajes enviados por el ordenador de a bordo hacia la carga útil

La Tabla 3-3 recoge los mensajes que se han definido para establecer la comunicación desde el OBC a la carga útil.

Tabla 3-3. Mensajes que el OBC enviará a la carga útil.

ID	Descripción	Bytes							
		0	1	2	3	4	5	6	7
0x105	Actualizar Reloj	Marca de tiempo							
0x106	Modo de operación	Modo							
0x108	Petición bajo demanda	Marca de tiempo							
0x109	ACK	ID							
0x10A	NACK	ID							

A continuación, se describe el propósito de los distintos mensajes que podrá enviar el OBC:

- **Actualizar reloj (0x105)**: Periódicamente, el OBC enviará un mensaje con este identificador para que la carga útil actualice su reloj de tiempo real y así poder incluir una marca de tiempo correcta en los distintos datos que se envíen al OBC.
- **Modo de operación (0x106)**: Mensaje en el que se indicará a la carga útil cambiar de modo de funcionamiento para consumir menor energía en caso de que la carga de las baterías sea insuficiente.
- **Petición bajo demanda (0x108)**: El OBC puede solicitar a la carga útil datos de la corriente consumida, la realización de una medida de la memoria SRAM o de los sensores de radiación total.
- **ACK (0x109)**: Mensaje de asentimiento en el que si indica el número de mensaje que se asiente.
- **NACK (0x10A)**: Trama para indicar que no se ha recibido un mensaje. Se indicará el número de secuencia que no se ha recibido.

3.2.2. Lógica de retransmisión de mensajes

En caso de que el ordenador de a bordo se encuentre inoperativo, los mensajes que genere la carga útil no serán recibidos. Por tanto, es necesario establecer una lógica de reenvío de mensajes para evitar así que se pierdan durante la transmisión.

Esta lógica se basa en un funcionamiento mediante **asentimientos negativos (NACKs)**, esto quiere decir que el OBC únicamente enviará un mensaje de respuesta a una trama de datos en el que caso de que se detecte la pérdida de un mensaje anterior. Gracias al uso de números de secuencia se hace el seguimiento de los mensajes enviados y se puede detectar la pérdida de mensajes.

En el caso de que la carga útil reciba un NACK, esta reenviará al OBC todos los mensajes que se han enviado, comenzando por el mensaje con el número de secuencia indicado en el NACK. Si el número de secuencia es antiguo y la carga útil ya no almacena el mensaje indicado, se eliminarán los mensajes pendientes de retransmisión y se volverán a sincronizar los números de secuencia entre carga útil y OBC. En la sección 4.2.3.2 se muestra el proceso de retransmisión de mensajes.

3.2.3. Diagramas de paso de mensajes

Para finalizar con el envío de mensajes entre OBC y carga útil, se muestran los distintos casos de paso de mensajes que pueden darse durante el transcurso de la misión.

3.2.3.1. Paso de mensajes con retransmisión exitosa

La carga útil enviará cada 30 segundos una trama de 'Heart Beat' (0x100) con un número de secuencia y medidas de las corrientes consumidas en ese momento. Tras ello, la carga útil espera como respuesta por parte del OBC una trama de tipo ACK (0x109). Este paso de mensaje se puede observar al comienzo de la comunicación ejemplificada en la Figura 3-4. En caso de que uno de los mensajes enviados por la carga útil no se reciba en el OBC y este envía una trama NACK (0x10A), se transmiten todos los mensajes que tiene en la cola de retransmisión desde el identificador recibido, en adelante. Se entiende, por tanto, que los mensajes anteriores sí han sido recibidos correctamente. Este hecho queda reflejado en la segunda parte de la comunicación representada en la Figura 3-4.

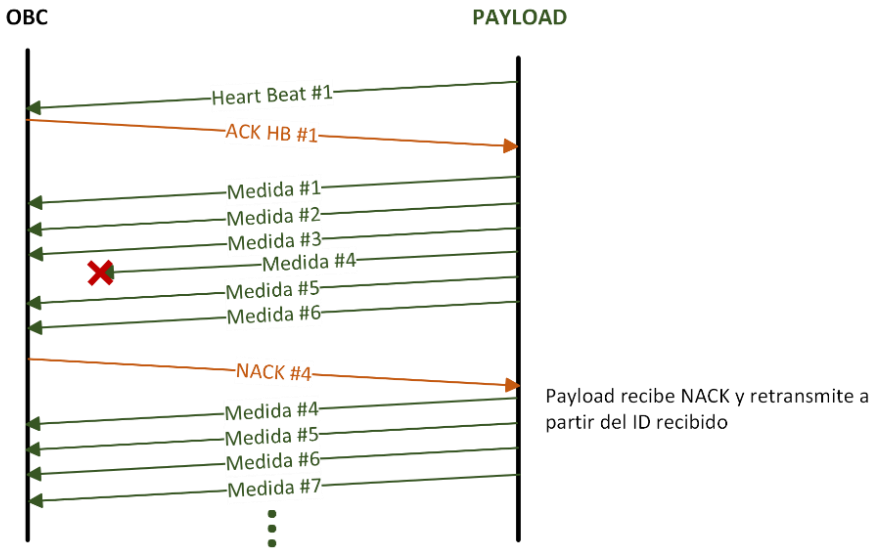

Figura 3-4. Paso de mensajes con retransmisión exitosa

3.2.3.2. Paso de mensajes con retransmisión fallida

Si el OBC envía un NACK (0x10A) con un número de secuencia antiguo, el cual ya no se encuentra en la cola de retransmisión, se vacía la cola y se envía un mensaje de respuesta al NACK (0x10B) indicando el nuevo número de secuencia a partir del cual continuar con el control de errores. Este proceso se representa en la Figura 3-5.

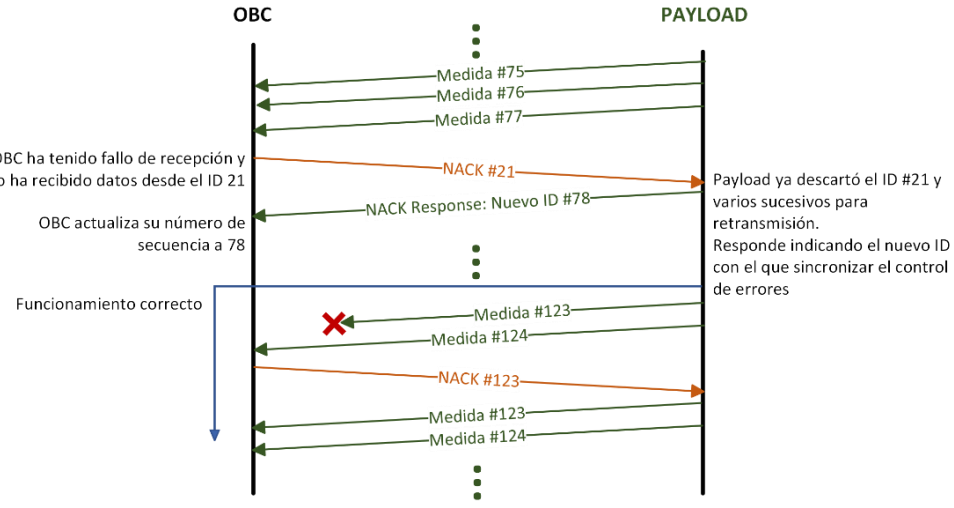

Figura 3-5. Paso de mensajes con retransmisión errónea.

3.2.3.3. Peticiones por parte del ordenador de a bordo

Si la carga útil recibe un mensaje del tipo 'Heart Beat Request' (0x108), esta responde con una trama de tipo 'Heart Beat' donde se indique el valor de la corriente consumida por cada raíl de alimentación. El paso de mensajes que se produce en este escenario se representa en la Figura 3-6.

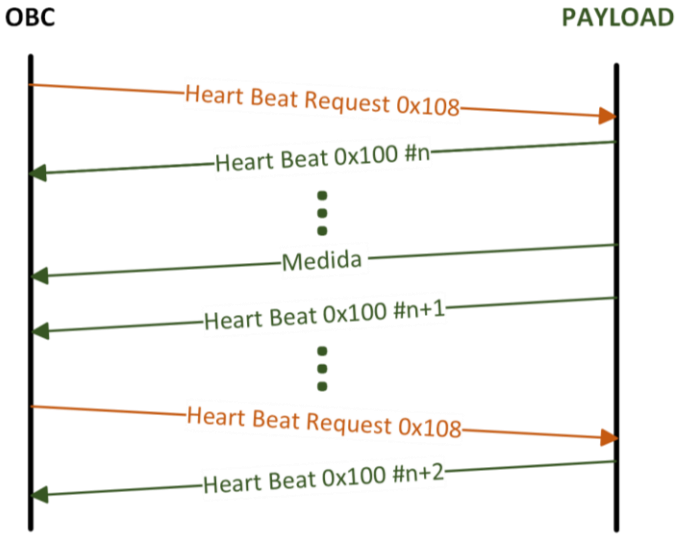

Figura 3-6. Petición de Heart Beat por parte del OBC.

3.2.3.4. Actualización de marca de tiempo y acceso a modo de bajo consumo por acumulación de 'Heart Beats' sin asentir

El OBC envía un mensaje cada cierto intervalo de tiempo de tipo Timestamp (0x105) para que la carga útil actualice su reloj de tiempo real. En el supuesto caso de que la carga de pago haya enviado 10 tramas de tipo 'Heart Beat' (0x100) y ninguna haya sido asentida, esta pasará a modo de bajo consumo ya que se entiende que se ha producido un fallo en el OBC. Ante esta situación, la carga útil deberá ser nuevamente activada por el OBC, si logra recuperarse su operación normal. La Figura 3-7 muestra el paso de mensajes que se genera en este escenario.

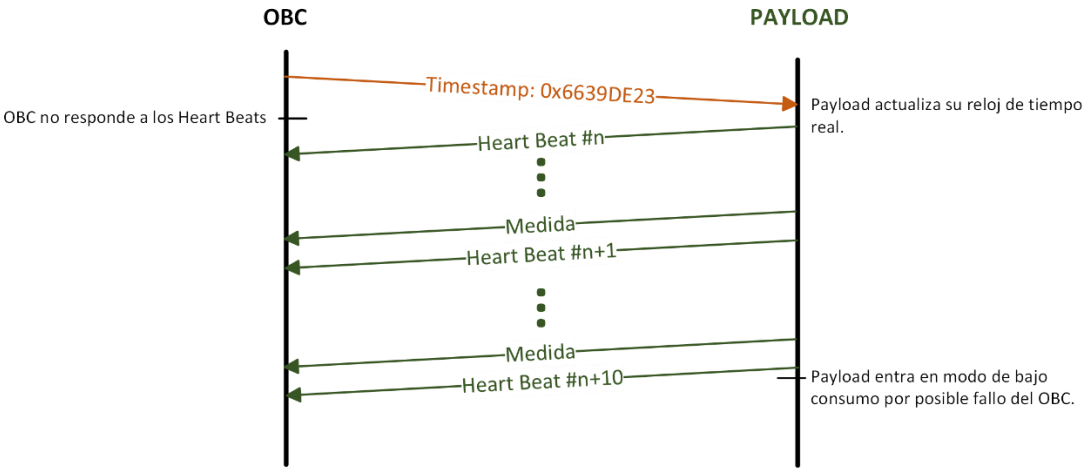

Figura 3-7. Actualización de RTC y acceso a modo de bajo consumo.

4. DESCRIPCIÓN DEL SISTEMA

En este capítulo se detallará cómo se ha desarrollado la arquitectura de software de la carga útil, la cual se basa en un sistema operativo de tiempo real para hacer un uso más eficiente de la capacidad de procesamiento del microcontrolador. Se mostrará también el diseño del controlador del bus CAN para el SAM3X8E y se indicará cómo se ha realizado la gestión de los modos de bajo consumo con los que cuenta el microcontrolador.

4.1. Sistema Operativo de Tiempo Real

Un sistema operativo de tiempo real o RTOS por sus siglas en inglés, es un sistema operativo para aplicaciones en las que las restricciones de tiempo son muy limitadas y es crítico atender eventos conforme ocurren. A diferencia de otro tipo de sistemas operativos de propósito general, los sistemas en tiempo real están basados en eventos que afectarán al comportamiento del sistema. Por ello, la principal característica de un sistema operativo de tiempo real radica en la asignación de prioridades a las distintas tareas que el sistema podrá ejecutar, garantizando así que los eventos de mayor prioridad son atendidos de manera inmediata.

En el caso de aplicaciones con CubeSats, el uso de sistemas operativos de tiempo real es fundamental puesto que permite al sistema mantener una comunicación eficaz con los sensores y el resto de los subsistemas, respondiendo ante eventos que requieren atención inmediata [15].

Se empleará FreeRTOS, el sistema operativo de tiempo real más usado en el mercado a nivel global debido a ser distribuido libremente bajo licencia de código abierto y contar con soporte para un gran número de microcontroladores de una gran diversidad de fabricantes. FreeRTOS permite desarrollar sistemas fiables y robustos de manera sencilla. Cuenta con una extensa documentación y una amplia comunidad de desarrolladores, lo cual facilita el proceso de desarrollo.

A continuación, se explicarán los conceptos de mayor relevancia de FreeRTOS que se han usado en el desarrollo de este proyecto:

4.1.1. Tareas

Las tareas son uno de los conceptos fundamentales de un sistema operativo de tiempo real. El uso de tareas es una herramienta muy potente que permite conseguir un pseudo-paralelismo con microcontroladores que cuentan con un único núcleo de procesamiento.

El desarrollador debe definir todas las tareas que el sistema requiera para funcionar, asignando a cada una su nivel de prioridad. Cada tarea tendrá su propia funcionalidad definida en el código como una función de lenguaje C que cuenta con un bucle infinito.

Durante la ejecución, un planificador o 'scheduler' es quién decidirá la tarea que se ejecuta en cada 'tick' del sistema operativo, en función de la prioridad y de si se ha detectado el evento que activa alguna tarea. Si durante la ejecución de una tarea, se detecta un evento de mayor prioridad, el planificador pausa la ejecución de la tarea que se encontraba atendiendo y comenzará con la ejecución de la tarea de mayor prioridad. Una vez termine de atenderla, volverá al estado en el que se encontraba cuando pausó la ejecución de la tarea anterior. En la Figura 4-1 se puede apreciar el funcionamiento.

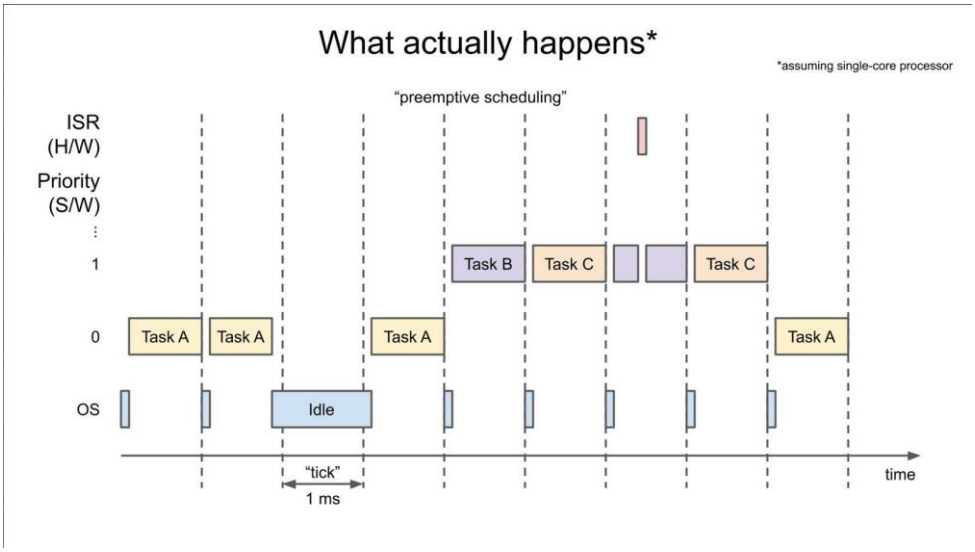

Figura 4-1. Gestión de prioridades y tareas en FreeRTOS [16].

A su vez, es importante conocer los 4 estados en los que se puede encontrar una tarea mientras el sistema se encuentra en funcionamiento. Estos estados los asigna el planificador automáticamente durante la ejecución:

- **'Ready' o preparada:** La tarea está lista para ejecutarse y se ejecutará en un 'tick' próximo tras la ejecución de otras tareas con el mismo nivel de prioridad.
- **'Running' o funcionando:** La tarea está en ejecución.
- **'Blocked' o bloqueada:** La tarea queda a la espera de que ocurra un evento que la active para comenzar a ejecutarse. Mientras se encuentra en este estado no consume recursos del microcontrolador.
- **'Suspended' o suspendida:** La tarea queda inactiva, sin consumir recursos y tampoco se activará en caso de que ocurra un evento. Sólo podrá activarse mediante una llamada desde otra tarea para que vuelva a activarse.

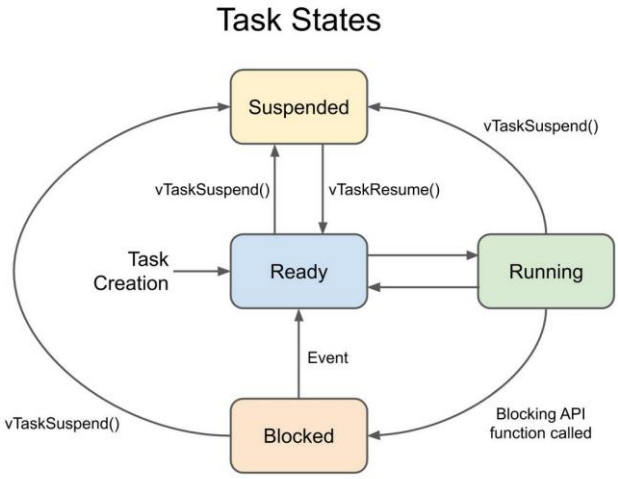

Figura 4-2. Estados de una tarea en FreeRTOS [16].

4.1.2. Colas

Una cola, es un objeto de FreeRTOS que permite intercambiar datos entre distintas tareas del sistema. Su principal ventaja es la protección frente a condiciones de carrera entre tareas que accedan a un mismo dato.

Una cola se puede entender como un sistema FIFO que permite lecturas y escrituras atómicas, asegurando que

otra tarea no pueda sobrescribir un dato antes de que sea leído por la tarea objetivo.

El uso de colas es fundamental para garantizar la paralelización de tareas, puesto que se pueden emplear para mantener tareas en estado de bloqueo. Por ejemplo, si una tarea se encarga de procesar datos de una cola en la que varias tareas copian distintos valores, en caso de que no haya datos para almacenar en la cola, la tarea quedará bloqueada a la espera de que otras tareas almacenen datos en ella. Cuando esto ocurre, la tarea pasa a estado 'Ready' para ejecutarse cuando el planificador lo permita.

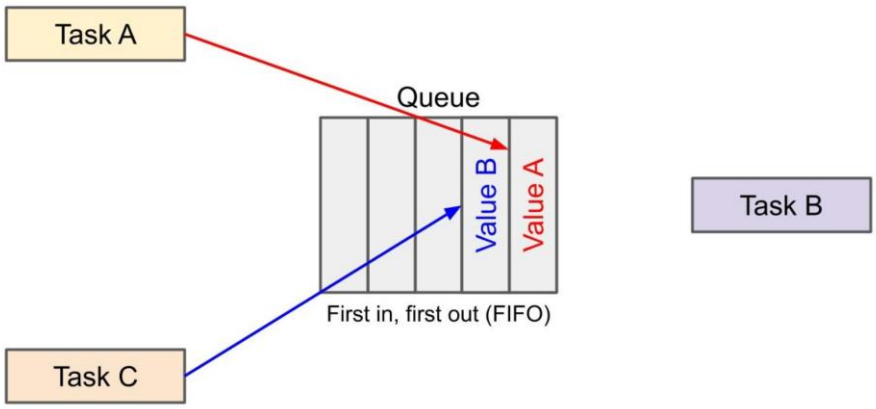

Figura 4-3. Funcionamiento de una cola en FreeRTOS. Las tareas A y C copian datos en la cola, que serán leídos por la tarea B [17].

El desarrollador indicará el tipo de datos que una cola almacenará, desde valores como enteros o números flotantes, hasta estructuras de datos. También será necesario indicar el número de datos que cada cola podrá almacenar, ya que supone una reserva de memoria en el momento de creación de la cola.

4.1.3. Notificaciones

Las notificaciones se incorporaron en la versión 8.2 de FreeRTOS. Son un mecanismo ligero y eficiente para la comunicación entre tareas. El uso de notificaciones puede suponer un funcionamiento un 45% más rápido y consume menor espacio en memoria RAM respecto al uso de otros recursos como colas o semáforos [18].

Al igual que las colas, las notificaciones contribuyen a la paralelización ya que tienen la capacidad de bloquear tareas para que esperen a recibir notificaciones por parte de otras tareas o de rutinas de interrupción. De este modo, las notificaciones son de gran utilidad para atender de manera eficiente eventos externos como la activación de GPIOs

Cada tarea tiene asociado un espacio en memoria para almacenar las múltiples notificaciones que puede recibir mientras se encuentra en estado bloqueado. Cada notificación puede encontrarse pendiente o no pendiente para indicar al planificador si ya ha sido atendida por la tarea. Además, cada una de las notificaciones cuenta con un *valor de notificación* de 32 bits que puede usarse para transmitir datos ligeros, ahorrando el uso de colas.

4.2. Arquitectura completa del sistema

Una vez han sido explicados los conceptos de FreeRTOS esenciales para entender el desarrollo del sistema, se procede a explicar la arquitectura completa para la gestión de mensajes y obtención de datos de los distintos sensores con los que contará la carga útil que generarán datos para ser enviados.

La arquitectura puede dividirse en 3 bloques funcionales dedicados a la gestión de los mensajes recibidos, obtención de los datos generados por los sensores y envío de mensajes al OBC. En la Figura 4-4 se puede apreciar el diseño descrito, formado por las distintas tareas y colas que constituyen la arquitectura completa.

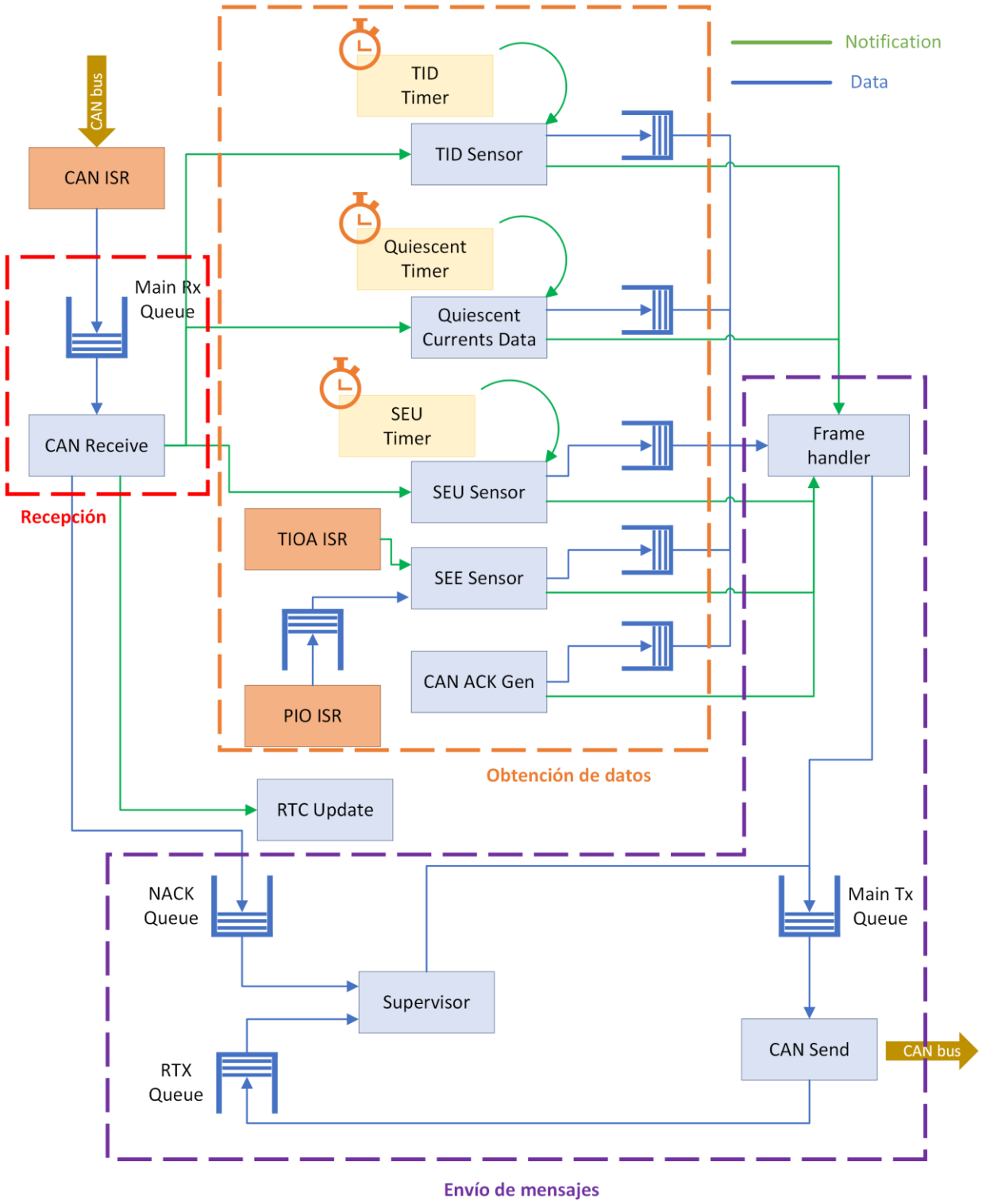

Figura 4-4. Arquitectura completa del sistema en FreeRTOS.

4.2.1. Gestión de mensajes recibidos

La parte encargada de la gestión de mensajes recibidos por el ordenador de a bordo está formada por la tarea denominada 'CAN_Receive' y la cola principal de recepción 'CAN Main Rx Queue'.

4.2.1.1. CAN Receive Task

Corresponde con la tarea encargada de obtener los datos de la cola de recepción 'CAN_Rx_Main_Queue' para

posteriormente notificar a la tarea que corresponda según el ID CAN recibido.

La tarea queda bloqueada a la espera de que la rutina de interrupción encargada de recibir los mensajes CAN almacene en la cola principal de transmisión la estructura con los datos del mensaje recibido.

Una vez ha recibido la estructura, dependiendo del ID se actúa de la siguiente forma:

- **Recibe OBC_NACK_RECV:** Envía la trama a la cola de NACKs, 'CAN_NACK_Queue'
- **Recibe OBC_TIMESTAMP_RECV:** Notifica a la tarea encargada de actualizar el RTC.
- **Recibe OBC_LPM_RECV:** La carga útil pasa a modo de bajo consumo.
- **Recibe OBC_ACK_RECV:** Se notifica a la tarea 'Quiescent Currents Data' que se ha recibido asentimiento de una trama de House Keeping/Heart Beat (0x100).
- **Recibe OBK_HK_RESP**: Notifica a la tarea de generación de ACKs y dependiendo del tipo de medida que el OBC haya pedido, se notifica a las distintas tareas de generación de datos.

4.2.1.1.1. Variables locales

Tabla 4-1. Variables locales empleadas en la función de recepción de mensajes CAN.

Nombre	Tipo	Descripción
Received_Frame	CAN_Frame_t	Estructura con los datos CAN recibidos.
Received_CAN_ID	MSG_ID_OBC_TO_PYL_t	ID recibido.
notify_value	QS_NOTIFY_t	Valor de notificación a enviar.

4.2.1.1.2. Colas empleadas por la tarea

Tabla 4-2. Colas empleadas por la función de recepción de mensajes CAN.

Nombre	Tipo	Descripción
CAN_Main_Rx_Queue	CAN_Frame_t	Cola principal de recepción de tramas.
CAN_NACK_Queue	CAN_Frame_t	Cola de NACKs recibidos.

4.2.1.1.3. Notificaciones

Tabla 4-3. Notificaciones usadas en la función de recepción de mensajes CAN.

Envía/Recibe	Destino/Origen	Valor
Envía	RTC_Update_Task	Marca de tiempo recibida.
Envía	QUIESCENTS_CURRENTS_Get_Data_Task	HB_ACK_RECV: Tipo QS_NOTIFY_t almacenado en notified_value.
Envía	CAN_ACK_Generate_Task	ACK recibido.
Envía	RADFET_Generate_Data_Task	Notificación binaria.
Envía	PHOTODIODE_X_Generate_Data_Task	Notificación binaria.
Envía	RAM_Generate_Data_Task	Notificación binaria.
Envía	QUIESCENTS_CURRENTS_Get_Data_Task	GENERATE_DATA:Tipo QS_NOTIFY_t almacenado en notified_value.

4.2.2. Obtención de datos de los sensores

Este bloque se encarga de la obtención y el procesado de los datos obtenidos por los sensores. Cada sensor tiene asociada su propia tarea y temporizador, encargadas de recopilar los datos y de activar las tareas, respectivamente. En el caso de los fotodiodos de detección de impactos, sus correspondientes tareas serán notificadas desde la rutina de interrupción de los GPIO asociados a cada sensor, que a su vez se activarán con el impacto de una partícula con la energía suficiente para generar el pulso que habilite la interrupción.

Todas las tareas almacenarán junto con el dato obtenido, la marca de tiempo del instante en que se ha tomado el dato para ser enviada al OBC.

4.2.2.1. RADFET Get Data Task

Genera datos del RADFET. La tarea espera una notificación por parte de un temporizador el cual, al expirar, notifica a la tarea para generar los datos.

Una vez se ha recibido la notificación, se generan todos los datos necesarios para formar un mensaje. Se obtiene la marca de tiempo, se asigna el ID del CAN asociado al RADFET, se obtienen los datos de la medición (en este caso datos) y se asigna un número de secuencia a la medida.

Tras almacenar la información en la estructura, esta se envía a la cola 'RADFET_Data_Queue', se notifica a la tarea 'CAN_Framing' para preparar la información para ser enviada por el bus CAN y se incrementa el número de secuencia.

4.2.2.1.1. Variables locales

Tabla 4-4. Variables locales empleadas en la función de obtención de datos del RADFET.

Nombre	Tipo	Descripción
pyl_to_obc_can_id	MDG_ID_PYL_TO_OBC_t	ID CAN asociado.
notify_value	ISR_NOTIFY_t	Valor de notificación.
timestamp	uint32_t	Marca de tiempo.
Data_Frame	SENSOR_Data_t	Estructura para almacenar datos.

4.2.2.1.2. Colas empleadas por la tarea

Tabla 4-5.Colas empleadas por la función de obtención de datos del RADFET.

Nombre	Tipo	Descripción
RADFET_Data_Queue	SENSOR_Data_t	Datos generados por el RADFET.

4.2.2.1.3. Notificaciones

Tabla 4-6. Notificaciones usadas en la función de obtención de datos del RADFET.

Envía/Recibe	Destino/Origen	Valor
Recibe	RADFET_Get_Data_Callback	Notificación binaria.
Envía	CAN_Framing_Task	RADFET_NOTIFY:Tipo ISR_NOTIFY_t almacenado en notify_value.

4.2.2.2. PHOTODIODE X Get Data Task

Tarea para obtener datos generados por los fotodiodos de detección de impactos. Hay un total de 4 tareas (X = 1,2,3,4), asociadas a cada uno de los fotodiodos.

Las tareas esperan a ser notificadas para obtener los datos. Una vez han recibido la notificación, almacenan en una estructura la marca de tiempo, el ID CAN asociado a los fotodiodos, la medida de los datos (en este caso se generan a partir del Tick Count) y el número de secuencia de la medida.

Finalmente, se envía la estructura a la cola 'PHOTODIODE_Data_Queue', se notifica a la tarea 'CAN_Framing' y se incrementa el número de secuencia.

4.2.2.2.1. Variables locales

Tabla 4-7. Variables locales empleadas en las funciones de obtención de datos de los fotodiodos.

Nombre	Tipo	Descripción
pyl_to_obc_can_id	MDG_ID_PYL_TO_OBC_t	ID CAN asociado.
photo_id	PHOTODIODE_ID_t	Identificador del fotodiodo.
notify_value	ISR_NOTIFY_t	Valor de notificación.
timestamp	uint32_t	Marca de tiempo.
Data_Frame	SENSOR_Data_t	Estructura para almacenar datos.

4.2.2.2.2. Colas empleadas por la tarea

Tabla 4-8. Colas empleadas por la función de obtención de obtención de datos de los fotodiodos.

Nombre	Tipo	Descripción
PHOTODIODE_Data_Queue	SENSOR_Data_t	Datos generados por fotodiodos.

4.2.2.2.3. Notificaciones

Tabla 4-9. Notificaciones usadas en la función de obtención de datos de los fotodiodos.

Envía/Recibe	Destino/Origen	Valor
Recibe	ISR: photodiode_handler	Notificación binaria.
Envía	CAN_Framing_Task	PHOTODIODES_NOTIFY: Tipo ISR_NOTIFY_t almacenado en notify_value.

4.2.2.3. PHOTODIODE Get PW Task

Tarea para almacenar los datos de medición del ancho de pulso detectado en los fotodiodos en la cola 'PHOTODIODE_Data_Queue'.

La tarea espera en estado bloqueado hasta que recibe una notificación por parte de la rutina de interrupción (ISR) que calcula el ancho del pulso. El valor de la notificación contiene el ancho del pulso y el número de impactos recibido durante el cálculo.

Una vez recibida la notificación, lee de la cola 'PHOTODIODE_ID_Queue' el identificador del fotodiodo que ha sido impactado y almacena en la estructura de datos la marca de tiempo, el ID CAN, el ID del fotodiodo, el

ancho del pulso y el número de secuencia.

Finalmente, envía a la cola 'PHOTODIODE_Data_Queue' la estructura de datos, notifica a la tarea 'CAN_Framing_Task' e incrementa el número de secuencia.

4.2.2.3.1. Variables locales

Tabla 4-10.Variables locales empleadas en las funciones de cálculo de ancho de pulso de los impactos detectados.

Nombre	Tipo	Descripción
pyl_to_obc_can_id	MDG_ID_PYL_TO_OBC_t	ID CAN asociado.
photo_id	uint8_t	ID del fotodiodo impactado
notify_value	ISR_NOTIFY_t	Valor de notificación a enviar.
timestamp	uint32_t	Marca de tiempo.
Data_Frame	SENSOR_Data_t	Estructura para almacenar datos.
PW	uint32_t	Ancho del pulso.

4.2.2.3.2. Colas empleadas por la tarea

Tabla 4-11. Colas empleadas por la función de cálculo de ancho de pulso de los impactos detectados.

Nombre	Tipo	Descripción
PHOTODIODE_ID_Queue	uint8_t	IDs de los fotodiodos impactados.
PHOTODIODE_Data_Queue	SENSOR_Data_t	Datos generados por fotodiodos.

4.2.2.3.3. Notificaciones

Tabla 4-12. Notificaciones usadas en la función de cálculo de ancho de pulso de los impactos detectados.

Envía/Recibe	Destino/Origen	Valor
Recibe	ISR: TC0_handler	Ancho del pulso en número de system ticks
Envía	CAN_Framing_Task	PHOTODIODES_NOTIFY: Tipo ISR_NOTIFY_t almacenado en notify_value.

4.2.2.4. RAM Get Data Task

Tarea para obtener datos de la memoria RAM usada para detectar eventos del tipo 'Single Event Upset'.

La tarea espera a recibir una notificación. Una vez recibida, almacena la marca de tiempo, el ID CAN asociado a las medidas de la RAM, los datos (en este caso a partir del Tick Count) y el número de secuencia en una estructura.

Una vez completa la estructura, se envía a la cola 'RAM_Data_Queue', se notifica a la tarea 'CAN_Framing_Task' y se incrementa el número de secuencia.

4.2.2.4.1. Variables locales

Tabla 4-13. Variables locales empleadas en la función de obtención de datos de la memoria RAM.

Nombre	Tipo	Descripción
pyl_to_obc_can_id	MDG_ID_PYL_TO_OBC_t	ID CAN asociado.
notify_value	ISR_NOTIFY_t	Valor de notificación.
timestamp	uint32_t	Marca de tiempo.
Data_Frame	SENSOR_Data_t	Estructura para almacenar datos.

4.2.2.4.2. Colas empleadas por la tarea

Tabla 4-14. Colas empleadas por la función de obtención de datos de la memoria RAM.

Nombre	Tipo	Descripción
RAM_Data_Queue	SENSOR_Data_t	Datos generados por RAM.

4.2.2.4.3. Notificaciones

Tabla 4-15. Notificaciones usadas en la función de obtención de datos de la memoria RAM.

Envía/Recibe	Destino/Origen	Valor
Recibe	ISR: photodiode_handler	Notificación binaria.
Envía	CAN_Framing_Task	RAM_NOTIFY: Tipo ISR_NOTIFY_t almacenado en notify_value.

4.2.2.5. QUIESCENT CURRENTS Get Data Task

Tarea para obtener los datos de las medidas de corrientes, que se envía en los mensajes de tipo 'Heartbeat'.

La tarea espera a ser notificada y a continuación, comprueba si ha recibido un ACK de heartbeats anteriores, en caso de haberlo recibido, reinicia el contador de asentimientos no recibidos a los heartbeats.

Si se ha recibido una notificación del tipo 'GENERATE_DATA', almacena en una estructura el ID CAN asociado al heartbeat, los datos de 3 medidas de corriente y un número de secuencia independiente para los heartbeats.

Finalmente, se envía la estructura a la cola 'QUIESCENTS_CURRENTS_Data_Queue', notifica a la tarea 'CAN_Framing_Task' y se incrementa el número de secuencia de las medidas de corriente.

Si el número de heartbeats enviados sin haber recibido asentimiento es superior a 'MAX_NUMBER_HEARTBEAT_WO_ACKS', el sistema pasará a modo de bajo consumo hasta que el OBC vuelva a comunicarse con la carga de pago.

4.2.2.5.1. Variables locales

Tabla 4-16. Variables locales empleadas en la función de generación de medidas de corriente y House Keeping.

Nombre	Tipo	Descripción
pyl_to_obc_can_id	MDG_ID_PYL_TO_OBC_t	ID CAN asociado.
notify_value	ISR_NOTIFY_t	Valor de notificación.
Data_Frame	QUIESCENTS_CURRENTS_Data_t	Estructura para almacenar datos de corrientes.
notified_value	uint32_t	Valor de notificación recibida.

4.2.2.5.2. Colas empleadas por la tarea

Tabla 4-17. Colas empleadas por la función de generación de medidas de corriente y House Keeping.

Nombre	Tipo	Descripción
QUIESCENT_CURRENTS_Data_Queue	QUIESCENT_CURRENTS_Data_t	Medidas de Corrientes.

4.2.2.5.3. Notificaciones

Tabla 4-18. Notificaciones usadas en la función de generación de medidas de corriente y House Keeping.

Envía/Recibe	Destino/Origen	Valor
Recibe	QUIESCENT_CURRENTS_Get_Data_Callback	GENERATE_DATA: Tipo QS_NOTIFY_t almacenado en notified_value.
Recibe	CAN_Receive_Task	HB_ACK_RECV: Tipo QS_NOTIFY_t almacenado en notified_value.
Envía	CAN_Framing_Task	QUIESCENT_CURRENTS_NOTIFY: Tipo ISR_NOTIFY_t almacenado en notify_value.

4.2.2.6. CAN ACK Generate Task

Tarea encargada de generar mensajes de asentimiento a los mensajes recibidos por el OBC.

La tarea permanece bloqueada esperando una notificación para generar un ACK. La propia notificación contiene en el campo de datos el ID para el que generar el ACK.

Al recibir la notificación, se almacena en la estructura de datos el ID CAN y el ID del ACK que tiene que responder.

Finalmente, la estructura se almacena en la cola ‘CAN_PYL_ACK_Queue’ y se notifica a la tarea ‘CAN_Framing_Task’.

4.2.2.6.1. Variables locales

Tabla 4-19. Variables locales empleadas en la función de generación de ACKs.

Nombre	Tipo	Descripción
notified_value	uint32_t	Valor de notificación recibido.
notify_value	uint32_t	Valor de notificación a enviar.
Data_Frame	SENSOR_Data_t	Estructura de datos para enviar.

4.2.2.6.2. Colas empleadas por la tarea

Tabla 4-20. Colas empleadas por la función de generación de ACKs.

Nombre	Tipo	Descripción
CAN_PYL_ACK_Queue	SENSOR_Data_t	Cola con tramas de ACK pendientes de envío.

4.2.2.6.3. Notificaciones

Tabla 4-21. Notificaciones usadas en la función de generación de ACKs.

Envía/Recibe	Destino/Origen	Valor
Recibe	CAN_Receive_Task	Ancho del pulso medido.
Envía	CAN_Framing_Task	ACK_NOTIFY: Tipo ISR_NOTIFY_t almacenado en notified_value.

4.2.3. Envío de mensajes al OBC

La última parte implementada en el firmware desarrollado es la gestión de los mensajes que se envían al ordenador de a bordo a través del bus CAN. Las funciones de este bloque consisten en unificar las distintas estructuras de datos provenientes de los distintos sensores en una común, que simplifique la tarea de enviar a través del bus CAN. Por otro lado, existirá una tarea de supervisión, encargada de que la retransmisión de mensajes se haga correctamente en caso de pérdida de tramas durante la comunicación.

4.2.3.1. CAN Framing Task

Tarea encargada de preparar todos los tipos de datos que generan los distintos sensores para ser enviados usando un formato común a través del bus CAN.

La tarea espera a ser notificada por cualquiera de las distintas tareas que generan datos de medidas para encapsular la información en una estructura del tipo 'CAN_Frame_t' creada para facilitar el uso de las funciones de transmisión por el bus CAN.

Dependiendo del bit que se haya activado en el valor de notificación, se lee una cola u otra para posteriormente almacenar en los campos de la estructura los datos de la medida.

Para los mensajes que contienen medidas de los sensores, en el campo 'Data_Low' se almacena la marca de tiempo y en 'Data_High' se guarda la medida y el número de secuencia de esta.

En caso de enviar un mensaje del tipo ACK, únicamente se encapsula el ID que le corresponde y en los 8 bits menos significativos el número de asentimiento.

Para el caso de las medidas de corriente, no se almacena marca de tiempo por lo que se emplean los 8 bytes de

la trama del CAN para guardar las 3 medidas de corriente.

Finalmente, en cualquiera de los casos, la estructura se envía a la cola 'CAN_Main_Tx_Queue'.

4.2.3.1.1. Variables locales

Tabla 4-22. Variables locales empleadas en la función de gestión de las tramas CAN.

Nombre	Tipo	Descripción
notified_value	uint32_t	Valor de notificación recibida.
Data_to_Frame	SENSOR_Data_t	Estructura de datos de medida de sensores.
Meas_to_Frame	QUIESCENTS_CURRENTS_Data_t	Estructura con datos de corrientes.
Frame_to_Send	CAN_Frame_t	Estructura con datos a enviar por el bus CAN.
sequence_id	uint8_t	Entero para almacenar número de secuencia de manera auxiliar.
data	uint32_t	Entero para almacenar datos de manera auxiliar.

4.2.3.1.2. Colas empleadas por la tarea

Tabla 4-23. Colas empleadas por la función de gestión de las tramas CAN.

Nombre	Tipo	Descripción
CAN_PYL_ACK_Queue	SENSOR_Data_t	ACKs pendientes.
RADFET_Data_Queue	SENSOR_Data_t	Medidas del RADFET.
PHOTODIODE_Data_Queue	SENSOR_Data_t	Medidas de fotodiodos.
RAM_Data_Queue	SENSOR_Data_t	Medidas de RAM.
QUIESCENT_CURRENTS_Data_Queue	QUIESCENT_CURRENTS_Data_t	Medidas de Corrientes.
CAN_Main_Tx_Queue	CAN_Frame_t	Cola principal de transmisión de tramas.

4.2.3.1.3. Notificaciones

Tabla 4-24. Notificaciones usadas en la función de gestión de las tramas CAN.

Envía/Recibe	Destino/Origen	Valor
Recibe	CAN_ACK_Generate_Task	ACK_NOTIFY: Tipo ISR_NOTIFY_t almacenado en notified_value.
Recibe	RADFET_Generate_Data_Task	RADFET_NOITIFY: Tipo ISR_NOTIFY_t almacenado en notified_value.
Recibe	PHOTODIODE_X_Generate_Data_Task	PHOTODIODES_NOTIFY: Tipo ISR_NOTIFY_t almacenado en notified_value.
Recibe	RAM_Generate_Data_Task	RAM_NOTIFY: Tipo ISR_NOTIFY_t almacenado en notified_value.
Recibe	QUIESCENTS_CURRENTS_Get_Data_Task	QUIECENT_CURRENTS_NOTIFY: Tipo ISR_NOTIFY_t almacenado en notified_value.

4.2.3.2. CAN Msg Supervisor Task

Tarea encargada de comprobar que la comunicación con el OBC se ha producido correctamente y en caso de detectar un error, retransmitir aquellas tramas que sean necesarias.

La tarea permanece bloqueada mientras no haya datos en la cola 'CAN_NACK_Queue'.

Una vez recibe un NACK se obtiene el número de secuencia del mensaje a retransmitir. Si hay datos disponibles en la cola de retransmisión 'CAN_RTX_Queue', comienza la comparación de los números de secuencia de las tramas almacenadas con el NACK recibido.

En caso de que el número de NACK recibido sea menor que el número de secuencia a retransmitir, la trama se descarta puesto que se entiende que esa trama sí se ha recibido.

Cuando el número de secuencia a retransmitir sea mayor o igual al número de NACK, se reenvían todas las tramas restantes en la cola de retransmisión.

En caso de que el número de NACK recibido no se encuentre en la cola de retransmisión, se reinicia la cola y se envía al OBC un mensaje indicando un nuevo número de secuencia para sincronizarse con la carga de pago.

En la Figura 4-5 se muestra el comportamiento descrito.

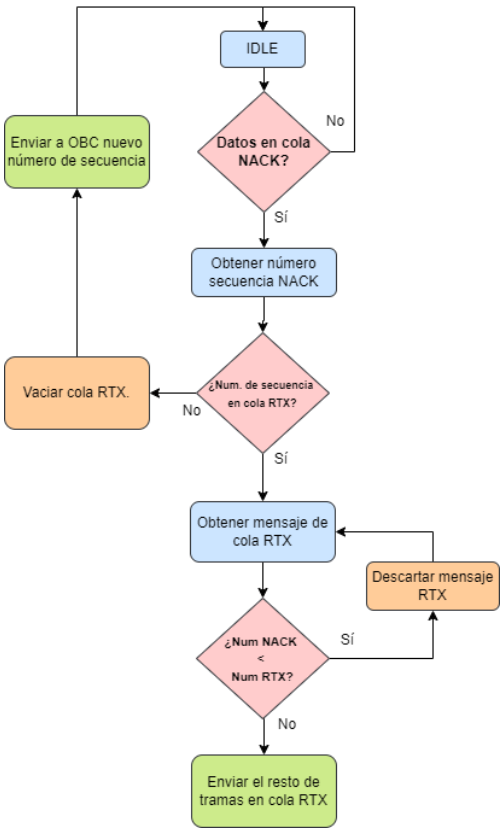

Figura 4-5. Diagrama de flujo de la gestión de NACKs por parte de la tarea de supervisión de mensajes.

4.2.3.2.1. Variables locales

Tabla 4-25. Variables locales empleadas en la función de supervisión de mensajes.

Nombre	Tipo	Descripción
Sent_Frame	CAN_Frame_t	Estructura con datos enviados por bus CAN.
Sent_Sequence_Id	uint8_t	Valor de secuencia enviado.
ReceivedACK_Frame	CAN_Frame_t	Estructura con datos recibidos por bus CAN.
ReceivedACK_Sequence_Id	uint8_t	Valor de secuencia recibido.
Frame_to_Send	CAN_Frame_t	Estructura con datos a enviar por bus CAN.
acks_pending	uint8_t	Número de ACKS pendientes.
rtx_msgs	uint8_t	Número de mensajes en la cola de retransmisión.
tx_msgs	uint8_t	Número de mensajes en la cola de transmisión.
Wrong_NACK_flag	uint8_t	Bandera para indicar que no el número de secuencia recibido no se encuentra en la cola de retransmisión.
Aux_Frame	CAN_Frame_t	Estructura auxiliar para almacenar trama CAN.

4.2.3.2.2. Colas empleadas por la tarea

Tabla 4-26. Colas empleadas por la función de gestión de supervisión de mensajes.

Nombre	Tipo	Descripción
CAN_NACK_Queue	CAN_Frame_t	Cola con tramas de NACK recibidas.
CAN_RTX_Queue	CAN_Frame_t	Cola con mensajes enviados para posible retransmisión.
CAN_Main_Tx_Queue	CAN_Frame_t	Cola principal de transmisión de mensajes del CAN.

4.2.3.2.3. Notificaciones

La tarea no recibe ni envía notificaciones.

4.3. Resumen de la arquitectura

En este apartado se realiza un resumen de las tareas, colas, estructuras y enumeraciones empleadas en la arquitectura completa del sistema.

4.3.1. Tareas

Las tareas descritas en la sección anterior deben ser creadas, asignándoles un manejador, un espacio en memoria, una prioridad y un nombre característico. La Tabla 4-27 recoge los datos de cada una de ellas.

Tabla 4-27. Tareas del sistema completo y asignación de prioridades.

Función	Nombre	Tamaño pila	Prioridad	Manejador
CAN_Framing_Task	CAN Framing	60	1	CAN_Framing_Handler
CAN_Send_Task	Can Send Task	70	2	CAN_Send_Handler
CAN_Receive_Task	CAN Receive Task	50	2	CAN_Receive_Handler
CAN_Msg_Supervisor_Task	Supervisor for messages	80	6	CAN_Msg_Supervisor_Handler
CAN_ACK_Generate_Task	Generate ACKs	50	5	CAN_ACK_Generate_Handler
RTC_Update_Task	RTC Update time	50	4	RTC_Update_Handler
RADFET_Get_Data_Task	RADFET producer	40	5	RADFET_Get_Data_Handler

Función	Nombre	Tamaño pila	Prioridad	Manejador
PHOTODIODE_x_Get_Data_Task	Photodiode x producer	40	5	PHOTODIODE_Get_Data_Handler
RAM_Get_Data_Task	RAM producer	40	5	RAM_Get_Data_Handler
QUIESCENT_CURRENTS_Get_Data_Task	Currents producer	40	5	QUIESCENT_CURRENTS_Get_Data_Handler
PHOTODIODE_Get_PW_Task	Photodiode Pulse Width calculation	80	5	PHOTODIODE_Get_PW_Handler

4.3.2. Colas

Al igual que con las tareas, se deben crear las colas. Hay que asignarles un nombre, el número de elementos que pueden almacenar y el tipo de datos que va a contener.

Nombre	Tamaño	Tipo de dato
RADFET_Data_Queue	15	SENSOR_Data_t
PHOTODIODE_Data_Queue	30	SENSOR_Data_t
PHOTODIODE_ID_Queue	10	uint8_t
RAM_Data_Queue	60	SENSOR_Data_t
QUIESCENT_CURRENTS_Data_Queue	30	QUIESCENT_CURRENTS_Data_t
CAN_Main_Tx_Queue	15	CAN_Frame_t
CAN_Main_Rx_Queue	40	CAN_Frame_t
CAN_RTX_Queue	10	CAN_Frame_t
CAN_NACK_Queue	40	CAN_Frame_t
CAN_PYL_ACK_Queue	10	SENSOR_Data_t

4.3.3. Estructuras de datos

4.3.3.1. CAN_Frame_t

Estructura para almacenar los datos de modo que sea sencillo enviarlos a través del CAN. La tarea 'CAN_Framing_Task' es la encargada de almacenar los datos que recibe en esta estructura para posteriormente ser enviada.

Tabla 4-28. Campos que forman la estructura CAN_Frame_t.

Campo	Tipo	Descripción
u16_CAN_ID	uint16_t	ID CAN asociado a la trama.
ul_Data_High	uint32_t	4 bytes altos del campo de datos del CAN.
ul_Data_Low	uint32_t	4 bytes bajos del campo de datos del CAN.

4.3.3.2. Sensor_Data_t

Estructura en la que las distintas tareas que generan datos almacenan la información. La tarea 'CAN_Framing_Task' se encarga de pasar los datos de esta estructura a una del tipo 'CAN_Frame_t'.

Tabla 4-29. Campos que forman la estructura Sensor_Data_t.

Campo	Tipo	Descripción
u16_CAN_ID	uint16_t	ID CAN asociado a los datos.
ul_TimeStamp	uint32_t	4 bytes altos del campo de datos del CAN.
ul_msgData	uint32_t	4 bytes bajos del campo de datos del CAN.
u_msgSeqId	uint8_t	Número de secuencia de la medida.

4.3.3.3. QUIESCENT_CURRENTS_Data_t

Estructura específica para las medidas de corriente que se envían a modo de "House Keeping".

Tabla 4-30. Campos que forman la estructura QUIESCENT_CURRENTS_Data_t.

Campo	Tipo	Descripción
u16_CAN_ID	uint16_t	ID CAN asociado a la trama.
u_msgSeqId	uint8_t	Número de secuencia de la medida.
u16_current [3]	uint16_t	Array de 3 datos para almacenar las medidas.

4.3.4. Enumeraciones

Las enumeraciones permiten definir valores que ayudan a diferenciar entre tipos de notificaciones, sensores o mensajes. A continuación, se recogen los tipos de definidos.

4.3.4.1. PHOTODIODE_ID_t

Esta enumeración asigna a cada fotodiodo un valor, permitiendo una distinción clara entre ellos.

Tabla 4-31. Valores de la enumeración PHOTODIODE_ID_t.

Nombre	Valor	Descripción
PHOTODIODE_1	0x00	Identifica al fotodiodo 1
PHOTODIODE_2	0x01	Identifica al fotodiodo 2
PHOTODIODE_3	0x02	Identifica al fotodiodo 3
PHOTODIODE_4	0x03	Identifica al fotodiodo 4

4.3.4.2. ISR_NOTIFY_t

En esta enumeración se definen los valores para que la tarea 'CAN_Framing_Task' pueda identificar qué sensor ha generado una medida y leer la cola correspondiente.

Tabla 4-32. Valores de la enumeración ISR_NOTIFY_t.

Nombre	Valor	Descripción
RADFET_NOTIFY	0x01	Notificación del RADFET
PHOTODIODES_NOTIFY	0x02	Notificación de los fotodiodos
RAM_NOTIFY	0x04	Notificación de la RAM
QUIESCENT_CURRENTS_NOTIFY	0x08	Notificación del heartbeat
ACK_NOTIFY	0x10	Notificación del ACK

4.3.4.3. QS_NOTIFY_t

Esta enumeración permite a la tarea 'QUIESCENT_CURRENTS_Get_Data_Task' diferenciar la acción a llevar a cabo según el valor notificado.

Tabla 4-33. Valores de la enumeración QS_NOTIFY_t.

Nombre	Valor	Descripción
GENERATE_DATA	0x01	Genera trama de heartbeat.
HB_ACK_RECV	0x02	Reinicia el contador de heartbeat no asentidos.

4.3.4.4. MSG_ID_PYL_TO_OBC_t

Con esta enumeración se asigna a los mensajes a retransmitir un identificador propio. Se encuentran los siguientes.

Tabla 4-34. Valores de la enumeración MSG_ID_TO_OBC_t.

Nombre	Valor	Descripción
HEARTBEAT_SENT	0X100	El mensaje enviado es un Heartbeat. Demuestra que el sistema sigue operativo.
RADFET_SENT	0X101	El mensaje enviado procede del RADFET.
PHOTODIODE_SENT	0X102	El mensaje enviado procede de los fotodiodos.
RAM_SENT	0X103	El mensaje enviado procede de la RAM.
PYL_ACK_SENT	0X104	El mensaje enviado es un ACK. Indica un mensaje recibido correctamente
NACK_RESP_SENT	0X10B	Respuesta a NACK recibido con ID ya olvidado. Se responde con el

4.3.4.5. MSG_ID_OBC_TO_PYL_t

La siguiente enumeración se emplea para identificar los mensajes recibidos desde el OBC.

Tabla 4-35. Valores de enumeración MSG_ID_TO_OBC_t.

Nombre	Valor	Descripción
OBC_TIMESTAMP_RECV	0X105	El mensaje recibido es una nueva marca de tiempo para actualizar el RTC del sistema.
OBC_LPM__RECV	0X106	El OBC indica al sistema pasar a un modo de bajo consumo
OBC_HK__RECV	0X108	El OBC solicita una medida de un sensor
OBC_ACK__RECV	0X109	El OBC informa que ha recibido el Heart Beat
OBC_NACK_RECV	0X10A	El OBC informa de que no ha recibido un mensaje.

4.3.4.6. HK_ID_t

Esta enumeración está relacionada con los mensajes recibidos del OBC con identificador 'OBC_HK_RECV', ampliando la información que ofrece. En concreto, solicita datos por parte de un sensor.

Tabla 4-36. Valores de la numeración HK_ID_t.

Nombre	Valor	Descripción
OBC_HK_RADFET	0X00	El OBC solicita datos del RADFET.
OBC_HK_PHOTO_1	0X01	El OBC solicita datos del fotodiodo 1.
OBC_HK_PHOTO_2	0X02	El OBC solicita datos del fotodiodo 2.
OBC_HK_PHOTO_3	0X03	El OBC solicita datos del fotodiodo 3.
OBC_HK_PHOTO_4	0X04	El OBC solicita datos del fotodiodo 4.
OBC_HK_RAM	0X05	El OBC solicita datos de la RAM.
OBC_HK_QS	0X06	El OBC solicita datos de las corrientes.

4.4. Controlador del Bus CAN

Aunque ya se ha explicado toda la arquitectura del sistema y el concepto de sistema operativo de tiempo real, el primer objetivo del proyecto consiste en el desarrollo del controlador del bus CAN para el SAM3X8E. A continuación, se muestran las funciones desarrolladas. Cabe indicar que estas funciones hacen uso de las librerías desarrolladas por Atmel.

Es importante conocer el concepto de 'mailbox' o buzón CAN, puesto que se mencionará con frecuencia durante el desarrollo de esta sección. Cada periférico CAN cuenta con 8 'mailboxes' o buzones en el microcontrolador SAM3X8E, cuya función es la de almacenar los mensajes que se reciben. Pueden emplearse en conjunto para aportar funcionalidades añadidas al controlador CAN. No obstante, en este proyecto sólo será necesario hacer uso de dos de estos buzones, uno para recepción y otro para transmisión.

4.4.1. CAN_Initializiation

Función encargada de realizar la configuración inicial del periférico. Los parámetros que recibe son los siguientes:

Tabla 4-37. Parámetros que recibe la función de inicialización del CAN.

Parámetro	Tipo	Descripción
***p_can**	Can	Puntero a una instancia de periférico CAN. (CAN0 o CAN1).
ul_sysclk	uint32_t	Frecuencia de reloj del sistema.
baudrate	uint32_t	Tasa de bits deseada para la transmisión.

Devuelve una variable del tipo uint8_t con valor igual a 1 si la inicialización ha sido correcta. En caso contrario, devuelve 0 si ha habido algún error.

La función, en primer lugar, activa la señal de reloj desde el PMC (Power Management Controller).

Posteriormente, llama a la función 'can_init' incluida en los drivers creados por Atmel, función en la que se configura la velocidad de la comunicación y se preparan los distintos *mailboxes*.

Finalmente, se activan las interrupciones asociadas a la instancia del periférico que se haya proporcionado como parámetro (CAN0 o CAN1) y se reinician todos los *mailboxes*.

4.4.2. reset_mailbox_conf

Esta función reinicia los datos de configuración de un *mailbox*. Recibe como parámetro:

Tabla 4-38.Parámetros que recibe la función de reinicio de mailboxes.

Parámetro	Tipo	Descripción
*p_mailbox	can_mb_conf	Puntero a una estructura de configuración de mailbox.

No devuelve nada. Únicamente fija a 0 todos los campos de la estructura.

4.4.3. Init_Reception

Realiza la configuración del mailbox 0 del periférico CAN que se le indique para que funcione en modo de recepción. Recibe como argumento:

Tabla 4-39. Parámetros que recibe la función para iniciar la recepción de mensajes.

Parámetro	Tipo	Descripción
*p_can	Can	Puntero a una instancia de periférico CAN. (CAN0 o CAN1).

Esta función hace uso de la función previamente descrita 'reset_mailbox_conf' para reiniciar el mailbox 0 del periférico CAN que se haya pasado como parámetro. Posteriormente, se configuran distintos parámetros de la estructura de configuración para que el mailbox funcione en modo de recepción.

Finalmente, llama a la función 'can_mailbox_init' desarrollada por Atmel para iniciar el funcionamiento del mailbox 0 y activa las interrupciones asociadas al mailbox 0.

No devuelve nada.

4.4.4. Configure_CAN_Tx

Configura el mailbox 1 del periférico CAN indicado en modo de transmisión. Recibe como parámetro:

Tabla 4-40. Parámetros que recibe la función para iniciar la transmisión de mensajes.

Parámetro	Tipo	Descripción
*p_can	Can	Puntero a una instancia de periférico CAN. (CAN0 o CAN1).

Al igual que 'init_Reception', se hace uso de la función 'reset_mailbox_conf' para reiniciar el mailbox 1 y posteriormente configurar los datos necesarios de la estructura asociada al mailbox 1.

Por último, llama a la función 'can_mailbox_init' para iniciar el funcionamiento del mailbox 1.

No devuelve ningún valor.

4.4.5. Send_generic_msg

Función para enviar tramas a través del bus CAN usando el mailbox 1 previamente configurado. La función recibe como parámetros:

Tabla 4-41. Parámetros que recibe la función para realizar la transmisión de un mensaje.

Parámetro	Tipo	Descripción
*p_can	Can	Puntero a una instancia de periférico CAN. (CAN0 o CAN1).
frame	CAN_Frame_t	Estructura con los campos necesarios para enviar datos a través del CAN.

Esta función se llama con mayor frecuencia puesto que es la que se emplea para enviar mensajes.

En primer lugar, se configuran los campos del mailbox 1 que contienen los datos que se pretenden enviar. Estos datos son los contenidos en la estructura pasada como parámetro 'frame'.

Posteriormente, se llama a la función 'can_mailbox_write' desarrollada por Atmel para modificar los campos de la estructura del mailbox 1.

Finalmente, se envía la trama usando la función 'can_global_send_transfer_cmd', también desarrollada por Atmel.

La función implementada no devuelve ningún valor.

4.4.6. Can_mailbox_get_recvId

Función que se usa para obtener el ID que se ha recibido por el bus. Recibe los parámetros:

Tabla 4-42. Parámetros que recibe la función para identificar el ID recibido.

Parámetro	Tipo	Descripción
*p_can	Can	Puntero a una instancia de periférico CAN. (CAN0 o CAN1).
uc_index	uint8_t	Entero para indicar el mailbox que se desea leer.

Devuelve una variable de tipo uint32_t con el ID CAN versión A almacenado en el mailbox indicado por uc_index.

La función obtiene en una variable el registro CAN_MIDx, donde se almacena el ID del mailbox x asociado a la instancia CAN proporcionada.

Finalmente, devuelve el valor del campo que almacena el ID obtenido.

5. PRUEBAS Y RESULTADOS

La validación del firmware se ha realizado haciendo uso de un prototipo denominado FlatSat, esto es una versión en la que se despliega cada subsistema que compone el CubeSat sobre una mesa y así poder validar la funcionalidad y prestaciones del satélite, previo al montaje del prototipo de vuelo. Este procedimiento de validación ha consistido en la ejecución de distintas pruebas para comprobar el correcto funcionamiento, en especial la comunicación con el OBC a través del bus CAN.

En una primera parte del capítulo se presentan los elementos usados en el FlatSat. Posteriormente, se recogen los resultados obtenidos tras la ejecución de cada prueba.

5.1. Montaje y software empleado para las pruebas

Todo el desarrollo se ha realizado usando la placa Arduino DUE. El motivo de la elección se debe a que el microcontrolador que integra esta placa es el mismo que el que se empleará en la carga útil desarrollada por el Grupo de Ingeniería Electrónica. Además, el uso de una placa ampliamente comercializada facilita el desarrollo debido a la gran cantidad de información y soporte disponible a través de la comunidad de usuarios que tiene.

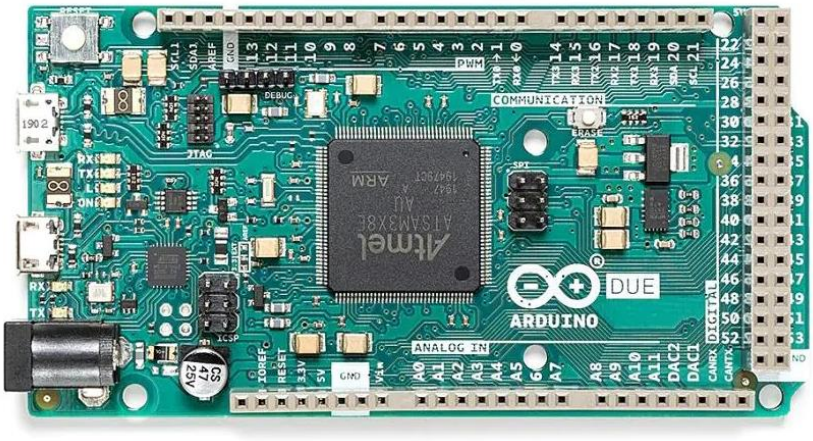

Figura 5-1. Placa Arduino DUE empleada durante el desarrollo.

El microcontrolador ATSAM3X8E implementa un periférico de CAN que requiere para su interconexión con otros dispositivos de una capa física externa. Por tanto, es necesario emplear un circuito integrado externo que permita convertir las señales digitales generadas por el microcontrolador a los niveles físicos que requiere el bus CAN, un transceptor. Se ha empleado el modelo SN65HVD230.

Figura 5-2. Transceptor CAN modelo SN65HVD230 empleado junto con el Microcontrolador.

El software empleado para el desarrollo ha sido Microchip Studio (antiguamente Atmel Studio). Este IDE permite compilar el firmware desarrollado para el microcontrolador, facilitando el uso de las librerías suministradas por Microchip. Además, permite configurar la estructura del proyecto para trabajar con FreeRTOS.

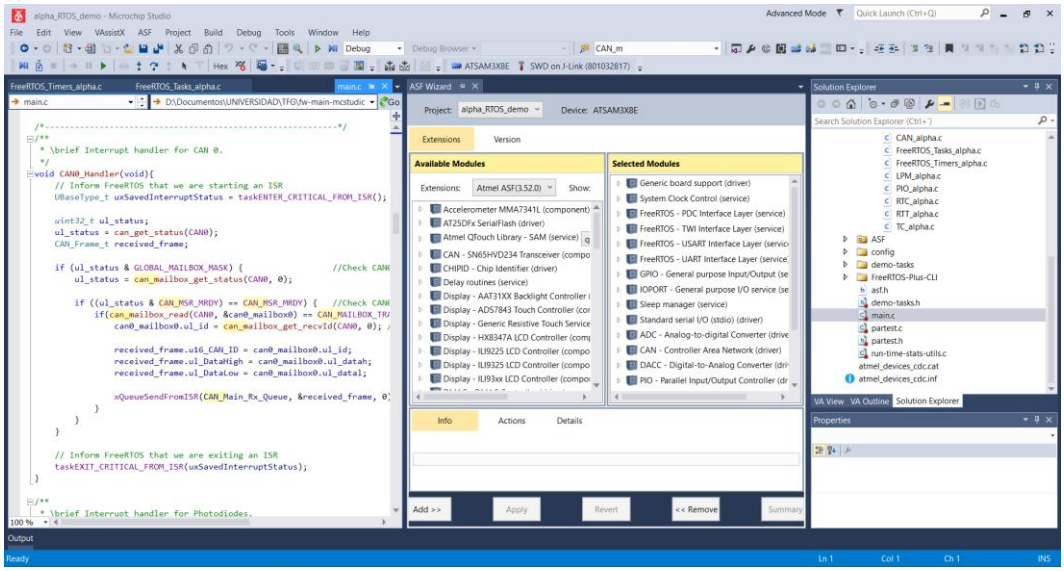

Figura 5-3. Captura de pantalla de la interfaz de usuario del IDE Microchip Studio.

El proceso de depuración y programación del software en el microcontrolador se ha realizado haciendo uso del puerto JTAG incorporado en la placa de Arduino y conectando la placa al ordenador mediante un depurador/programador de la empresa SEGGER, concretamente el modelo J-Link Edu Mini.

Figura 5-4. Depurador empleado para el desarrollo.

Para comprobar la correcta transmisión de los mensajes, se ha hecho uso de un dispositivo capaz de

monitorizar los mensajes CAN que se transmiten a través de un bus, para verificar que el comportamiento de la carga útil es el adecuado y a su vez, poder simular el comportamiento del OBC durante la misión.

El dispositivo empleado es un adaptador CAN a USB de la empresa PEAK System.

Figura 5-5. Adaptador CAN a USB para monitorización de mensajes.

Haciendo uso de este dispositivo, se podrán visualizar, de manera cómoda, los mensajes que viajan a través del bus desde un ordenador usando el software PCAN-View.

Stopped	33.2581 s	0.01 %	Ring Buffer	Rx: 8	Tx: 4
Time	CAN-ID	Rx/Tx	Type	Length	Data
3.5905	105h	Tx	Data	8	66 3B 22 BA 00 00 00 00
15.2888	100h	Rx	Data	8	00 50 00 00 1E 00 0A 00
17.7704	108h	Tx	Data	1	05
17.7719	104h	Rx	Data	8	05 00 00 00 00 00 00 00
17.7787	103h	Rx	Data	8	72 3B 22 BA 00 55 44 00
29.1704	108h	Tx	Data	1	05
29.1721	104h	Rx	Data	8	05 00 00 00 00 00 00 00
29.1789	103h	Rx	Data	8	7B 3B 22 BA 01 DD 70 00

Figura 5-6. Ejemplo de visualización de mensajes en el software para ordenador usando PCAN-View.

Por último, el montaje de todos estos componentes se puede apreciar en la Figura 5-7, donde se observa el uso de una placa de prototipado para su interconexión.

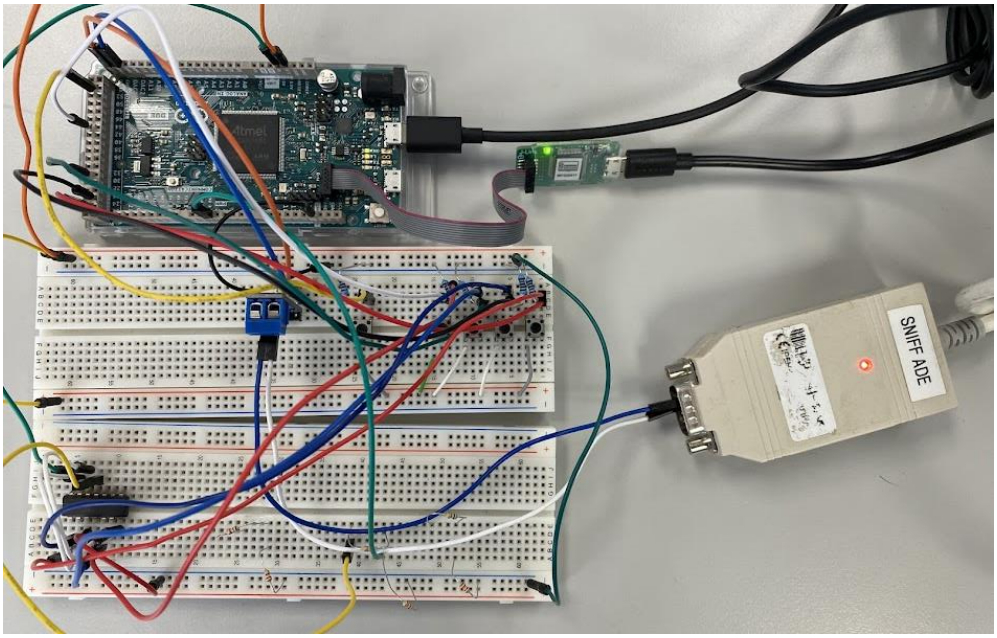

Figura 5-7. Configuración de los distintos dispositivos para realizar las pruebas de comunicación.

5.2. Pruebas realizadas

En esta sección, se muestran las distintas pruebas realizadas para validar el correcto funcionamiento de la arquitectura desarrollada, comprobando específicamente la correcta transmisión y retransmisión de mensajes simulando distintas situaciones que pueden ocurrir durante la misión.

Se han realizado un total de 7 pruebas, enumeradas a continuación:

1. Actualización de la marca de tiempo.
2. Retransmisión exitosa.
3. Retransmisión fallida.
4. Envío de Heart Beat.
5. Solicitud de datos por parte del OBC.
6. Acceso a modo de bajo consumo tras falta de asentimientos.
7. Solicitud de acceso a modo de bajo consumo.

En las siguientes secciones se describe el procedimiento llevado a cabo en cada prueba y los resultados obtenidos.

5.2.1. Actualización de la marca de tiempo

Prueba para comprobar que la carga útil actualiza su reloj de tiempo real al recibir el correspondiente mensaje por parte del ordenador de a bordo. Se espera que una vez recibido el mensaje del OBC para actualizar el reloj (0x105), los siguientes mensajes que envíe la carga útil contengan la marca de tiempo actualizada.

En la Figura 5-8 se puede apreciar el resultado de la prueba:

1. La carga útil se encuentra enviando datos de sensores usando una marca de tiempo desactualizada.
2. El OBC envía una trama de actualización de marca de tiempo (0x105) con la hora actualizada.
3. La carga útil continúa con el envío de mensajes habiendo actualizado su reloj de tiempo real.

Paused	37.7860 s	0.01 %	Ring Buffer	Rx: 11	Tx: 1
Time	CAN-ID	Rx/Tx	Type	Length	Data
0.9111	103h	Rx	Data	8	00 4F 98 45 00 00 00 00
3.5772	102h	Rx	Data	8	02 4F 98 45 01 88 B5 02
11.8695	102h	Rx	Data	8	09 4F 98 45 02 04 36 0B
14.5885	102h	Rx	Data	8	0B 4F 98 45 03 31 50 20
16.9036	102h	Rx	Data	8	0D 4F 98 45 04 99 6B 21
19.8101	105h	Tx	Data	8	66 3B 22 BA 00 00 00 00
23.7217	102h	Rx	Data	8	69 3B 22 BA 05 A3 79 2C
25.1938	102h	Rx	Data	8	6A 3B 22 BA 06 B5 51 25
30.9149	100h	Rx	Data	8	00 50 00 00 1E 00 0A 00
34.4070	102h	Rx	Data	8	72 3B 22 BA 07 3A 15 2A
36.3050	102h	Rx	Data	8	74 3B 22 BA 08 57 0D 03
37.7860	102h	Rx	Data	8	75 3B 22 BA 09 8E 81 07

Figura 5-8. Mensajes transmitidos durante la prueba de actualización de la marca de tiempo.

Tras la prueba, se verifica el correcto funcionamiento de la identificación de mensajes por parte de la carga útil y de la actualización de la marca de tiempo.

5.2.2. Retransmisión exitosa

Se simulará un fallo de recepción por parte del OBC en el que **se pierde un mensaje que la carga útil mantiene en la cola de retransmisión**, se espera que una vez recibido el NACK, la carga útil reenvíe todas las tramas que envió desde el número de secuencia indicado en el NACK.

En la Figura 5-9 se puede observar el resultado de la prueba, en la que ocurre lo siguiente:

1. El OBC envía trama para actualizar la marca de tiempo (ID 0x105) con la marca de tiempo.
2. La carga útil comienza enviando datos desde el número de secuencia 00.
3. Tras cierto tiempo, el OBC envía trama NACK (ID 0x10A) indicando que no ha recibido el mensaje con número de secuencia 04
4. La carga útil responde con el reenvío de todas las tramas que había enviado desde la número 04.
5. La comunicación continúa con el envío de mensajes siguiendo la secuencia.

Stopped	107.6463 s	0.07 %	Ring Buffer	Rx: 62	Tx: 3	St
Time	CAN-ID	Rx/Tx	Type	Length	Data	
1.9698	105h	Tx	Data	8	66 3B 22 BA 00 00 00 00	
3.6440	101h	Rx	Data	8	67 3B 22 BA 00 D0 07 00	
5.6441	101h	Rx	Data	8	68 3B 22 BA 01 A0 0F 00	
7.6442	101h	Rx	Data	8	6A 3B 22 BA 02 70 17 00	
9.6442	101h	Rx	Data	8	6C 3B 22 BA 03 40 1F 00	
11.6444	101h	Rx	Data	8	6D 3B 22 BA 04 10 27 00	
13.6444	101h	Rx	Data	8	6F 3B 22 BA 05 E0 2E 00	
15.6444	101h	Rx	Data	8	71 3B 22 BA 06 B0 36 00	
17.6442	101h	Rx	Data	8	72 3B 22 BA 07 80 3E 00	
19.6444	101h	Rx	Data	8	74 3B 22 BA 08 50 46 00	
20.8095	10Ah	Tx	Data	1	04	
20.8165	101h	Rx	Data	8	6D 3B 22 BA 04 10 27 00	
20.8175	101h	Rx	Data	8	6F 3B 22 BA 05 E0 2E 00	
20.8185	101h	Rx	Data	8	71 3B 22 BA 06 B0 36 00	
20.8195	101h	Rx	Data	8	72 3B 22 BA 07 80 3E 00	
20.8205	101h	Rx	Data	8	74 3B 22 BA 08 50 46 00	
21.6445	101h	Rx	Data	8	76 3B 22 BA 09 20 4E 00	
23.6445	101h	Rx	Data	8	77 3B 22 BA 0A F0 55 00	
25.6447	101h	Rx	Data	8	79 3B 22 BA 0B C0 5D 00	
27.6447	101h	Rx	Data	8	7B 3B 22 BA 0C 90 65 00	
29.6447	101h	Rx	Data	8	7C 3B 22 BA 0D 60 6D 00	
31.6446	101h	Rx	Data	8	7E 3B 22 BA 0E 30 75 00	

Figura 5-9.Mensajes transmitidos durante la prueba de retransmisión exitosa.

Se valida el correcto funcionamiento de la retransmisión de mensajes por parte de la carga útil.

5.2.3. Retransmisión fallida

Se simulará un fallo de recepción por parte del OBC en el que **se pierde un mensaje que la carga útil no mantiene en la cola de retransmisión**, se espera que una vez recibido el NACK, la carga útil indique el nuevo número de secuencia para que el OBC se sincronice y continúe enviando datos.

En la Figura 5-10 se puede observar el resultado de la prueba, en la que ocurre lo siguiente:

1. La carga útil se encuentra enviando mensajes con datos de medidas del sensor de radiación total.
2. Tras 42 mensajes (0x2A), el OBC envía una trama NACK indicando que se le reenvíen los mensajes desde el número de secuencia 04.
3. Dado que la cola de retransmisión almacena 10 mensajes, ese mensaje ya no se encuentra en la cola. La carga útil envía trama con identificador 0x10B indicando el número de secuencia que debe de esperar en el siguiente mensaje el OBC.
4. La carga útil continúa con el envío de mensajes, empleando el número de secuencia indicado en el mensaje anterior.

Stopped	107.6463 s	0.07 %	Ring Buffer	Rx: 62	Tx: 3
Time	CAN-ID	Rx/Tx	Type	Length	Data
73.6455	101h	Rx	Data	8	A1 3B 22 BA 23 40 19 01
75.6455	101h	Rx	Data	8	A3 3B 22 BA 24 10 21 01
77.6456	101h	Rx	Data	8	A5 3B 22 BA 25 E0 28 01
79.6456	101h	Rx	Data	8	A6 3B 22 BA 26 B0 30 01
81.6458	101h	Rx	Data	8	A8 3B 22 BA 27 80 38 01
83.6458	101h	Rx	Data	8	AA 3B 22 BA 28 50 40 01
85.6458	101h	Rx	Data	8	AB 3B 22 BA 29 20 48 01
87.6459	101h	Rx	Data	8	AD 3B 22 BA 2A F0 4F 01
88.9799	10Ah	Tx	Data	1	04
88.9810	10Bh	Rx	Data	8	AE 3B 22 BA 2B 00 00 00
89.6460	101h	Rx	Data	8	AF 3B 22 BA 2B C0 57 01
91.6460	101h	Rx	Data	8	B0 3B 22 BA 2C 90 5F 01
91.6550	100h	Rx	Data	8	02 50 00 00 1E 00 0A 00
93.6462	101h	Rx	Data	8	B2 3B 22 BA 2D 60 67 01
95.6462	101h	Rx	Data	8	B4 3B 22 BA 2E 30 6F 01
97.6461	101h	Rx	Data	8	B5 3B 22 BA 2F 00 77 01
99.6461	101h	Rx	Data	8	B7 3B 22 BA 30 D0 7E 01
101.6462	101h	Rx	Data	8	B9 3B 22 BA 31 A0 86 01
103.6463	101h	Rx	Data	8	BB 3B 22 BA 32 70 8E 01

Figura 5-10. Mensajes transmitidos durante la prueba de transmisión fallida.

Se valida el correcto comportamiento de la carga útil en caso de recibir un NACK de un mensaje que ya no se encuentra en la cola de retransmisión.

5.2.4. Envío de Heart Beat

La carga útil debe enviar cada 30 segundos un mensaje al OBC para indicar que se encuentra activa y que no ha ocurrido ningún fallo de funcionamiento. En esta prueba, se analizarán los mensajes enviados por la carga útil para comprobar que se envían los mensajes de tipo Heat Beat con identificador 0x100.

En la Figura 5-11 se muestra el resultado obtenido, habiendo ocurrido lo siguiente:

1. El OBC envía trama para actualizar marca de tiempo
2. La carga útil envía datos de sensores.
3. En el segundo 31, la carga útil envía un mensaje de tipo Heart Beat (0x100).
4. 30 segundos más tarde, en el segundo 61, la carga útil vuelve a enviar el mensaje con ID 0x100.

Stopped	76.1649 s	0.01 %	Ring Buffer	Rx: 7	Tx: 1
Time	CAN-ID	Rx/Tx	Type	Length	Data
3.9308	105h	Tx	Data	8	66 3B 22 BA 00 00 00 00
16.1634	101h	Rx	Data	8	70 3B 22 BA 00 98 3A 00
31.1637	101h	Rx	Data	8	7C 3B 22 BA 01 30 75 00
31.1727	100h	Rx	Data	8	00 50 00 00 1E 00 0A 00
46.1641	101h	Rx	Data	8	89 3B 22 BA 02 C8 AF 00
61.1644	101h	Rx	Data	8	95 3B 22 BA 03 60 EA 00
61.1734	100h	Rx	Data	8	01 50 00 00 1E 00 0A 00
76.1649	101h	Rx	Data	8	A2 3B 22 BA 04 F8 24 01

Figura 5-11. Mensajes transmitidos durante la prueba de Heart Beat.

Se verifica el correcto envío de tramas de 'Housekeeping' o Heart Beat por parte de la carga útil.

5.2.5. Solicitud de datos por parte del OBC

El ordenador de a bordo puede solicitar a la carga útil el envío de datos de alguno de los sensores. En esta prueba, se simulará la solicitud de datos por parte del OBC y se comprobará si la carga útil responde con las tramas correctas.

En la Figura 5-12 se pueden consultar los resultados que ahora se detallan:

1. El OBC envía trama para actualizar marca de tiempo
2. El OBC envía trama de solicitud de medida del sensor de SEUs (ID 0x108 con valor 05).
3. La carga útil responde enviando un ACK (0x104) y seguidamente envía el mensaje con la medida.

Stopped	33.2581 s	0.01 %	Ring Buffer	Rx: 8	Tx: 4
Time	CAN-ID	Rx/Tx	Type	Length	Data
3.5905	105h	Tx	Data	8	66 3B 22 BA 00 00 00 00
15.2888	100h	Rx	Data	8	00 50 00 00 1E 00 0A 00
17.7704	108h	Tx	Data	1	05
17.7719	104h	Rx	Data	8	05 00 00 00 00 00 00 00
17.7787	103h	Rx	Data	8	72 3B 22 BA 00 55 44 00

Figura 5-12. Mensajes enviados durante la prueba de solicitud de datos por parte del OBC.

Se verifica la correcta respuesta de la carga útil ante la solicitud de medidas.

5.2.6. Acceso a modo de bajo consumo tras falta de asentimientos

La carga útil también debe conocer el estado en el que se encuentra el ordenador de a bordo. Por ello, cada vez que envía una trama de tipo Heart Beat (ID 0x100), la carga útil espera recibir por parte del OBC un mensaje de asentimiento a este Heart Beat (ID 0x109) con la que se le indique que el OBC está funcionando correctamente.

En caso de que el OBC no responda a 10 mensajes de tipo Heart Beat de manera consecutiva, la carga útil pasará a modo de bajo consumo durante 2 minutos.

En la Figura 5-13 se aprecia el intercambio de mensajes de la prueba realizada:

1. La carga útil envía un par de tramas Heart Beat (0x100) las cuales son asentidas por parte del OBC (0x109).
2. Se simula un fallo en el OBC.
3. La carga útil envía 10 tramas tipo Heart Beat (0x100) sin recibir su correspondiente asentimiento.
4. Tras la décima trama, la carga útil pasa a modo de bajo consumo durante 120 segundos.
5. Pasado el tiempo, la carga útil continúa con el envío de mensajes.

Stopped	538,9218 s	0,02 %	Ring Buffer	Rx: 14	Tx: 2	
Time	CAN-ID	Rx/Tx	Type	Length	Data	
30,0560	100h	Rx	Data	8	00 50 00 00 1E 00 0A 00	
30,5673	109h	Tx	Data	1	00	
60,0568	100h	Rx	Data	8	01 50 00 00 1E 00 0A 00	
60,7200	109h	Tx	Data	1	01	
90,0576	1	100h	Rx	Data	8	02 50 00 00 1E 00 0A 00
120,0586	2	100h	Rx	Data	8	03 50 00 00 1E 00 0A 00
150,0594	3	100h	Rx	Data	8	04 50 00 00 1E 00 0A 00
180,0603	4	100h	Rx	Data	8	05 50 00 00 1E 00 0A 00
210,0612	5	100h	Rx	Data	8	06 50 00 00 1E 00 0A 00
240,0621	6	100h	Rx	Data	8	07 50 00 00 1E 00 0A 00
270,0629	7	100h	Rx	Data	8	08 50 00 00 1E 00 0A 00
300,0638	8	100h	Rx	Data	8	09 50 00 00 1E 00 0A 00
330,0646	9	100h	Rx	Data	8	0A 50 00 00 1E 00 0A 00
360,0655	10	100h	Rx	Data	8	0B 50 00 00 1E 00 0A 00
508,9209		100h	Rx	Data	8	0C 50 00 00 1E 00 0A 00
538,9218		100h	Rx	Data	8	0D 50 00 00 1E 00 0A 00

Figura 5-13. Mensajes enviados durante la prueba acceso a modo de bajo consumo debido a la falta de asentimientos.

Se verifica que la carga útil se comporta como se desea, comprobando si el OBC confirma la recepción de mensajes de tipo Heart Beat y accediendo a modo de bajo consumo en caso de detectar fallo en el OBC. En la Figura 5-15 pueden observar los consumos en modo normal de funcionamiento y en modo de bajo consumo.

5.2.7. Solicitud de acceso a modo de bajo consumo

Otra de las solicitudes que puede hacer el ordenador de a bordo, es solicitar a la carga útil a entrar en modo de bajo consumo. Se simula el envío de esta trama de petición (0x106) por parte el OBC y se espera que la carga útil acceda a modo de bajo consumo durante 60 segundos.

En la Figura 5-14 se aprecia el intercambio de mensajes, detallado a continuación:

1. La carga útil se encuentra enviando datos al OBC.
2. El OBC envía la petición de acceso al modo de bajo consumo (ID 0x106).
3. La carga útil accede a modo de bajo consumo durante 60 segundos.
4. Tras los 60 segundos, continúa con el envío de mensajes.

Stopped	76,6130 s	0,01 %	Ring Buffer	Rx: 5	Tx: 1
Time	CAN-ID	Rx/Tx	Type	Length	Data
9,0589	101h	Rx	Data	8	03 4F 98 45 00 D0 07 00
11,0590	101h	Rx	Data	8	05 4F 98 45 01 A0 0F 00
13,0591	101h	Rx	Data	8	07 4F 98 45 02 70 17 00
13,9348	106h	Tx	Data	0	
74,6130	101h	Rx	Data	8	3A 4F 98 45 03 40 1F 00
76,6130	101h	Rx	Data	8	3B 4F 98 45 04 10 27 00

Figura 5-14. Mensajes enviados durante la prueba de acceso a modo de bajo consumo.

La carga útil accede correctamente a modo de bajo consumo durante 60 segundos. Este tiempo siempre puede modificarse, dependiendo de los requisitos establecidos. En la Figura 5-15 pueden observar los consumos en modo normal de funcionamiento y en modo de bajo consumo.

Figura 5-15. Consumos del Arduino en modo normal y en modo de bajo consumo del SAM3X8E.

6. CONCLUSIONES Y LÍNEAS FUTURAS

El análisis del contexto de la Misión Alpha ha resultado de gran interés para profundizar en el concepto del NewSpace, una nueva filosofía empresarial del sector espacial que va más allá del uso de CubeSats para misiones espaciales.

Una vez expuesto el propósito de la Misión Alpha, se ha presentado la carga útil y los experimentos diseñados por el Grupo de Ingeniería Electrónica, que serán de gran utilidad para analizar efectos dañinos de la radiación espacial en componentes electrónicos, un problema creciente debido al auge de las misiones espaciales provocado por el NewSpace.

Respecto a los objetivos del trabajo, se ha diseñado con éxito una primera versión de la lógica de comunicación entre la carga útil y el ordenador de a bordo. Se han establecido los distintos mensajes a emplear en la comunicación y previo al desarrollo del sistema, se han mostrado los diagramas de paso de mensajes, para entender de forma gráfica la dinámica de la comunicación y facilitar el diseño del sistema de tiempo real para la gestión de mensajes.

Desarrollar el controlador para la comunicación empleando el periférico CAN ha servido para aprender el flujo de trabajo empleando un IDE como Microchip Studio, comprendiendo la estructura de ficheros que sigue un proyecto de software para microcontroladores y aprendiendo buenas prácticas durante el desarrollo. También ha sido un gran aprendizaje el uso de herramientas de depuración para la detección de fallos en el código.

El uso de un sistema operativo de tiempo real ha facilitado en gran medida el diseño de la arquitectura del sistema completo, aportando gran modularidad y permitiendo un proceso de desarrollo más fluido debido al uso de funciones en C para la descripción de cada una de las tareas.

Para concluir, las pruebas se han realizado con éxito, verificando la correcta implementación de la arquitectura del sistema cumpliendo con la lógica de envío establecida y con un funcionamiento acorde a lo mostrado en los diagramas de paso de mensajes.

6.1. Líneas futuras

En esta primera versión del software de la carga útil se ha diseñado un sistema capaz de establecer una comunicación básica con el ordenador de a bordo que garantiza la correcta transmisión de los datos medidos y permite la detección de fallos mediante el uso de tramas de tipo ‘Heart Beat’.

Una futura versión, podría implementar un mayor número de mensajes enfocados en el envío de tele comandos que permitan más opciones de control al ordenador de a bordo.

Por otra parte, sería interesante profundizar en la implementación de los modos de bajo consumo para poder así emplear otras fuentes de activación del modo normal de funcionamiento y proporcionando al OBC un mayor control sobre el consumo de la carga útil.

También resultaría de gran interés realizar pruebas en un entorno real, como las instalaciones CHARM del CERN sometiendo al sistema completo a pruebas de radiación. Podría ser de espacial relevancia para detectar posibles fallos de firmware que no son posibles de recrear en un entorno convencional de laboratorio.

REFERENCIAS

[1] A. Golkar, S. Member, and A. Salado, "Definition of New Space-Expert Survey Results and Key Technology Trends," *IEEE JOURNAL ON MINIATURIZATION FOR AIR AND SPACE SYSTEMS*, vol. 2, no. 1, 2021, doi: 10.1109/JMASS.2020.3045851.

[2] H. Bokil, *COTS Semiconductor Components for the New Space Industry*.

[3] "Annual number of objects launched into space." Accessed: Jul. 09, 2024. [Online]. Available: https://ourworldindata.org/grapher/yearly-number-of-objects-launched-into-outer-space?time=2001..latest#sources-and-processing

[4] "Órbita terrestre baja - Wikipedia, la enciclopedia libre." Accessed: Jun. 28, 2024. [Online]. Available: https://es.wikipedia.org/wiki/%C3%93rbita_terrestre_baja

[5] "What Is a Van Allen Radiation Belt? | HowStuffWorks." Accessed: Jun. 28, 2024. [Online]. Available: https://science.howstuffworks.com/dictionary/astronomy-terms/van-allen-radiation-belts-info.htm

[6] "Entrevista a Jordi Puig-Suari - TEDAE." Accessed: Jul. 02, 2024. [Online]. Available: https://tedae.org/uncategorized/entrevista-a-jordi-puig-suari/

[7] "File:SamSat-ION in vertical test rig.jpg - Wikimedia Commons." Accessed: Jul. 09, 2024. [Online]. Available: https://commons.m.wikimedia.org/wiki/File:SamSat-ION_in_vertical_test_rig.jpg#filelinks

[8] "'Mision Alpha' DESARROLLO DE UN CUBESAT CON TECNOLOGÍA ANDALUZA - SmartCityCluster." Accessed: Jul. 02, 2024. [Online]. Available: https://smartcitycluster.org/proyectos-2023/mision-alpha-desarrollo-de-un-cubesat-con-tecnologia-andaluza/

[9] R. Li *et al.*, "Impact of TID on the transient ionizing irradiation response of CMOS circuits," *2018 International Conference on Radiation Effects of Electronic Devices, ICREED 2018*, May 2018, doi: 10.1109/ICREED.2018.8905056.

[10] "Seminario Misión Alpha_06032024_GIE_US - YouTube." Accessed: Jul. 02, 2024. [Online]. Available: https://www.youtube.com/watch?v=8XaYS_pv15s&t=442s

[11] K. A. Label, "Single Event Effects (SEEs) Specification Approach Co-Manager, NASA Electronic Parts and Packaging (NEPP) Program Group Leader, Radiation Effects and Analysis Group (REAG), NASA/GSFC Project Technologist, Living With a Star (LWS) Space Environment Testbeds (SET)".

[12] "Overview of the CAN Bus Protocol - Embedded - Electronic Component and Engineering Solution Forum - TechForum | DigiKey." Accessed: Jul. 05, 2024. [Online]. Available: https://forum.digikey.com/t/overview-of-the-can-bus-protocol/21170

[13] M. Dominguez-Morales *et al.*, "An AER to CAN bridge for spike-based robot control," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, pp. 124–132. doi: 10.1007/978-3-642-21501-8_16.

[14] "CAN bus - Wikipedia." Accessed: Jul. 06, 2024. [Online]. Available: https://en.wikipedia.org/wiki/CAN_bus

[15] A. K. El Allam, A. H. M. Jallad, M. Awad, M. Takruri, and P. R. Marpu, "A Highly Modular Software Framework for Reducing Software Development Time of Nanosatellites," *IEEE Access*, vol. 9, pp. 107791–107803, 2021, doi: 10.1109/ACCESS.2021.3097537.

[16] "RTOS Task Scheduling and Prioritization." Accessed: Jul. 08, 2024. [Online]. Available: https://www.digikey.ee/en/maker/projects/introduction-to-rtos-solution-to-part-3-task-scheduling/8fbb9e0b0eed4279a2dd698f02ce125f

[17] “FreeRTOS Queue Example.” Accessed: Jul. 08, 2024. [Online]. Available: https://www.digikey.com/en/maker/projects/introduction-to-rtos-solution-to-part-5-freertos-queue-example/72d2b361f7b94e0691d947c7c29a03c9

[18] “FreeRTOS task notifications, fast Real Time Operating System (RTOS) event mechanism.” Accessed: Jul. 08, 2024. [Online]. Available: https://www.freertos.org/RTOS-task-notifications.html

GLOSARIO

ARM
Advanced RISC Machine 8
CAN
Controller Area Networkxi, xiii, xiv, xv, xvi, xvii, xix, 4, 8, 12, 13, 14, 15, 19, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 37, 38, 39, 41, 42, 43, 51, 53
COTS
Commercial Off The Shelf 1, 53
ESA
European Space Agency 1
FIFO
First In First Out 20
IDE
Integrated Development Environment 41, 42, 51
LEO
Low Earth Orbit 2
MCU
Microcontroller Unit 7
MOS
Metal-Oxide-Semiconductor 7, 9
NASA
National Aeronautics and Space Administration 1, 53
OBC
On Board Computerxv, xvi, xvii, xix, 4, 15, 16, 17, 18, 21, 23, 24, 25, 26, 27, 28, 29, 31, 36, 37, 42, 44, 45, 46, 47, 48, 49, 51
PMC
Power Management Controller 37
RADFET
Radiation Field Effect Transistorxvi, xix, 9, 23, 24, 30, 33, 35, 36
RTC
Real Time Clock xix, 8, 18, 23, 33, 36
RTOS
Real Time Operating System 19, 53, 54
RTT
Real Time Timer 8
SEE
Single Effect Events xiv, 7
SET
Single Event Transient 9, 53
SEU
Single Event Upset 7, 10
SRAM
Static Random Access Memoryxiv, 8, 10, 15, 16
TID
Total Ionizing Dose xiv, xix, 7, 9, 53
TQ
Time Quanta 14
UART
Universal Asynchronous Receiver/Transmitter 8

ANEXO: CÓDIGOS

- Configuración del periférico CAN.

```
/**
* \brief Initialize CAN interface
*
* \param p_can    Pointer to a CAN peripheral instance.
* \param ul_sysclk System Main clock frequency
* \param baudrate Baudarte desired for CAN Transmission
*
* \retval 1 if there where no errors during setup.
*/
uint8_t CAN_initialization(Can *p_can, uint32_t ul_sysclk, uint32_t baudrate){

    uint8_t correct_initialization;
    uint8_t CAN_IRQn;

    //Configure CAN pins.
    gpio_configure_pin(PIO_PA1_IDX, PIN_CAN0_RX_FLAGS);
    gpio_configure_pin(PIO_PA0_IDX, PIN_CAN0_TX_FLAGS);

    //CAN0 PMC Enable
    if(p_can == CAN0){
        pmc_enable_periph_clk(ID_CAN0);
        CAN_IRQn = CAN0_IRQn;
    }
    if(p_can == CAN1){
        pmc_enable_periph_clk(ID_CAN1);
        CAN_IRQn = CAN1_IRQn;
    }

    if (can_init(p_can, ul_sysclk, baudrate)) {
        //printf("CAN initialization completed.\n");

        /* Disable all CAN0 interrupt. */
        can_disable_interrupt(p_can, CAN_DISABLE_ALL_INTERRUPT_MASK);

        /* Configure and enable interrupt of CAN0 as it will be receiving data*/
        NVIC_DisableIRQ(CAN_IRQn); /* Disable TC interrupts*/
        NVIC_ClearPendingIRQ(CAN_IRQn); /* Clear pending interrupt */
        NVIC_SetPriority(CAN_IRQn, configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY); /* Set interrupt
priority */
        NVIC_EnableIRQ(CAN_IRQn);

        /*Reset mailbox*/
        can_reset_all_mailbox(p_can);
        correct_initialization = 1;

    }else{
        correct_initialization = 0;
    }

    return correct_initialization;
}
```

- Función para reinicio de mailbox CAN.

```
/**
 * \brief Reset mailbox configure structure.
 */
void reset_mailbox_conf(can_mb_conf_t *p_mailbox)
{
    p_mailbox->ul_mb_idx = 0;
    p_mailbox->uc_obj_type = 0;
    p_mailbox->uc_id_ver = 0;
    p_mailbox->uc_length = 0;
    p_mailbox->uc_tx_prio = 0;
    p_mailbox->ul_status = 0;
    p_mailbox->ul_id_msk = 0;
    p_mailbox->ul_id = 0;
    p_mailbox->ul_fid = 0;
    p_mailbox->ul_datal = 0;
    p_mailbox->ul_datah = 0;
}
```

- Configuración de Mailbox 0 para recepción.

```
/**
 * \brief Configure MB0 in CANx interface in Rx mode.
 */
void init_Reception(Can *p_can){
    /* Init CAN0 Mailbox 0 to Reception Mailbox. */
    reset_mailbox_conf(&can0_mailbox0);
    can0_mailbox0.ul_mb_idx = CAN_MB_ID(0);
    can0_mailbox0.uc_obj_type = CAN_MB_RX_MODE;
    can0_mailbox0.ul_id_msk = 0;
    can0_mailbox0.ul_id = 0; //Accept all ID from can version A.
    can_mailbox_init(CAN0, &can0_mailbox0);

    /* Enable CAN0 mailbox 0 interrupt. */
    can_enable_interrupt(CAN0, CAN_IER_MB0);
}
```

- Configuración de Mailbox 1 para transmisión.

```
/**
 * \brief Configure Mailbox 1 in CANx interface to transfer Data.
 */
void configure_CAN_Tx(Can *p_can){
    // Mailbox Initialization - MOVE THIS BLOCK TO ANOTHER FUNCTION
    reset_mailbox_conf(&can0_mailbox1);
    can0_mailbox1.ul_mb_idx = CAN_MB_ID(1);
    can0_mailbox1.uc_obj_type = CAN_MB_TX_MODE;
    can0_mailbox1.uc_tx_prio = MSG_RAM_ADDR_PRIO;
    can0_mailbox1.uc_id_ver = 0; //Version A ID. (11 bits)
    can0_mailbox1.ul_id_msk = 0; //Set to 0 to receive all the Frames.
    can_mailbox_init(p_can, &can0_mailbox1);
}
```

- Envío de mensaje a través de mailbox 1.

```
/**
 * \brief Transfer using CANx MB1 a RAM Address Frame.
 */
void send_generic_msg(Can *p_can, CAN_Frame_t frame){
    //Information in mailbox
    can0_mailbox1.ul_id = CAN_MID_MIDvA(frame.u16_CAN_ID);
    can0_mailbox1.ul_datal = frame.ul_DataLow;
    can0_mailbox1.ul_datah = frame.ul_DataHigh;
    can0_mailbox1.uc_length = MAX_CAN_FRAME_DATA_LEN;
    can_mailbox_write(p_can, &can0_mailbox1);

    can_global_send_transfer_cmd(p_can, CAN_TCR_MB1);
}
```

- Identificación del ID CAN recibido en el mailbox indicado.

```
/**
 * \brief Get ID of the message received at mailbox i.
 */
uint32_t can_mailbox_get_recvId(Can *p_can, uint8_t uc_index)
{
    //Getting CAN_MIDx register
    uint32_t ul_register = p_can->CAN_MB[uc_index].CAN_MID;
    //From CAN_MIDi register, get the IDvA.
    uint32_t ul_recv_id = (ul_register & CAN_MID_MIDvA_Msk) >> CAN_MID_MIDvA_Pos;

    return (ul_recv_id);
}
```

- Rutina de interrupción de periférico CAN 0.

```
/*-----------------------------------------------------------*/
/**
 * \brief Interrupt handler for CAN 0.
 */
void CAN0_Handler(void){
    // Inform FreeRTOS that we are starting an ISR
    UBaseType_t uxSavedInterruptStatus = taskENTER_CRITICAL_FROM_ISR();

    uint32_t ul_status;
    ul_status = can_get_status(CAN0);
    CAN_Frame_t received_frame;

    if (ul_status & GLOBAL_MAILBOX_MASK) {                  //Check CAN0 peripheral
        ul_status = can_mailbox_get_status(CAN0, 0);

        if ((ul_status & CAN_MSR_MRDY) == CAN_MSR_MRDY) {   //Check CAN0 MB0 for data
            if(can_mailbox_read(CAN0, &can0_mailbox0) == CAN_MAILBOX_TRANSFER_OK){
                can0_mailbox0.ul_id = can_mailbox_get_recvId(CAN0, 0); //Get ID

                received_frame.u16_CAN_ID = can0_mailbox0.ul_id;
                received_frame.ul_DataHigh = can0_mailbox0.ul_datah;
                received_frame.ul_DataLow = can0_mailbox0.ul_datal;

                xQueueSendFromISR(CAN_Main_Rx_Queue, &received_frame, 0);   //Send to queue
            }
        }
    }

    // Inform FreeRTOS that we are exiting an ISR
    taskEXIT_CRITICAL_FROM_ISR(uxSavedInterruptStatus);
}
```

- Tarea ‘CAN Framing’ para obtener datos de sensores y encapsular en trama CAN.

```
/*Task to read queues with data from sensors*/
void CAN_Framing_Task(void *parameters){
    uint32_t notified_value;

    SENSOR_Data_t Data_to_Frame;
    QUIESCENT_CURRENTS_Data_t Meas_to_Frame;

    CAN_Frame_t Frame_to_Send;
    uint8_t sequence_id;
    uint32_t data;


    while(1){
        xTaskNotifyWait(0x00, 0xFFFFFFFF, &notified_value, portMAX_DELAY);

        if( (notified_value & ACK_NOTIFY) != 0){
            if(xQueueReceive(CAN_PYL_ACK_Queue, &Data_to_Frame, portMAX_DELAY) == pdPASS){
                Frame_to_Send.u16_CAN_ID = Data_to_Frame.u16_CAN_ID;    //Set ID to send
                Frame_to_Send.ul_DataLow = (Data_to_Frame.ul_msgData & 0xFF);
                Frame_to_Send.ul_DataHigh = 0;

                xQueueSend(CAN_Main_Tx_Queue, &Frame_to_Send, portMAX_DELAY);
            }
        }

        vTaskDelay(1);
        if( (notified_value & RADFET_NOTIFY) != 0){
            if(xQueueReceive(RADFET_Data_Queue, &Data_to_Frame, portMAX_DELAY) == pdPASS){
                Frame_to_Send.u16_CAN_ID = Data_to_Frame.u16_CAN_ID;    //Set ID to send
                Frame_to_Send.ul_DataLow = Data_to_Frame.ul_TimeStamp;  //Set Timestamp to send

                sequence_id = Data_to_Frame.u_msgSeqId;
                data = Data_to_Frame.ul_msgData;
                Frame_to_Send.ul_DataHigh = (data << 8 | sequence_id);  //Get data into 32 bits.

                xQueueSend(CAN_Main_Tx_Queue, &Frame_to_Send, portMAX_DELAY);
            }
        }
        vTaskDelay(3);
        if( (notified_value & PHOTODIODES_NOTIFY) != 0){
            if(xQueueReceive(PHOTODIODE_Data_Queue, &Data_to_Frame, portMAX_DELAY) == pdPASS){
                Frame_to_Send.u16_CAN_ID = Data_to_Frame.u16_CAN_ID;    //Set ID to send
                Frame_to_Send.ul_DataLow = Data_to_Frame.ul_TimeStamp;  //Set Timestamp to send

                sequence_id = Data_to_Frame.u_msgSeqId;
                data = Data_to_Frame.ul_msgData;
                Frame_to_Send.ul_DataHigh = ((data << 8) & 0xFFFFFF00) | sequence_id;   //Get data into 32
bits.

                xQueueSend(CAN_Main_Tx_Queue, &Frame_to_Send, portMAX_DELAY);
            }
        }
        vTaskDelay(3);
        if( (notified_value & RAM_NOTIFY) != 0){
            if(xQueueReceive(RAM_Data_Queue, &Data_to_Frame, portMAX_DELAY) == pdPASS){
                Frame_to_Send.u16_CAN_ID = Data_to_Frame.u16_CAN_ID;    //Set ID to send
                Frame_to_Send.ul_DataLow = Data_to_Frame.ul_TimeStamp;  //Set Timestamp to send

                sequence_id = Data_to_Frame.u_msgSeqId;
                data = Data_to_Frame.ul_msgData;
                Frame_to_Send.ul_DataHigh = (data << 8 | sequence_id);  //Get data into 32 bits.

                xQueueSend(CAN_Main_Tx_Queue, &Frame_to_Send, portMAX_DELAY);
            }
        }
        vTaskDelay(3);
        if( (notified_value & QUIESCENT_CURRENTS_NOTIFY) != 0){
```

```
            if(xQueueReceive(QUIESCENT_CURRENTS_Data_Queue, &Meas_to_Frame, portMAX_DELAY) == pdPASS){
                Frame_to_Send.u16_CAN_ID = Meas_to_Frame.u16_CAN_ID;
                Frame_to_Send.ul_DataLow = (Meas_to_Frame.u16_current[2] << 8) |
                                           (Meas_to_Frame.u_msgSeqId);
                Frame_to_Send.ul_DataHigh = (Meas_to_Frame.u16_current[0] << 16) |
                                            (Meas_to_Frame.u16_current[1]);

                xQueueSend(CAN_Main_Tx_Queue, &Frame_to_Send, portMAX_DELAY);
            }
        }
        vTaskDelay(3);
    }
}
```

- Tarea ‘CAN Send’ para enviar datos a través del bus CAN.

```
/*Task that sends data when notified*/
void CAN_Send_Task(void * parameters){
    CAN_Frame_t Frame_to_Send;
    CAN_Frame_t Aux_Frame;  //Auxiliary frame to implement Circular buffer
    while (1){
        if(xQueueReceive(CAN_Main_Tx_Queue, &Frame_to_Send, portMAX_DELAY) == pdPASS){
            //Send via CAN Bus
            send_generic_msg(CAN0, Frame_to_Send);

            //Send Data Frame to RTX Queue, only if it's a frame with Data
            if(Frame_to_Send.u16_CAN_ID != HEARTBEAT_SENT && Frame_to_Send.u16_CAN_ID != PYL_ACK_SENT){
                if(uxQueueSpacesAvailable(CAN_RTX_Queue) != 0){
                    xQueueSend(CAN_RTX_Queue, &Frame_to_Send, portMAX_DELAY);
                }else{
                    //If the queue was full, get the first data out and put the new one.
                    xQueueReceive(CAN_RTX_Queue, &Aux_Frame, 0);
                    xQueueSend(CAN_RTX_Queue, &Frame_to_Send, portMAX_DELAY);
                }
            }
        }
        //while(xTimerStart(ACK_Timeout_Timer, 0) == pdFALSE);
        //vTaskDelay(10);   //PROVISIONAL Para que no bloquee otras tareas.
    }
}
```

- Tarea ‘CAN ACK Generate’ para generar tramas ACK de respuesta al OBC.

```
void CAN_ACK_Generate_Task(void *parameters){
    uint32_t notified_value;
    uint32_t notify_value = ACK_NOTIFY;
    SENSOR_Data_t Data_Frame;
    while (1)
    {
        xTaskNotifyWait(0x00, 0xFFFFFFFF, &notified_value, portMAX_DELAY);
        Data_Frame.u16_CAN_ID = PYL_ACK_SENT;
        Data_Frame.ul_msgData = notified_value;

        xQueueSend(CAN_PYL_ACK_Queue, &Data_Frame, portMAX_DELAY);
        xTaskNotify( CAN_Framing_Handler, notify_value, eSetBits );
    }
}
```

- Tarea ‘CAN Receive’ para procesar los mensajes recibidos.

```
//Task to get CAN Frames and send them to the correct queue
void CAN_Receive_Task(void *parameters){
    CAN_Frame_t Received_Frame;
    MSG_ID_OBC_TO_PYL_t Received_CAN_ID;
    QS_NOTIFY_t notify_value;
    uint32_t sleep_time;

    while(1){
        //Get the CAN Frame received
        if(xQueueReceive(CAN_Main_Rx_Queue, &Received_Frame, portMAX_DELAY) == pdPASS){
            Received_CAN_ID = Received_Frame.u16_CAN_ID;

            switch(Received_CAN_ID){
                case OBC_NACK_RECV:
                    xQueueSend(CAN_NACK_Queue, &Received_Frame, 0);
                    break;

                case OBC_TIMESTAMP_RECV:
                    xTaskNotify(RTC_Update_Handler,
                                Received_Frame.ul_DataLow,
                                eSetValueWithOverwrite);
                    break;

                case OBC_LPM_RECV:
                    start_wait_mode(60);    //Start wait mode for 1 minute.
                break;

                case OBC_ACK_RECV:
                    notify_value = HB_ACK_RECV;
                    xTaskNotify( QUIESCENT_CURRENTS_Get_Data_Handler, notify_value, eSetBits );
                    break;

                case OBC_HK_RESP:
                    xTaskNotify(CAN_ACK_Generate_Handler, (Received_Frame.ul_DataLow & 0xFF),
eSetValueWithOverwrite);
                    if( (Received_Frame.ul_DataLow & 0xFF) == OBC_HK_RADFET ){
                        xTaskNotifyGive(RADFET_Get_Data_Handler);
                    } else  if( (Received_Frame.ul_DataLow & 0xFF) == OBC_HK_PHOTO_1 ){
                        xTaskNotifyGive(PHOTODIODE_Get_Data_Handler[0]);
                    } else  if( (Received_Frame.ul_DataLow & 0xFF) == OBC_HK_PHOTO_2 ){
                        xTaskNotifyGive(PHOTODIODE_Get_Data_Handler[1]);
                    } else  if( (Received_Frame.ul_DataLow & 0xFF) == OBC_HK_PHOTO_3 ){
                        xTaskNotifyGive(PHOTODIODE_Get_Data_Handler[2]);
                    } else  if( (Received_Frame.ul_DataLow & 0xFF) == OBC_HK_PHOTO_4 ){
                        xTaskNotifyGive(PHOTODIODE_Get_Data_Handler[3]);
                    } else if( (Received_Frame.ul_DataLow & 0xFF) == OBC_HK_RAM ){
                        xTaskNotifyGive(RAM_Get_Data_Handler);
                    } else if( (Received_Frame.ul_DataLow & 0xFF) == OBC_HK_QS ){
                        notify_value = GENERATE_DATA;
                        xTaskNotify( QUIESCENT_CURRENTS_Get_Data_Handler, notify_value, eSetBits );
                    }
                    break;

                default:
                    break;
            }
        }
    }
}
```

- Tarea ‘CAN Msg Supervisor’ para la gestión de la retransmisión de mensajes.

```
/*Task to supervise CAN Msgs received and look for errors*/
void CAN_Msg_Supervisor_Task(void *parameters){
    CAN_Frame_t Sent_Frame;
    uint8_t Sent_Sequence_Id;

    CAN_Frame_t ReceivedACK_Frame;
    uint8_t ReceivedACK_Sequence_Id;

    CAN_Frame_t Frame_to_Send;
    //uint8_t acks_pending;

    uint8_t rtx_msgs;
    uint8_t Wrong_NACK_flag = 0;    //Start asuming there are no errors
    //CAN_Frame_t Aux_Frame;

    while (1)
    {
        Wrong_NACK_flag = 0;
        //Wait for notification
        if(xQueueReceive(CAN_NACK_Queue, &ReceivedACK_Frame, portMAX_DELAY) == pdPASS){
            //Get the NACK ID received
            ReceivedACK_Sequence_Id = getSequenceID(ReceivedACK_Frame);

            //Get number of messages pending to RTX
            rtx_msgs = uxQueueMessagesWaiting(CAN_RTX_Queue);

            if(rtx_msgs > 0){
                xQueuePeek(CAN_RTX_Queue, &Sent_Frame, 0);
                Sent_Sequence_Id = getSequenceID(Sent_Frame);

                //Check if the NACK received is lower than the first Frame to RTX
                if(Sent_Sequence_Id > ReceivedACK_Sequence_Id){
                    Wrong_NACK_flag = 1;
                    //ReceivedACK_Sequence_Id = Sent_Sequence_Id;
                    xQueueReset(CAN_RTX_Queue); //Reset CAN RTX Queue

                } else{
                    while (rtx_msgs > 0)
                    {
                        //Start Asuming the NACK is higher than the first Frame to RTx.
                        Wrong_NACK_flag = 1;

                        xQueueReceive(CAN_RTX_Queue, &Sent_Frame, portMAX_DELAY); //Antes estaba a 0
                        Sent_Sequence_Id = getSequenceID(Sent_Frame);

                        vTaskDelay(1); //Provisional para que funcione correctamente

                        if(Sent_Sequence_Id >= ReceivedACK_Sequence_Id){
                            xQueueSend(CAN_Main_Tx_Queue, &Sent_Frame, portMAX_DELAY);
                            Wrong_NACK_flag = 0;
                        }
                        rtx_msgs--;
                    }
                }
            }

            if(Wrong_NACK_flag == 1){
                Frame_to_Send.u16_CAN_ID = NACK_RESP_SENT;
                Frame_to_Send.ul_DataLow = RTC_get();   //Set timestamp

                if(uxQueueMessagesWaiting(CAN_Main_Tx_Queue) == 0){
                    Frame_to_Send.ul_DataHigh = (SENSORS_seq & 0xFF);
                }else{
```

```
                    xQueuePeek(CAN_Main_Tx_Queue, &Sent_Frame, portMAX_DELAY);
                    Frame_to_Send.ul_DataHigh = (getSequenceID(Sent_Frame) & 0xFF);
                }
                xQueueSendToFront(CAN_Main_Tx_Queue, &Frame_to_Send, portMAX_DELAY);
            }
        }
    }
}
```