Escuela de Ciencia y Tecnología

Proyecto Final Integrador

Ingeniería en las Telecomunicaciones

Desarrollo del firmware de una Computadora de a bordo de Payload y cálculo de un sistema de comunicación con tecnología LoRa para un nanosatélite

Docente: Marcelo Edgardo Romeo

Alumno: Federico Di Nardo

Tutores Carlos Canal - Leandro Gagliardi

Año: 2024

Índice

1. Abstrac

El objetivo principal de nuestro proyecto es llevar a cabo el análisis, diseño e implementación del firmware destinado a la Computadora de A Bordo de Payload (POBC) perteneciente al grupo de investigación LabOSat, y el presupuesto de enlace para un Subsistema de Comunicación de Payload (PSC).

Los nanosatélites están compuestos por diversos subsistemas, cada uno con funciones específicas y vitales. En este contexto, la computadora de a bordo de Payload asume un rol crítico al coordinar y supervisar las diversas tareas necesarias para cumplir con los objetivos de su misión. Su labor abarca desde el control de la Carga Útil hasta la gestión de la comunicación con la estación terrena.

La POBC estará equipada con un sistema operativo en tiempo real que gestionará las distintas tareas. Tendrá distintos modos de funcionamientos, entre los que se destacarán un modo de operación nominal, y un modo de actualización del firmware. Además, utilizará dos protocolos de comunicación principales: el Control Area Network (CAN) para interactuar con el Subsistema de Comunicación de Payload y el protocolo Ethernet para el control de la Carga Útil.

En cuanto al Subsistema de Comunicación de Payload, su responsabilidad radicará en establecer una comunicación efectiva entre la POBC y la Estación Terrena. Su objetivo incluye la transmisión de datos desde la POBC y la recepción de comandos desde la Estación Terrena para su posterior entrega a la POBC. Esta comunicación se llevará a cabo mediante la tecnología LoRa, una elección innovadora en la tecnología satelital.

Nuestra meta es desarrollar y validar un firmware para la POBC que cumpla con los estándares de robustez, confiabilidad y escalabilidad necesarios para garantizar nuestra misión. Este firmware será probado y validado en un kit de desarrollo, incluyendo pruebas de comunicación con una Estación Terrena simulada.

La estructura de este proyecto se divide en siete capítulos: una introducción, un capítulo dedicado a las técnicas y tecnologías empleadas, la descripción del proyecto en sí, un apartado destinado a las pruebas, las conclusiones finales, agradecimientos y, finalmente, la bibliografía utilizada. Cada uno de estos capítulos contribuirá a comprender y evaluar la implementación y el impacto de nuestro trabajo en el contexto satelital.

2. Introducción

En el siguiente capítulo se hará una introducción al proyecto, pasando por los objetivos del mismo, el ambiente en el cual fue desarrollado, los alcances que se establecieron y las limitaciones a las cuales nos hemos enfrentado, la tecnología a utilizar y los servicios que se puedan prestar.

El objetivo principal es el desarrollo integral del software para una computadora de a bordo de Payload para un nanosatélite, que se encargará de controlar y gestionar de manera eficiente su Carga Útil. También garantizará una comunicación fluida y confiable del mismo con una estación terrena. El desarrollo de este software implicará una serie de desafíos técnicos y científicos que abordaremos cuidadosamente para garantizar el correcto funcionamiento.

En primer lugar, trabajaremos en el diseño e implementación del firmware de la POBC para que realice el control de la Carga útil en forma altamente productiva y flexible. Esto implicará desarrollar algoritmos y protocolos avanzados que permitan la gestión óptima de los diferentes instrumentos y sensores que la componen. El software será capaz de recibir comandos desde la estación terrena y ejecutarlos de manera precisa y oportuna, garantizando la adquisición y procesamiento de datos.

En segundo lugar, pondremos énfasis en el diseño e implementación de un sistema de comunicación bidireccional entre la computadora de a bordo de Payload y la estación terrena. Esto implicará el uso de protocolos de comunicación robustos y sólidos, que permitirán la transmisión y recepción de datos de telemetría, comandos y otros tipos de información crítica para la misión.

Además de los aspectos técnicos, consideramos importante destacar que este proyecto también contribuirá al avance de la ciencia espacial y la tecnología satelital. La experiencia adquirida durante el desarrollo del software de la computadora de a bordo de Payload nos permitirá obtener conocimientos valiosos sobre el diseño y operación de nanosatélites, así como sobre los desafíos asociados a su implementación en misiones espaciales reales.

Llevaremos a cabo el proyecto en colaboración con el grupo de investigación LabOSat, perteneciente a la Escuela de Ciencia y Tecnología (ECyT) de la Universidad Nacional de San Martín (UNSAM). LaboSat cuenta con experiencia en el desarrollo y operación de nanosatélites, así como una destacada trayectoria en el campo de la investigación espacial.

La realización de este proyecto está dirigida a satisfacer parte de las necesidades y objetivos del grupo mencionado en lo que se refiere a la computadora de a bordo de Payload. El software que desarrollaremos será una herramienta fundamental para las misiones espaciales de LaboSat, permitiendo un control sobre la POBC y brindando una mejora significativa en las investigaciones realizadas.

Este grupo de investigación cuenta con instalaciones y recursos para el desarrollo y pruebas de nanosatélites. El equipo de investigadores y científicos altamente capacitados nos brindará el soporte necesario durante las etapas del proyecto, desde el diseño inicial hasta las pruebas necesarias para su validación. Esta colaboración

nos proporcionará un entorno propicio para la investigación, desarrollo e innovación, permitiendo la incorporación de los avances tecnológicos más recientes.

Además, el ambiente de la Escuela de Ciencia y Tecnología de la UNSAM nos ofrece un marco académico sólido y una comunidad de investigación dinámica, que fomenta la colaboración interdisciplinaria y el intercambio de conocimientos entre diferentes áreas de estudio. Esto nos proporcionará un contexto enriquecedor para el desarrollo de nuestro proyecto, permitiendo la exploración de nuevas ideas y la generación de soluciones innovadoras.

Trabajaremos sobre un kit de desarrollo propuesto por el grupo de estudio. Este kit será fundamental, ya que permitirá llevar a cabo cada una de las etapas de desarrollo y sus correspondientes pruebas.

Nuestro punto de partida es fue la necesidad de un firmware para la POBC. Comenzaremos evaluando diferentes códigos y configuraciones en el kit de desarrollo para comprender su funcionamiento y las capacidades que ofrecen.

Luego de esas pruebas iniciales podremos avanzar en la planificación y diseño del firmware. Elaboraremos diagramas en bloques que describan las diferentes funciones y procesos que la POBC deberá cumplir, considerando los requerimientos específicos del proyecto.

Una vez elaborados los diagramas en bloques, procederemos a programar el firmware de la POBC. Implementaremos distintos algoritmos, protocolos de comunicación y rutinas de control para garantizar el correcto funcionamiento de la Carga útil y establecer la comunicación con el sistema de comunicación y la Carga útil. Durante el proceso de desarrollo del firmware, realizaremos pruebas y verificaciones para asegurar su correcto funcionamiento y confiabilidad.

El resultado buscado es un firmware sólido y eficiente que permita controlar el funcionamiento de la Carga útil, establecer la comunicación con otros subsistemas y enviar la información generada a la estación terrena. El proceso de desarrollo del firmware involucrará un trabajo exhaustivo de planificación, diseño y programación en colaboración con el equipo de LabOSat y otros especialistas en la materia.

Al trabajar con un kit de desarrollo, se introducen ciertas restricciones que vamos a considerar a la hora de llevar a cabo el desarrollo del firmware.

Las limitaciones vinculadas al hardware son esenciales. Aunque el kit de desarrollo comparte el mismo chip que se empleará en las misiones satelitales, es crucial reconocer que el diseño y las características de la placa en un nanosatélite en funcionamiento pueden diferir notablemente. Estas diferencias surgen debido a que la placa en un nanosatélite real está especialmente diseñada y optimizada para operar en el espacio, teniendo en cuenta factores como la resistencia a la radiación, el consumo de energía y las limitaciones de espacio físico, entre muchos otros.

También, las limitaciones relacionadas con la interface de comunicación del kit de desarrollo son relevantes, ya que puede variar en comparación con la interface real que se emplea en el nanosatélite.

Asimismo, las restricciones originadas por el entorno de laboratorio deben ser

consideradas. Estas pueden diferir del entorno real de operación, lo cual puede influir en los resultados obtenidos en las pruebas.

La condiciones presupuestaria disponible para llevar a cabo las pruebas es un factor significativo que puede limitar la amplitud y la cantidad de las mismas.

Es muy relevante es el tiempo disponible para la realización de las pruebas, la cantidad de pruebas posibles y la precisión de los resultados obtenidos, los cuales también estarán sujetos a limitaciones.

Utilizaremos el lenguaje de programación C y un sistema operativo en tiempo real específico para el desarrollo del software. Seleccionaremos estas tecnologías cuidadosamente debido a su confibilidad y relevancia en la industria espacial.

El lenguaje de programación C es ampliamente utilizado en aplicaciones espaciales debido a sus características y beneficios para el desarrollo de software en el contexto de los satélites. Una de las ventajas clave de C es su capacidad para programar a bajo nivel, lo que significa que podremos acceder directamente al hardware del sistema. Esto es importante en sistemas donde buscamos maximizar la eficiencia y la gestión de recursos.

Al programar en C, tendremos un control óptimo sobre los recursos disponibles. Esto será fundamental para aprovechar al máximo el rendimiento del sistema y garantizar un uso eficiente de la energía y la capacidad de procesamiento. Al tener acceso directo al hardware, podremos ajustar y optimizar el funcionamiento para satisfacer los requerimientos.

Además de su capacidad para programar a bajo nivel, el lenguaje C cuenta con una amplia base de conocimientos y una gran cantidad de bibliotecas y herramientas desarrolladas específicas. Es decir, soluciones previamente probadas y optimizadas. Esto nos facilitará el desarrollo del software, ya que podremos aprovechar estas soluciones especializadas para tareas como la comunicación, el procesamiento de datos y la gestión de memoria.

El uso del lenguaje C en el desarrollo de software para nanosatélites también nos permitirá una mayor portabilidad y escalabilidad. El código escrito puede ser fácilmente transferido y adaptado a diferentes plataformas y sistemas operativos. Esto es importante en las computadoras de a bordo, donde los recursos pueden ser limitados y es necesario garantizar la compatibilidad con diferentes componentes.

Para la elección del sistema operativo en tiempo real nos basamos en varias consideraciones importantes. En primer lugar, un sistema operativo en tiempo real es especialmente diseñado para aplicaciones donde la precisión temporal y la respuesta determinista son críticas. En el caso de los nanosatélites, existen tareas y operaciones que deben ejecutarse de manera precisa y en momentos específicos, un sistema operativo en tiempo real nos proporcionará las herramientas necesarias para administrar y coordinar estas actividades. Esto lo lograremos a través de la planificación de tareas en función de prioridades y restricciones temporales, asegurando que las operaciones se realicen en el momento adecuado y con la máxima eficiencia.

Además, el sistema operativo en tiempo real nos ofrece servicios de comunicación

entre las diferentes partes del sistema, lo que facilitará la coordinación y la cooperación entre los diversos sistemas. Esto es especialmente importante cuando trabajaremos con componentes que deben interactuar de manera sincronizada.

También consideramos otros aspectos para la gestión efectiva de los recursos del sistema, como la memoria y el procesador. El sistema operativo en tiempo real nos permite una asignación óptima de estos recursos, asegurando que cada tarea reciba los recursos necesarios para su correcta ejecución y evitando conflictos o interferencias entre ellas.

El software que desarrollaremos desempeñará un papel fundamental en el control y funcionamiento organizado de la Carga útil, así como en la facilitación de su comunicación con la estación terrestre.

La POBC se encargará de llevar a cabo los chequeos internos necesarios para garantizar que todos los componentes y sistemas propios estén funcionando correctamente. Además, tendrá la responsabilidad de controlar su Carga útil. Esto implicará recibir los datos generados por la Carga útil y procesarlos de acuerdo con los algoritmos y protocolos establecidos. La POBC almacenará y transferirá estos datos hacia el PSC, para su posterior traspaso a Tierra. De esta manera, garantizaremos la recepción de datos desde la Carga Útil.

En cuanto a la comunicación, la POBC controlará directamente el PSC, facilitando la transferencia de datos desde la Carga Útil. La POBC actuará como un intermediario entre la Carga Útil y el PSC, garantizando una comunicación fluida y sin problemas.

El PSC desempeñará un papel crucial en la interconexión de la POBC, la Carga Útil y la estación terrestre. Este subsistema permitirá la transmisión bidireccional de datos. Será responsable de establecer y mantener una conexión confiable y segura, corroborando que los datos se transmitan de manera efectiva y que se mantenga una comunicación constante entre todas las partes involucradas. Esto lo lograremos mediante el uso de comandos y la recepción de datos, que nos permitirá una interacción efectiva y un monitoreo completo del funcionamiento del satélite.

Utilizaremos los comandos para enviar instrucciones y comandos específicos desde la estación terrena hacia la POBC y la Carga Útil, permitiendo el control y la configuración remota de diferentes funciones y parámetros. Esto es realmente importante porque nos permitirá que el nanosatélite sea flexible y adaptable a diferentes situaciones. Si ocurre algo inesperado o si cambian los requisitos de la misión, los operadores podrán enviadel payload. De esta manera, podremos garantizar que el payload se adapte a las circunstancias cambiantes y funcione de manera óptima. En nuestro caso, podremos dar distintas instrucciones tanto a la POBC, como a la Carga Útil. Por ejemplo, si quisiéramos que la Carga Útil sea activada en un momento dado, enviaremos la instrucción a la POBC y esta la recibirá a través de un comando.

Por otro lado, los datos transmisitos por el Subsistema de Comunicación de Payload desempeñará un papel igualmente importante al permitir recibir datos y mediciones en tiempo real desde el Payload hacia la estación terrena. A través de esto, podremos recibir datos que incluyen parámetros operativos vitales, como la tempe-

ratura, el consumo de energía y otros indicadores relevantes. La recepción de datos en tiempo real proporcionará una visibilidad completa del estado y el rendimiento de la Payload. Los datos recibidos permitirán realizar un monitoreo continuo y una evaluación precisa de la condición del satélite.

La capacidad de enviar comandos y recibir datos en tiempo real desde la estación terrena brindará a los operadores un control completo y una visión detallada del estado y el rendimiento de la Payload. Esto les permitirá tomar decisiones informadas y llevar a cabo acciones para garantizar el éxito de la misión. Además, la comunicación bidireccional entre la estación terrena y el Payload permitirá una supervisión constante y una gestión efectiva de los recursos y funciones del satélite.

El alcance de este proyecto incluirá el control integral de una Carga Útil. Nuestro objetivo es asegurar la supervisión y funcionamiento eficiente de esa Carga Útil, así como establecer una comunicación bidireccional entre la Carga Útil y la estación terrestre.

Para lograr esto, el software que desarrollaremos se encargará de monitorear constantemente el estado y rendimiento de la Carga Útil. Implementaremos algoritmos y protocolos para recopilar y procesar los datos generados por la Carga Útil, asegurando que se realicen las operaciones y cálculos necesarios para su correcto funcionamiento. Además, estableceremos mecanismos de control y supervisión.

En términos de comunicación, el software permitirá una conexión directa entre la Carga Útil y la Estación Terrena. Esto implicará el establecimiento de canales de comunicación confiables y seguros para la transmisión de datos generados por la Carga Útil hacia la estación terrestre y viceversa. Además, implementaremos mecanismos de control de calidad para garantizar la integridad de los datos transmitidos.

La interconexión desempeña un papel crucial en este proyecto, ya que garantizará la comunicación efectiva entre los diferentes componentes del sistema. En nuestro enfoque, estableceremos conexiones sólidas y precisas para habilitar una comunicación fluida y adecuada.

En primer lugar, nuestro software permitirá la comunicación entre la computadora de a bordo de Payload y la Carga Útil a través de una conexión Ethernet. Esta interface proporcionará una conexión de alta velocidad y confiable para transmitir datos y comandos entre la POBC y la Carga Útil. Esto permitirá un control y una supervisión constante de la Carga Útil, facilitando su funcionamiento. La capacidad de Ethernet para transmitir grandes volúmenes de datos a alta velocidad es especialmente relevante en aplicaciones satelitales, donde se generan y procesan cantidades significativas de información científica.

Además, implementaremos un protocolo robusto conocido como CAN-BUS (Controller Area Network) para facilitar la comunicación entre la POBC y el Subsistema de Comunicaciones. CAN es ampliamente utilizado en aplicaciones aeroespaciales debido a su capacidad para soportar comunicaciones confiables en entornos adversos. Este protocolo permitirá una transferencia eficiente de datos y comandos entre la POBC y el Subsistema de Comunicaciones, asegurando una coordinación efectiva y

un intercambio de información sin problemas.

El grupo de investigación LabOSat fue quien definió el uso de los protocolos CAN y Ethernet debido a sus características y capacidades específicas que se alineaban con los requisitos y necesidades del proyecto. El protocolo CAN proporciona una comunicación segura y funcional, mientras que Ethernet ofrece una alta velocidad de transmisión de datos y flexibilidad en la arquitectura de red.

Por último, adaptaremos el protocolo LoRa (Long Range) para la comunicación con la estación terrena. Dentro del ámbito satelital, la tecnología LoRa surgió como una solución adecuada y confiable para las comunicaciones de larga distancia. LoRa es un protocolo de comunicación de baja potencia y largo alcance diseñado para aplicaciones de Internet de las Cosas (IoT). Su capacidad para transmitir datos a distancias significativas con un consumo mínimo de energía lo convierte en una elección ideal para los nanosatélites, donde el ahorro de energía y la eficiencia son factores críticos.

El uso de LoRa en nanosatélites nos permitirá establecer una comunicación bidireccional estable y de largo alcance con la estación terrena. Utilizando técnicas avanzadas de modulación de espectro, LoRa puede transmitir datos a distancias de varios kilómetros, incluso en entornos con obstáculos y ruido de radiofrecuencia. Esta capacidad de alcance extendido es especialmente beneficiosa para las misiones espaciales, donde la distancia y las condiciones adversas pueden presentar desafíos significativos.

La organización de este trabajo se estructurará en siete capítulos: esta introducción, uno de técnicas y tecnologías, el proyecto en sí, el testeo, conclusiones finales, agradecimientos y bibliografía utilizada.

En este capítulo introductorio, se ha trazado un recorrido a través de una serie de temas fundamentales que constituyen el núcleo de este trabajo. Desde los objetivos que impulsan el proyecto hasta la exploración del entorno en el que se desenvuelve, se han abordado cuestiones esenciales para comprender la magnitud y el impacto de esta investigación. Las delimitaciones y alcances han sido identificados con el propósito de establecer un marco sólido para las futuras etapas del análisis. Además, se ha explorado la tecnología que subyace en este proyecto, así como los servicios que emergen de ella, detallando su relevancia en el contexto más amplio. La interconexión entre estos elementos ha sido destacada como un vínculo crítico para comprender la integralidad de este enfoque.

3. Técnicas y tecnología

En el siguiente capítulo haremos una introducción a los nanosatélies, se explicará cómo es su arquitectura interna, se analizarán las funciones de las computadoras de a bordo, los firmwares que utilizan, cómo es el manejo de datos entre los subsistemas, se explicarán los distintos sistemas de comunicación existentes y, por último, se analizará la tecnología Lora.

3.1. Introducción a los nanosatélites

Los nanosatélites son pequeños dispositivos de una masa de entre 1 y 10 kg. A pesar de su tamaño, se han convertido en una tecnología clave para una amplia gama de aplicaciones en la industria espacial. La creciente necesidad de información en tiempo real y la disminución de los costos de producción y lanzamiento han impulsado su gran popularidad.

La historia de los nanosatélites se remonta a los primeros días de la carrera espacial, cuando los primeros satélites artificiales fueron lanzados al espacio. A medida que la tecnología espacial avanzó, los satélites se hicieron cada vez más pequeños y eficientes, lo que llevó al desarrollo de los nanosatélites en la década de 1990. Desde entonces, han evolucionado rápidamente, pasando de ser experimentos académicos a desempeñar un papel fundamental en una amplia variedad de aplicaciones.

Operan principalmente en órbitas bajas terrestres (LEO, por sus siglas en inglés de Low Earth Orbit), lo que marca una de sus características distintivas. Esta elección orbital se traduce en una serie de ventajas que amplían significativamente sus capacidades y utilidad. Las órbitas bajas terrestres son trayectorias cercanas a la superficie de la Tierra, lo que les permite mantener una distancia relativamente pequeña entre el satélite y nuestro planeta. Esta proximidad brinda varios beneficios cruciales para su funcionamiento.

En primer lugar, las órbitas LEO posibilitan que los nanosatélites realicen misiones de corta duración de manera eficiente. Dada la cercanía a la Tierra, el tiempo que un nanosatélite pasa en órbita puede ser más breve que si estuviera en una órbita más alejada. Esto resulta especialmente útil para misiones específicas y rápidas, como la recopilación de datos en tiempo real.

Además, las órbitas LEO permiten un acceso rápido y económico al espacio. Debido a que el satélite no tiene que superar altitudes extremadamente altas, los lanzamientos suelen ser más económicos y, a menudo, más frecuentes. Esto agiliza la implementación de proyectos y la realización de investigaciones, ya que los nanosatélites pueden ser desplegados con mayor facilidad y a un costo reducido en comparación con los satélites más grandes y en órbitas más altas.

La distancia en las órbitas LEO puede variar, pero generalmente se sitúan a altitudes que van desde los 200 a los 2000 Kilómetros sobre la Tierra. El periodo de la órbita depende principalmente de la altitud, y varía en el rango de 90 a 120 minutos. Cómo la altitud de los nanosatélites en esta órbita es baja, su velocidad es muy alta

(mayor a 25.000 Km/h) y dan entre 12 y 16 vueltas terrestres por día. En consecuencia, el tiempo máximo durante el cual un nanosatélite está por encina del horizonte local para un observador en la Tierra es de hasta 20 minutos. Este tiempo se utiliza para transferir todos los datos que nos pueda enviar la Carga Útil.

Una de las principales razones detrás de la creciente popularidad de los nanosatélites es su capacidad para democratizar el acceso al espacio. Antes, solo grandes organizaciones gubernamentales o empresas multinacionales podían permitirse el lujo de lanzar satélites al espacio. Ahora, con los nanosatélites, universidades, pequeñas empresas e incluso países en desarrollo pueden participar en misiones espaciales a un costo mucho menor.

La continua evolución de la tecnología ha permitido que los nanosatélites sean cada vez más capaces y versátiles. Los avances en miniaturización de componentes, sistemas de energía eficientes y sistemas de propulsión innovadores han ampliado las posibilidades de lo que estos pequeños dispositivos pueden lograr.

Una de las principales ventajas es su mayor flexibilidad y adaptabilidad en términos de diseño y aplicación. Debido a su tamaño compacto y modularidad, pueden ser diseñados y configurados de manera más ágil y rápida para adaptarse a diversas misiones y objetivos. Su versatilidad permite que se utilicen en una amplia variedad de aplicaciones, desde la observación de la Tierra hasta la investigación científica y tecnológica.

Además, la naturaleza simplificada y estandarizada contribuye a una menor complejidad en su desarrollo y operación. Al utilizar componentes y sistemas comunes y probados, se reduce la necesidad de diseñar y construir sistemas personalizados desde cero. Esto no solo reduce los costos, sino que también acelera el proceso de desarrollo y lanzamiento.

Otra ventaja significativa es su menor costo en comparación con los satélites convencionales. El desarrollo y lanzamiento de un nanosatélite son considerablemente más accesibles, lo que abre oportunidades para una amplia gama de actores, incluidas universidades, empresas emergentes y países con recursos limitados. Esta democratización del acceso al espacio ha permitido una mayor participación en la exploración espacial y ha impulsado la innovación en la industria.

Además, la posibilidad de lanzar nanosatélites en órbitas más bajas representa una ventaja en términos de eficiencia y rendimiento. Al estar más cerca de la Tierra, experimentan un menor tiempo de tránsito y una menor latencia en las comunicaciones, lo que permite una transmisión de datos más rápida y confiable. Esto es especialmente crucial en aplicaciones que requieren información en tiempo real, como la observación de desastres naturales o el seguimiento de eventos climáticos.

Hay algunas limitaciones y desventajas que deben ser tomadas en cuenta a la hora de su diseño, implementación y planificación de misiones.

Una de las principales limitaciones es su menor vida útil en comparación con los satélites convencionales. Debido a su tamaño compacto y a la naturaleza de sus componentes, pueden tener una vida útil más corta, generalmente de unos pocos meses

a unos pocos años. Esto se debe en parte a la mayor vulnerabilidad a los efectos del ambiente espacial, como la radiación cósmica y el deterioro térmico. Además, la falta de sistemas de mantenimiento y reparación en órbita también puede acortar la vida útil de estos dispositivos. Al ser de un tamaño tan reducido, no suelen contar con unidades de respaldo. Es decir, el daño de una de sus plaquetas, podría significar la pérdida total del satélite.

Otra limitación importante es la menor capacidad de carga. Dado su tamaño reducido, tienen menos espacio para alojar instrumentos y cargas útiles en comparación con los satélites más grandes. Esto puede restringir el tipo de experimentos científicos que se pueden llevar a cabo y limitar el alcance de las misiones. Sin embargo, los avances en miniaturización y la mejora de la eficiencia de los componentes han permitido que los nanosatélites alberguen una variedad creciente de cargas útiles, lo que ha ampliado su capacidad para llevar a cabo investigaciones y experimentos científicos. Además, pueden enfrentar una menor capacidad de procesamiento de datos debido a sus limitaciones de energía y hardware. Con recursos limitados, como la capacidad de procesamiento de la CPU y la memoria, pueden tener dificultades para manejar grandes volúmenes de datos o realizar cálculos complejos en tiempo real. Esto puede ser un desafío en aplicaciones que requieren un procesamiento intensivo, como el procesamiento de imágenes o el análisis de datos científicos.

A pesar de estas limitaciones, es importante destacar que los nanosatélites han demostrado ser una herramienta valiosa y efectiva en una amplia variedad de misiones espaciales. Su bajo costo y rápida implementación los convierten en una opción atractiva para proyectos de investigación y exploración espacial. A medida que la tecnología continúa evolucionando y mejorando, es probable que muchas de estas limitaciones se superen con el tiempo, lo que abrirá nuevas oportunidades para la utilización de nanosatélites en aplicaciones más complejas y ambiciosas. Al comprender y abordar estas limitaciones de manera efectiva, podemos maximizar el potencial y aprovechar al máximo su capacidad.

Su uso se ha extendido en una amplia gama de aplicaciones, destacando su versatilidad y adaptabilidad en diversas áreas de la industria espacial y más allá. Estos pequeños dispositivos han demostrado ser una solución valiosa para enfrentar desafíos tanto en la Tierra como en el espacio, y su creciente popularidad ha sido impulsada por sus numerosas ventajas y beneficios.

Una de las aplicaciones más prominentes de los nanosatélites es la observación de la Tierra. Equipados con cámaras y sensores de última generación, estos satélites en miniatura pueden recopilar datos y capturar imágenes detalladas de la superficie terrestre. Esto es fundamental para el monitoreo ambiental, el seguimiento de cambios climáticos, el control de la deforestación, el monitoreo agrícola y la planificación urbana. Su capacidad para orbitar la Tierra rápidamente y en órbitas más bajas permite una frecuencia de observación más alta y una actualización continua de la información.

Otra aplicación significativa es su contribución en las comunicaciones. Estos pe-

queños dispositivos han demostrado ser eficientes en la implementación de sistemas de telecomunicaciones y conectividad de Internet en áreas remotas y desatendidas. Proporcionan servicios de telecomunicaciones vitales para comunidades rurales y regiones aisladas, superando las barreras geográficas y mejorando la conectividad global. Además, la capacidad de desplegar constelaciones de nanosatélites permite la cobertura global y la entrega de servicios de comunicación de manera más efectiva.

La ciencia y la exploración del espacio también han visto un impulso significativo gracias a ellos. Utilizados como herramientas de investigación, estos dispositivos compactos permiten realizar estudios sobre la atmósfera terrestre, el clima espacial, la física de partículas y la astronomía. Su capacidad para operar en formaciones y trabajar en conjunto con otros satélites más grandes ofrece oportunidades únicas para realizar investigaciones de mayor escala y alcance.

El ámbito educativo también ha encontrado beneficios en su uso. Estos dispositivos ofrecen una plataforma educativa única para estudiantes e ingenieros en formación, brindándoles una experiencia práctica invaluable en la construcción, operación y gestión de misiones espaciales. Además, el desarrollo de nanosatélites permite la formación de equipos multidisciplinarios y fomenta el espíritu de innovación y colaboración en el campo de la ingeniería espacial.

3.2. Arquitectura de los nanosatélites

Los satélites están compuestos de dos partes principales: la Carga Útil y la plataforma de vuelo. La Carga Útil, está definida por la misión del satélite, como por ejemplo la observación de la tierra o la recolección de datos. Esta Carga Útil no podría funcionar en el entorno espacial sin la plataforma de vuelo. La plataforma de vuelo se compone de varios subsistemas, en donde cada uno de ellos debe cumplir un rol para poder llevar a cabo la misión satelital. La Carga Útil debe estar dentro de un subsistema estructural para, por un lado, ser utilizado como interface con el lanzamiento del vehículo y, por otro lado, proteger al satélite de los efectos del entorno espacial tras su puesta en órbita. Esto es responsabilidad del Subsistema Estructural. Los diversos equipos electrónicos a bordo necesitan ser alimentados por la energía eléctrica. Por lo tanto, los nanosatélites necesitan un subsistema de energía tal que sea responsable de la generación, el almacenamiento y la distribución a los distintos subsistemas. La temperatura en el ambiente del espacio puede variar, por lo que necesitan de una subsistema térmico de control que asegure que la temperatura de cada equipo a bordo se mantenga dentro de un intervalo de temperatura de funcionamiento correcto. El nanosatélite debe tener cierto control sobre su actitud y orientación, por lo cual también va a necesitar de un subsistema propicio para ello, lo que se traduce en un Subsistema de Control. Estos diversos subsistemas satelitales, incluidos la Carga Útil, deben controlarse de forma remota, por lo que va a ser necesario un PSC.

Todos estos requerimientos hacen que la complejidad intrínseca de los nanosatélites se revele en su diseño y funcionamiento coordinado. Los subsistemas satelitales, en su conjunto, permiten que estos pequeños artefactos espaciales alcancen sus objetivos

de misión.

En particular, la interconexión de estos subsistemas es esencial para garantizar un rendimiento óptimo del nanosatélite en el entorno espacial. La comunicación, el control de orientación, la gestión térmica y la provisión de energía son elementos cruciales que deben trabajar en armonía para lograr una operación exitosa.

En resumen, la arquitectura de los nanosatélites se encuentra compuesta por los siguientes subsistemas:

- Subsistema de Energía (PS - 'Power Subsystem'): Es responsable de proporcionar energía eléctrica al satélite. Puede incluir paneles solares, baterías, reguladores de tensión y convertidores de energía.
- Subsistema de Comunicaciones (TTRS - 'Telemetry, telecommand and Ranging Subsystem'): Permite al satélite tener una comunicación bidireccional con la estación terrena.. Puede incluir antenas, transmisores, receptores, moduladores y demoduladores.
- Subsistema de Control de Actitud (AOCS - 'Attitude Orbit and Control Subsystem'): Es responsable de controlar la orientación del satélite y su posición en el espacio. Puede incluir sensores, actuadores, sistemas de control de reacción y giroscopios.
- Subsistema de Carga Útil (PAY - 'Payload'): Este subsistema es responsable de llevar a cabo la misión científica del satélite. Puede incluir instrumentos de medición, cámaras, sensores remotos y otros equipos científicos.
- Subsistema de Estructura (SS - 'Structural Subsystem'): Proporciona una estructura mecánica para el satélite y lo protege durante el lanzamiento y la operación en el espacio. Puede incluir paneles solares, paneles térmicos, marcos y cubiertas protectoras.
- Subsistema de Manejo de Datos (C&DHS - 'Command and Data Handling Subsystem'): Es quien se encarga de la comunicación entre los distintos subsistemas.
- Subsistema Térmico (TCS - 'Thermal Control Subsystem'): Es el encargado de mantener las temperaturas de todos los equipos del satélite en un rango determinado.
- Subsistema de Computadora de A Bordo (OBCS - 'On Board Computer Subsysem'): Algunos autores suelen considerar la OBCS como un subsistema satelital, aunque no es llamada así mismo en todos los casos. Le proporciona la inteligencia al satélite y se encarga de controlar al resto de los subsistemas.

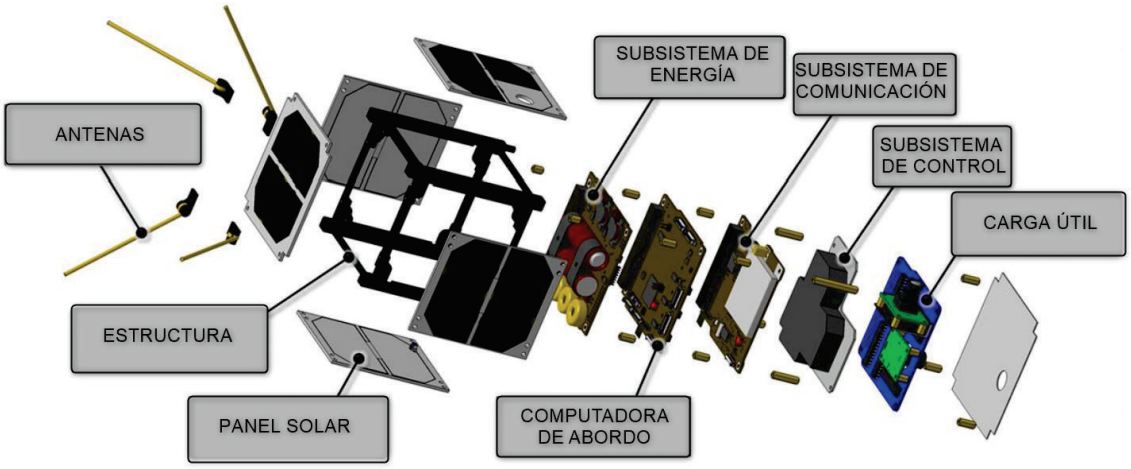


Figura 1: Subsistemas satelitales

Cada uno de estos subsistemas es crítico para el éxito de la misión del satélite y debe ser diseñado cuidadosamente para garantizar la funcionalidad y la confiabilidad del satélite en el espacio. Para aumentar su robustez y asegurar una operación continua y segura, muchos proyectos de nanosatélites implementan estrategias de redundancia en subsistemas vitales.

La redundancia implica la inclusión de componentes duplicados o sistemas de respaldo que pueden entrar en acción en caso de que el subsistema principal experimente una falla. Esta medida de seguridad adicional es especialmente crucial en el entorno hostil del espacio, donde los nanosatélites están expuestos a radiación, fluctuaciones de temperatura y otros desafíos extremos. Al tener subsistemas redundantes, se reduce la posibilidad de que una sola falla pueda poner en peligro toda la misión. Sin embargo, esto no siempre es posible, ya que un subsistema extra aumenta el peso, volumen y costo del satélite, y no siempre es factible llevar a cabo su integración.

Tomando en cuenta la miniaturización de los satélites, existe otra nomenclatura para los distintos subsistemas que los componen. En la figura número 2 se puede observar cómo es que se encuentran compuestos:

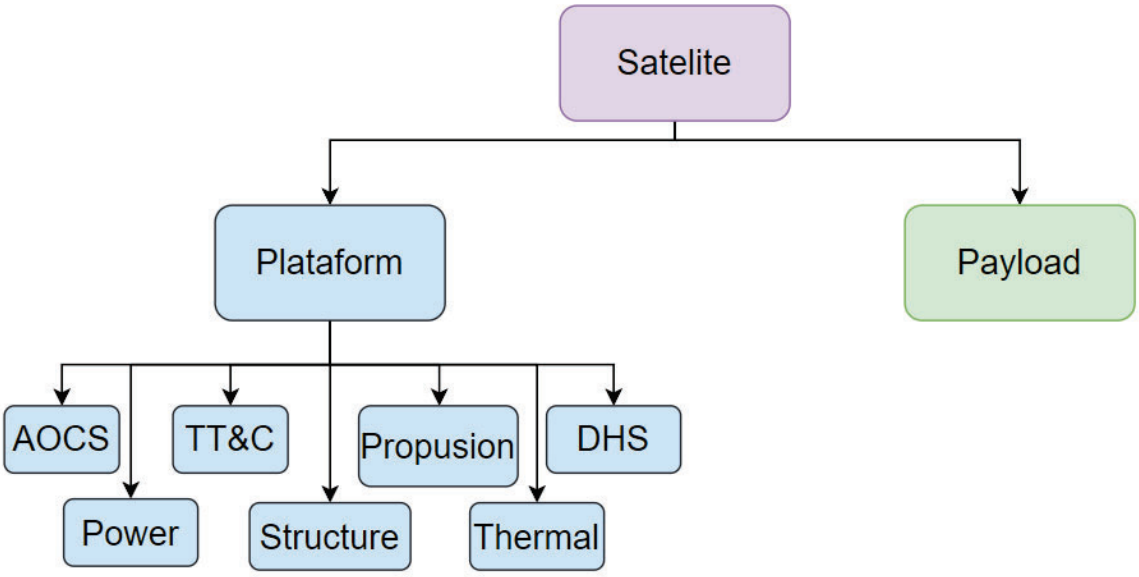


Figura 2: Arquitectura de los Subsistemas sateliteales

El DHS es considerado una interfase entra la estación de control y la del resto de los subsistemas. Se encarga de comunicar a la OBC con el resto de los subsistemas. Para poder realizar esa comunicación, existen dos estructuras principales:

- Arquitectura estrella: La OBC tiene una línea de comunicación por cada uno de los subsistemas. En esta arquitectura, la OBC debe tener al menos una interface por cada uno de los subsistemas, y puede que presente problemas en cuanto al cableado.
- Arquitectura BUS: En este caso, todos los subsistemas están conectados a un BUS de comunicación, similar a una Local Area Network (LAN) a bordo del satélite. En general, se usa un protocolo del tipo Maestro-Esclavo. Los tres protocolos más comunes usados en los sistemas embebidos son: CAN (Controller Are Network), SPI (Serial Peripheral Interface) y el IIC (Inter Integrated Circuit).

En cuanto al armado y desarrollo, se suele dividir en varias etapas. El tiempo estimado va a depender de la complejidad y el tipo de misión que se le quiera designar al nanosatélite. Pero como el proceso de fabricación suele ser similar en la mayoría de dispositivos, se suele estimar un cronograma, el cual toma entre 18 y 24 meses desde el desarrollo hasta su finalización, tal como podemos observar en la imagen número 5

1. Concepto de desarrollo (1 a 3 meses): Define los objetivos primarios y los detalles básicos del proyecto.

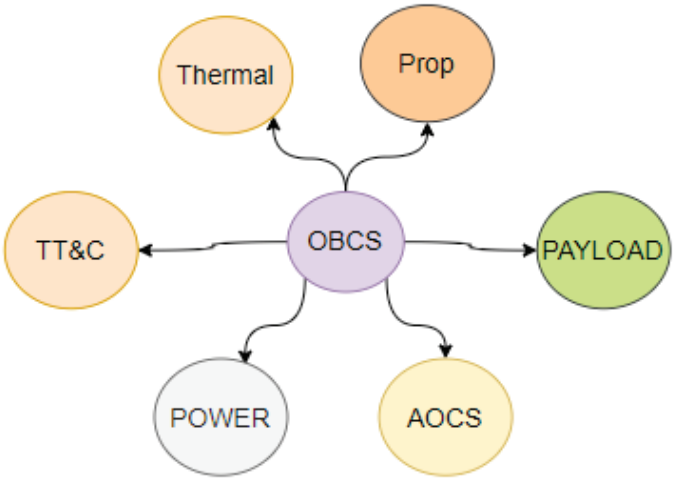


Figura 3: Arquitectura Estrella

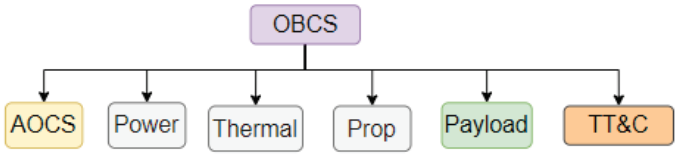


Figura 4: Arquitectura BUS

2. Financiación (1 a 12 meses): Simultáneamente con el paso 1, la financiación es una parte crítica del proyecto. Se debe tener en cuanta los costos de lanzamiento, los cuales pueden llegar a ser más caros que el armado del nanosatélite.

3. Revisiones de mérito y viabilidad (1 a 2 meses): Se evalúa la probabilidad de éxito del proyecto teniendo en cuenta la viabilidad técnica.

4. Diseño (1 a 6 meses): Despiece de los módulos básicos que van a componer al satélite.

5. Desarrollo (6 a 10 meses): Es en donde se desarrolla cada uno de los subsistemas que van a componente al satélite.

6. Integración (1 a 2 meses): Una vez que se han diseñado los diferentes subsistemas, se lleva a cabo la integración de los componentes en la estructura del satélite. Esto implica la conexión de los diferentes componentes y la realización de pruebas para garantizar que el satélite funcione correctamente.

7. Ensamblado (1 a 2 meses): Después de la integración, se lleva a cabo el ensamblado del satélite, que implica la conexión de todos los componentes y la

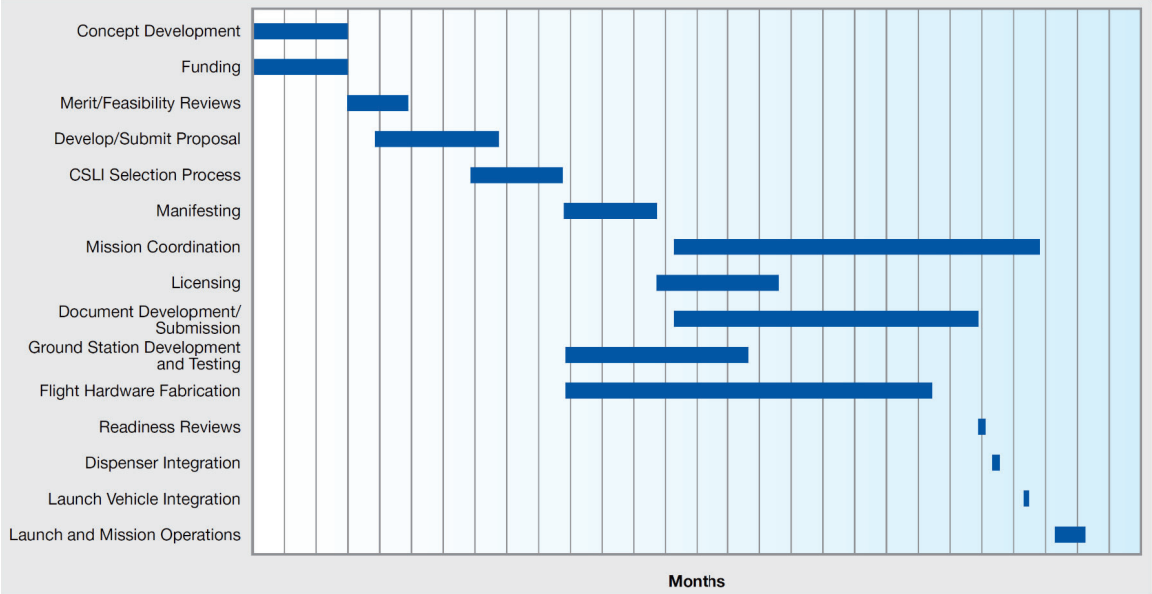


Figura 5: Línea temporal estimativa para proyectos satelitales

realización de pruebas adicionales para verificar que el satélite esté listo para su lanzamiento.

8. Lanzamiento: Finalmente, el nanosatélite es lanzado al espacio a bordo de un cohete. Una vez en órbita, el satélite debe activarse y configurarse para comenzar a realizar su misión científica.

El desarrollo de un nanosatélite implica varios desafíos, como el tamaño reducido del satélite, la necesidad de minimizar el consumo de energía y la limitación en la cantidad de datos que pueden transmitirse desde el satélite. Sin embargo, los avances en la tecnología y en los métodos de diseño han permitido la construcción de nanosatélites cada vez más complejos y con mayores capacidades.

3.3. Computadora de A Bordo de Payload (POBC)

El subsistema de computadora de a bordo de Payload (POBC) se considera el cerebro del satélite. Es el principal responsable de coordinar y realizar las funciones básicas requeridas para el funcionamiento del Payload, así como de su control desde tierra. Además, las responsabilidades del subsistema incluyen la integración de los diferentes componentes, el cumplimiento del programa operativo del satélite, la corrección de errores y fallas, gestionar el almacenamiento de datos, manejar ciertas tareas de comunicación, monitorear e informar sobre el estado de ciertos subsistemas, implementar leyes de control y accionar aspectos de la configuración de aviónica del sistema.

Debido a la amplia gama de estas tareas, junto con el requisito de que una POBC esté esencialmente siempre activa (o al menos, siempre accesible), es muy importante que se incorpore una programación y un equilibrio de carga eficaces.

Como podemos observar en la imagen número 6, una computadora de a bordo de Payload generalmente se divide en 3 partes principales distintas

- La electrónica: proporciona protección contra los efectos de la radiación, las fuerzas magnéticas y las interferencias de ruido.
- Interfaces: recopila la energía suministrada por el sistema de energía eléctrica (EPS) y gestiona la comunicación con otros módulos a bordo.
- El microcontrolador: la unidad de procesamiento dedicada a la gestión de satélites. Gestiona datos de otros sistemas (recopilados a través de la interface) y da comandos. También almacena mediciones, registros y otros datos del sistema

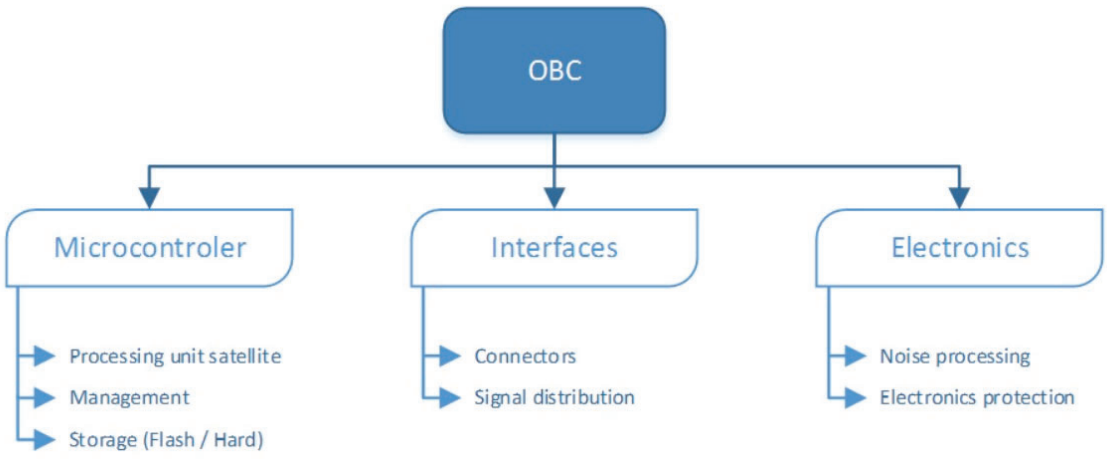


Figura 6: Arquitectura de una POBC

Utilizan microcontroladores a través de los cuales puede gestionar los datos del satélite. Un microcontrolador es esencialmente una CPU que tiene un tamaño y un consumo de energía mucho más pequeños para poder colocarse en un nanosatélite.

Para seleccionar el mejor microcontrolador, se deben tener en cuenta muchos aspectos como el consumo de energía, la temperatura, la tensión de funcionamiento, las interfaces y la compatibilidad del bus serie para evitar algunos problemas. Los microcontroladores están disponibles de diferentes fabricantes en variantes que admiten longitudes de palabra de 8, 16 y 32 bits. Las arquitecturas de 8 y 16 bits fueron las preferidas en la tecnología CubeSat porque muchas de las aplicaciones integradas y en tiempo real en ese momento no dependían críticamente de la memoria, la potencia o la velocidad y la cantidad de datos a manejar era suficiente.

Algunos de los criterios claves para la selección de una POBC son:

- Capacidad de procesamiento: la unidad informática debe poder manejar la capacidad de procesamiento necesaria para operar la Carga Útil y los subsistemas que la soportan (por ejemplo, control de altitud, comunicación, distribución de energía, etc.).
- Memoria (almacenamiento y RAM): tanto la capacidad como el formato de la memoria deben elegirse para satisfacer sus necesidades. Un POBC normalmente incluye memorias volátiles y no volátiles con diferentes capacidades.
- Interoperabilidad y capacidades de interface: como unidad de control central, es vital que el POBC pueda funcionar de manera efectiva con las interfaces requeridas (por ejemplo, USB, I2C, etc.) y tenga suficiente capacidad y puertos para los subsistemas externos a los que se conectará.
- Confiabilidad del software: la POBC necesita tener un software sólido ejecutándose para poder manejar la secuenciación de eventos, monitorear el estado y el rendimiento de todos los sistemas y manejar cualquier problema en órbita.
- Requisitos de energía: aunque los requisitos de energía de POBC son generalmente bajos en comparación con otros subsistemas, es importante tener en cuenta qué energía se necesita para sus cálculos generales.
- Tamaño y peso : el sistema que elija debe ajustarse a su presupuesto actual de masa y volumen para evitar re diseños más extensos.

Con el tiempo, más productos y aplicaciones empezaron a requerir una mayor capacidad de procesamiento. Por lo que fue necesaria una migración de una arquitectura central de 8 y 16 bits a una de 32 bits. Es por esto que de momento existen más POBC de CubeSat desarrollados con un microcontrolador de 16 bits, aunque se prevé en el futuro una migración a arquitecturas de 32 bits.

La memoria interna (también denominada memoria en chip) del microcontrolador está dividida en dos tipos:

- Memoria Flash: Es una variación de la memoria de sólo lectura programable eléctricamente borrable (EEPROM) que es accesible en bloques de datos. Es la parte no volátil de la memoria, lo que significa que conserva su valor incluso cuando se quita la alimentación en el sistema. Tiene la ventaja de presentar un tiempo de acceso rápido y tiene una alta tolerancia a los efectos de la radiación.
- Memoria estática de acceso aleatorio (SRAM): Es la parte volátil de la memoria, lo que significa que pierde datos cuando se corta la alimentación. Las operaciones de lectura y escritura en SRAM son muy rápidas; por lo tanto, se utiliza para almacenar datos de programa que se asignan temporalmente y cambian constantemente durante la ejecución del programa.

La memoria en chip suele ser insuficiente y, en la mayoría de los casos, los microcontroladores ofrecen memoria externa para ampliar la capacidad de memoria del sistema. En un subsistema POBC, el microcontrolador seleccionado debe permitir la ampliación de la memoria externa, ya que los datos de mantenimiento y de Carga Útil suelen ser de mayor tamaño que el espacio disponible en el chip. El módulo de memoria externa también puede utilizarse para aplicaciones de programa que se ejecutarían en el microcontrolador y que podrían requerir espacio de memoria adicional. Existen varios componentes disponibles que pueden utilizarse como módulos de expansión de memoria, como las tarjetas digital/multimedia (SD/MMC) y circuitos integrados de memoria (IC).

Los dispositivos de I/O (o periféricos) están integrados en el microcontrolador y son utilizados por el procesador para comunicarse con otros dispositivos externos. En el caso de los nanosatélites, estos periféricos crean una interface entre el POBC y los demás subsistemas. La arquitectura del satélite requiere el uso de un protocolo de bus de datos en serie estándar para transferencias de datos rápidas entre todos los subsistemas. Esta transferencia de datos se inicializa y monitoriza por el microcontrolador del POBC. Las interfaces serie más comunes que se encuentran en microcontroladores incluyen la recepción-transmisión asíncrona universal (UART), el circuito integrado (I2C) y la interface periférica en serie (SPI).

Para que el POBC funcione con eficacia, se necesitan componentes de hardware de apoyo adicionales al microcontrolador. Es recomendable que el subsistema POBC incluya una protección contra sobre corriente, un reloj de tiempo real (RTC) respaldado por batería y uno o más sensores de temperatura a bordo. La protección contra sobre corriente es necesaria para controlar la corriente consumida por el subsistema y desconectar la alimentación en caso de sobre corriente. El RTC controla la hora y ayuda a sincronizar las operaciones a bordo y también mantiene sincronizados los datos de a bordo con las operaciones en tierra. La hora suele codificarse en un formato de temporización conocido como contador de tiempo Unix (UTC8). La alimentación del RTC suele estar respaldada por una batería, de modo que el tiempo no se pierde cuando se desconecta la alimentación del POBC por cualquier motivo. Los sensores de temperatura a bordo son necesarios para controlar las variaciones de temperatura en diferentes puntos del nanosatélite expuestos al duro entorno espacial.

3.4. Firmware y Sistemas Operativos en nanosatélites

El firmware se refiere al conjunto de instrucciones de software que se programa en la POBC. Es esencial para el correcto funcionamiento del satélite, ya que se encarga de controlar todos los subsistemas, como la Carga Útil, los sistemas de comunicación, los sensores y la electrónica a bordo.

Es como un conjunto de reglas y procedimientos que guían al satélite en su operación. Por ejemplo, cuando el satélite está en órbita, el firmware puede estar programado para controlar el sistema de propulsión para ajustar su posición y velocidad. También puede gestionar la orientación del satélite para apuntar sus sensores o Carga Útil hacia una región específica de la Tierra.

Además, el firmware permite gestionar la comunicación con la estación terrena y recibir comandos desde ella. De esta manera, los operadores en tierra pueden enviar instrucciones y actualizaciones al satélite, y este se encarga de interpretar y ejecutar esas ordenes.

Permite generar respuestas automáticas ante ciertas situaciones, como la detección de eventos o anomalías. Por ejemplo, si el satélite experimenta un aumento inesperado de temperatura, el firmware puede estar programado para activar ciertos mecanismos de enfriamiento o poner en marcha procedimientos de seguridad.

El objetivo del firmware de la POBC es satisfacer los requisitos y las funcionalidades que la misma necesite. Las posibilidades para eso son muchas, por lo que es importante identificar qué separa un buen diseño de uno defectuoso. McConnel, quien es el autor de 'Code Complete', afirma que hay varias características generales para un diseño de alta calidad. Algunas de estas son:

- Mínima complejidad
- Facilidad de mantenimiento
- Expansibilidad (que se pueda mejorar, cambiando mínimas partes)
- Reutilización (significa diseñar el sistema de modo que se puedan reutilizar partes del mismo en otros sistemas).

La complejidad mínima y la facilidad de mantenimiento son las principales características de un buen diseño. Sumado a una baja complejidad para integrar distintos módulos, lo cual minimiza el tiempo de trabajo, testeo y mantenimiento. Hay muchos caminos diferentes para hacer un sistema embebido. Los programas integrados y de tiempo real tiene que controlar e interactuar con distintos dispositivos, y cuentan con ciertas ventajas y funcionalidades a la hora de tener que desarrollar un sistema.

3.4.1. RTOS

Los softwares que se utilizan en misiones satelitales suelen ser sistemas operativos de tiempo real, ya que estos permiten la programación de tareas y eventos en tiempo

real y pueden ser configurados para adaptarse a los requisitos específicos del satélite y de la misión que se lleve a cabo.

Algunos sistemas estrictos en tiempo real todavía se programan a veces en lenguaje Assembler para poder cumplir con plazos ajustados. Sin embargo, también se utilizan ampliamente lenguajes a nivel de sistemas, como C, que permiten generar código eficiente. La ventaja de utilizar un lenguaje de programación de sistemas como C es que permite el desarrollo de programas muy eficientes, que permiten el acceso directo al hardware. Como no existen restricciones de tiempo real que exijan la implementación en Assembler, el lenguaje C se considera el más apropiado para el software.

Algunas de las características de los sistemas operativos son:

- Puntualidad en las tareas ejecutadas.
- Garantizar el procesamiento de datos dentro de un tiempo límite.
- Que las funciones complejas se completen en un tiempo determinado.

El objetivo principal de un sistema operativo de tiempo real es garantizar la funcionalidad de ejecutar múltiples tareas al mismo tiempo, lo que obviamente no es posible, ya que la programación de las tareas es secuencial, y un procesador sólo puede ejecutar una tarea por cada núcleo / hilo disponible. Una tarea es un fragmento de código que el programador del sistema operativo puede programar y dedicar a una funcionalidad específica. Las tareas pueden tener diferentes prioridades para ejecutarse en un orden predecible.

Por lo que, un RTOS lo que hace es repartir el tiempo del procesador entre cada una de las tareas que lo demanden. Mientras que para un observador, las tareas van a parecer estar ejecutándose en paralelo, lo que realmente hace el microprocesador es repartir el tiempo entre cada una de ellas. Justamente, están diseñados para gestionar diferentes tareas y para garantizar que todas las operaciones críticas se realicen a tiempo y dentro de un tiempo determinado. La ejecución de las tareas la podemos observar en las figuras 7 y 8, en donde se demuestra la ejecución de las mismas según la perspectiva de las personas, y la ejecución real según el sistema operativo.

El núcleo de un RTOS es un algoritmo avanzado para la programación, cuyos factores claves son una latencia mínima de interrupción y una latencia mínima de conmutación de subprocesos. Uno de los sistemas operativos de tiempo real más utilizados en nanosatélites es FreeRTOS (Free Real-Time Operating System), el cual es una plataforma de software de código abierto que proporciona una base para la programación de tareas en tiempo real y la gestión de recursos de hardware. FreeRTOS se utiliza comúnmente en nanosatélites debido a su bajo consumo de recursos y su capacidad para soportar múltiples procesadores y sistemas de comunicación. Una de las características claves es su capacidad para permitir la programación concurrente y la gestión de prioridades de tareas. Esto significa que el sistema operativo puede ejecutar varias tareas simultáneamente (siempre y cuando contenga más de un núcleo), y asegurarse de que las tareas críticas se ejecuten con prioridad. Esto es

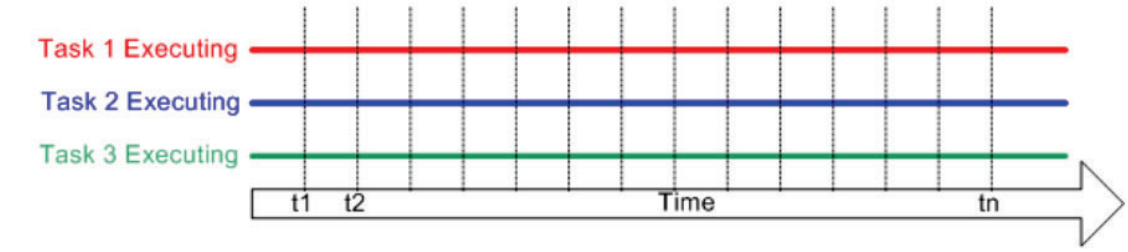


Figura 7: Ejecución de las tareas según percepción de las personas

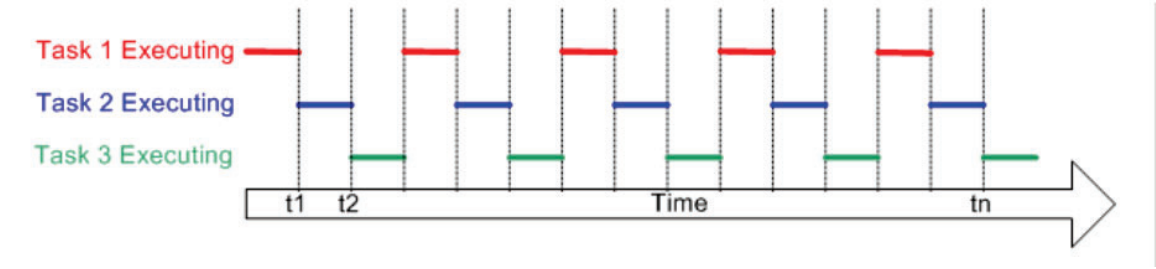


Figura 8: Ejecución real de las tareas en un RTOS

especialmente importante en los nanosatélites, donde la coordinación precisa de los distintos subsistemas es esencial para el correcto funcionamiento del satélite.

Otra característica importante es su capacidad para gestionar la memoria de forma eficiente. Los nanosatélites suelen tener recursos de memoria limitados, por lo que es importante que el sistema operativo utilice la memoria de forma eficiente para asegurar que haya suficiente memoria disponible para las tareas críticas.

Algunos de los firmware más comunes utilizados en las POBC son:

- FreeRTOS: Como ya mencionamos, es un sistema operativo en tiempo real que se utiliza comúnmente en satélites para controlar la POBC y coordinar la ejecución de tareas.
- RTEMS (Real-Time Executive for Multiprocessor Systems): es otro sistema operativo en tiempo real que se utiliza en satélites para controlar la POBC. Es un software libre y de código abierto, que ha sido utilizado en varios proyectos espaciales.
- VxWorks: es un sistema operativo en tiempo real comercial desarrollado por Wind River Systems y se utiliza en muchas aplicaciones en tiempo real, incluyendo satélites.

- Linux: Aunque no es común, también es posible utilizar sistemas operativos de propósito general como Linux en las POBC. Esto puede ser especialmente útil si se requiere una gran cantidad de capacidad de procesamiento para la Carga Útil del satélite.

Cada uno de los sistemas operativos en tiempo real tienen sus propias ventajas y desventajas, dependiendo de las necesidades específicas del proyecto. A continuación, se presentan algunas ventajas y desventajas de FreeRTOS en comparación con RTEMS, VxWorks y Linux:

Ventajas de FreeRTOS:

1. Es un sistema operativo de tiempo real muy popular y ampliamente utilizado en proyectos de sistemas embebidos.
2. Es de código abierto y gratuito.
3. Es altamente configurable y programable para satisfacer las necesidades del proyecto.
4. Soporta una amplia variedad de arquitecturas de procesadores.
5. Tiene una comunidad de usuarios activa y un amplio conjunto de herramientas y recursos disponibles en línea.

La elección de un sistema operativo en tiempo real depende de las necesidades específicas del proyecto. FreeRTOS es un sistema operativo de tiempo real popular y ampliamente utilizado, pero puede tener limitaciones en términos de seguridad y certificación. RTEMS es altamente configurable y ha sido utilizado en proyectos críticos y certificados, pero puede tener una curva de aprendizaje empinada. VxWorks es conocido por su alta confiabilidad y seguridad, pero puede tener costos asociados y recursos limitados para usuarios que no cuentan con soporte de Wind River.

3.4.2. FreeRTOS

FreeRTOS está disponible como una biblioteca de tipos y funciones para crear aplicaciones de software integradas, multitarea y en tiempo real. La programación puede ser preventiva, cooperativa o una configuración híbrida. Está escrito principalmente en C, con algunas partes en Assembler.

El espacio necesario en ROM para el núcleo en sí es entre 5 y 10 KByte, mientras que necesaria entre unos 11 Kbytes de Flash y 64 Kbytes de RAM para funcionar. Por lo cual se puede concluir en que FreeRTOS no tendrá ningún problema para caber dentro de ninguno de los microcontroladores ya que sólo debería usar una pequeña parte de la memoria del sistema. Dentro del “Core” mismo del funcionamiento de FreeRTOS encontramos el concepto de “Tarea”. De forma resumida, una tarea es

un envoltorio en torno a una función, que pasamos a FreeRTOS para que la gestione. Íntimamente relacionado tenemos el concepto de 'Scheduler' que es la parte de FreeRTOS que se encarga de gestionar y ejecutar las distintas tareas que tengamos definidas. Para ello cada tarea tiene un estado que puede ser disponible, en ejecución, suspendida o bloqueada. El cambio de estados lo realiza el 'Scheduler', como podemos observar en la imagen número 9.

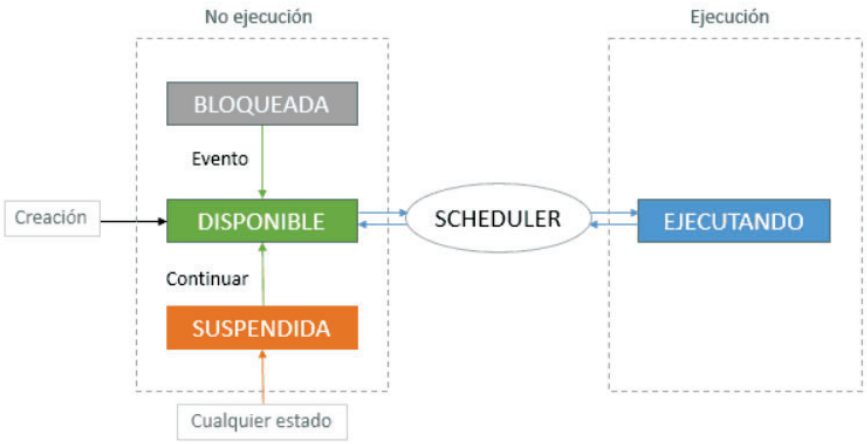


Figura 9: Estado de una tarea en FreeRTOS

Los estados de las tareas pueden ser:

- Ejecutando: La tarea está usando el procesador
- Disponible: La tarea está disponible para pasar a ejecución, pero aún no lo ha hecho porque hay otra tarea ejecutándose
- Bloqueada: Básicamente la tarea esta "esperando". Por ejemplo, porque ha hecho un vTaskDelay, está esperando un semáforo, etc. La tarea pasará a disponible cuando ocurra un evento (temporal, timeout, evento externo)
- Suspendida: La tarea está "parada", para lo cuál hay que llamar explícitamente a vTaskSuspend(). Se reactivará usando xTaskResume()

Una consideración muy importante en los sistemas operativos es el manejo de la memoria. Entender donde y cómo se alojan los datos nos permite evitar errores y asegurar el buen funcionamiento de nuestro programa. La memoria volátil (por ejemplo, la RAM) en la mayoría de los sistemas que usan microcontroladores se divide en 3 partes: estática ('static'), montón ('heap') y pila ('stack').

La memoria estática se utiliza para almacenar variables globales y variables designadas como 'estáticas' en el código (persisten entre llamadas a funciones). La pila

se utiliza para la asignación automática de variables locales. La memoria de la pila está organizada como un sistema de último en entrar, primero en salir (LIFO) para que las variables de una función puedan 'empujarse' a la pila cuando se llama a una nueva función. Al regresar a la primera función, las variables de esa función se pueden quitar, lo que la función puede usar para continuar ejecutándose donde lo dejó. El manejo de memorias de FreeRTOS lo podemos observar en la imagen número 10.

Cuando se crea una tarea en FreeRTOS (por ejemplo, con la función xTaskCreate), el sistema operativo asignará una sección de memoria dinámica para esa tarea en particular.

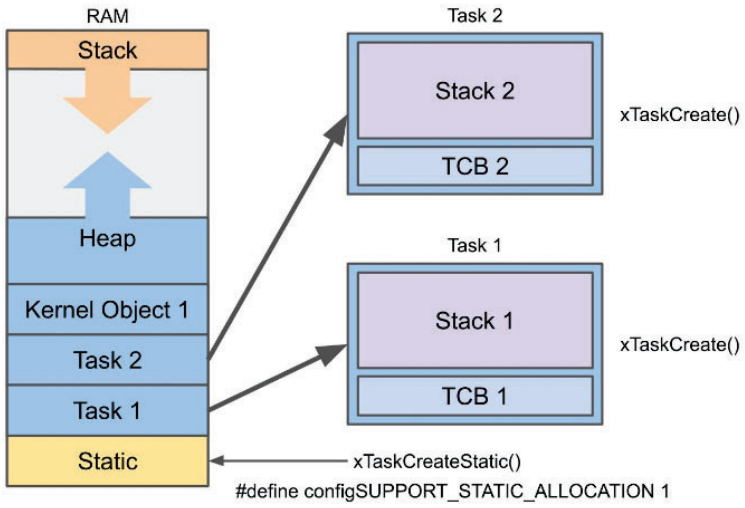


Figura 10: Uso de memoria en FreeRTOS

Una parte de esa memoria asignada es el Bloque de control de tareas (TCB), que se utiliza para almacenar información sobre la tarea, como su prioridad y el puntero de la pila local. La otra sección está reservada como una pila local que funciona igual que la pila global (pero en una escala más pequeña solo para esa tarea).

Las variables locales creadas durante las llamadas a funciones dentro de una tarea se envían a la pila local de la tarea. Debido a esto, es importante calcular el uso de pila previsto de una tarea con anticipación e incluirlo como parámetro de tamaño de pila en xTaskCreate().

Cuando se utiliza FreeRTOS, malloc() y free() no se consideran seguros para subprocesos. Como resultado, se recomienda utilizar pvPortMalloc() y vPortFree() en su lugar. Al usarlos, la memoria se asignará desde el montón global del sistema (en lugar del montón asignado para la tarea).

Las versiones recientes de FreeRTOS permiten la creación de tareas estáticas. Estos usan solo memoria estática (asignando su propia pila local y TCB en la memoria estática en lugar del montón). Esto es útil para situaciones en las que no puede o no

desea utilizar la memoria del montón para evitar desbordamientos del montón.

La creación dinámica de objetos RTOS tiene la ventaja de una mayor simplicidad y el potencial de minimizar el uso máximo de RAM de la aplicación:

- La asignación de memoria se produce automáticamente.
- La RAM utilizada por un objeto RTOS se puede reutilizar si se elimina el objeto.
- El esquema de asignación de memoria utilizado se puede elegir según el mejor ajuste para la aplicación.

Mientras que la creación de objetos RTOS utilizando RAM asignada estáticamente tiene la ventaja de proporcionar más control a la aplicación:

- Los objetos RTOS se pueden colocar en ubicaciones de memoria específicas.
- Permite utilizar RTOS en aplicaciones que simplemente no permiten ninguna asignación de memoria dinámica.
- Evita problemas relacionados con la memoria, como pérdidas de memoria, punteros colgantes y objetos indefinidos.

Como podemos observar en la imagen número 11, FreeRTOS ofrece distintos modos de comunicación para poder compartir los datos entre las tareas programadas:

- Grupos de eventos : Los grupos de eventos (event groups) son mecanismos que permiten a las tareas esperar y recibir múltiples señales en función de eventos específicos. Cada evento se representa mediante un bit en el grupo de eventos. Las tareas pueden esperar a que se establezcan ciertos bits en el grupo de eventos y luego proceder en función de las señales recibidas.
- Colas: Las colas son estructuras de datos que permiten a las tareas intercambiar datos de manera segura. Una tarea puede enviar datos a una cola y otra tarea puede recibirlos. Esto es útil para compartir información entre tareas de manera eficiente y garantizando que los datos no se corrompan. Las colas también pueden ser utilizadas para la sincronización, ya que una tarea puede bloquearse si intenta recibir de una cola vacía o enviar a una cola llena.
- Mutex/Semaphore : Los semáforos y mutex son mecanismos de sincronización que permiten a las tareas controlar el acceso a recursos compartidos. Los semáforos son más simples y pueden ser utilizados para la sincronización básica y la prevención de condiciones de carrera. Los mutex (o 'mutual exclusion semaphores') son una forma más avanzada de semáforos y se utilizan para garantizar que solo una tarea tenga acceso exclusivo a un recurso compartido en un momento dado. Si una tarea adquiere un mutex, ninguna otra tarea puede adquirirlo hasta que se libere.

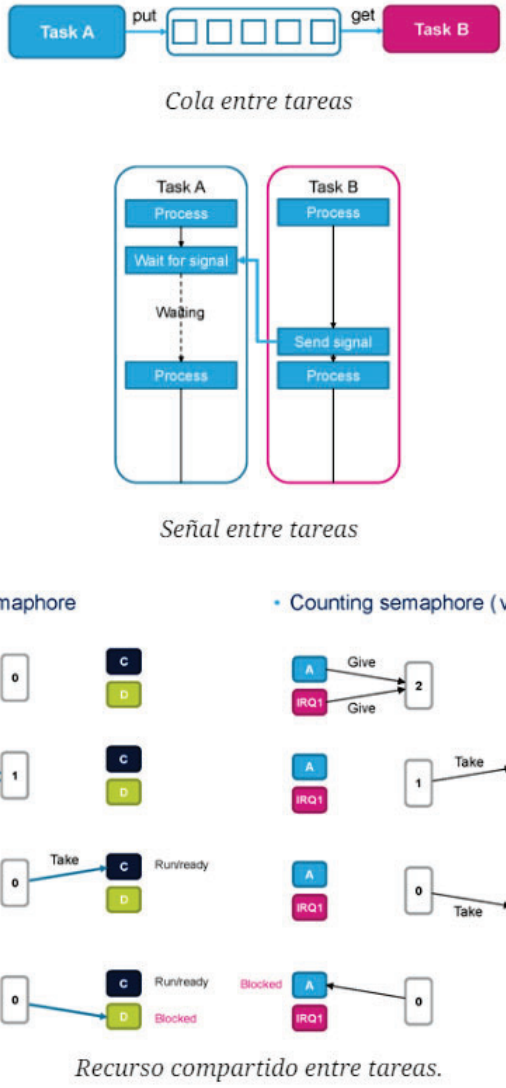


Figura 11: Comunicación entre distintas tareas en FreeRTOS

Estos mecanismos permiten que las tareas se comuniquen y coordinen de manera efectiva en sistemas multitarea, evitando condiciones de carrera y asegurando que los recursos compartidos se utilicen de manera segura y eficiente. Es importante utilizar estos mecanismos adecuadamente y planificar la sincronización y comunicación entre tareas de manera cuidadosa para evitar problemas de concurrencia y comportamiento no deseado.

3.5. Manejo de datos en nanosatélites

El manejo de datos en nanosatélites es un aspecto crucial para garantizar una misión exitosa. Los nanosatélites recopilan datos de sus subsistemas, sensores y cámaras, y luego transmiten estos datos a la Tierra para su procesamiento y análisis. Los datos pueden incluir imágenes, videos, telemetría, información de altitud y otros tipos de información relevante para la misión.

Los subsistemas en los nanosatélites se comunican entre sí utilizando diferentes tipos de protocolos de comunicación. La elección del protocolo dependerá de la naturaleza de la comunicación que se debe realizar entre los subsistemas y de las restricciones técnicas del satélite.

Por lo general, los subsistemas se comunican mediante un bus de comunicaciones, que es una red que conecta los diferentes subsistemas del satélite. El bus de comunicaciones puede ser de diferentes tipos:

- Inter-integrated Circuit (I2C): Este es un protocolo de comunicación serial de dos cables que se utiliza para la comunicación entre microcontroladores y sensores en el satélite. I2C es una forma simple y de bajo costo de interconectar múltiples dispositivos.
- Serial Peripheral Interface (SPI): Este protocolo de comunicación se utiliza para la comunicación en serie de alta velocidad entre microcontroladores y otros dispositivos electrónicos en el satélite. SPI se utiliza comúnmente para la transferencia de datos entre sensores y microcontroladores.
- Controller Area Network (CAN): CAN es un protocolo de comunicación serie de dos cables que se utiliza en los sistemas de control de automóviles, pero también se utiliza en los nanosatélites para la comunicación entre subsistemas, como los sistemas de control de la altitud.
- Universal Asynchronous Receiver/Transmitter (UART): UART es un protocolo de comunicación en serie que se utiliza para la transferencia de datos entre dispositivos en el satélite. UART se utiliza comúnmente para la comunicación entre microcontroladores y otros dispositivos, como módulos GPS.
- Ethernet: Este un protocolo de comunicación en red que se utiliza ampliamente en sistemas informáticos. En nanosatélites, Ethernet es utilizado para la comunicación de datos de alta velocidad.

Cada subsistema está conectado al bus de comunicaciones a través de un nodo, que es un dispositivo que permite la comunicación entre el subsistema y el bus de comunicaciones. Los nodos pueden ser microcontroladores o circuitos integrados especializados, dependiendo de la complejidad del subsistema y de las necesidades de comunicación.

Los subsistemas también pueden estar conectados directamente entre sí, sin pasar por el bus de comunicaciones. Esta conexión directa se utiliza generalmente para la comunicación crítica en tiempo real, como la comunicación entre el subsistema de control de altitud y el subsistema de propulsión.

3.5.1. CAN

Este protocolo se ha establecido como un estándar robusto y confiable en el ámbito de las comunicaciones, particularmente en entornos donde la fiabilidad y la eficiencia son fundamentales. Su funcionamiento se basa en un esquema de comunicación diferencial, en donde los datos se transmiten en forma de tensiones entre dos líneas, CAN High y CAN Low. De esta manera, es que minimiza la susceptibilidad a ruidos y perturbaciones electromagnéticas, y lo convierten en un protocolo muy robusto. Utiliza un protocolo de acceso al medio de tipo CSMA/CR (Carrier Sense Multiple Access with Collision Resolution). Este enfoque permite a los nodos verificar la disponibilidad del bus antes de transmitir y, en caso de colisiones, implementa un mecanismo de resolución eficiente.

Admite diversas velocidades de transmisión, lo que lo hace altamente versátil para adaptarse a diferentes requisitos de aplicaciones. Las velocidades típicas incluyen 125 kbit/s, 250 kbit/s, 500 kbit/s y 1 Mbit/s. La elección de la velocidad depende de la cantidad de datos a transmitir y de la distancia entre los nodos.

Las comunicaciones CAN se estructuran en tramas, siendo la trama estándar la más común. Una trama estándar consta de varios campos, como el Identificador de Trama, que indica la prioridad del mensaje, el campo de Datos, que lleva la información actual, y los campos de Control y CRC para garantizar la integridad de los datos.

Algunas de sus características son:

- Robustez:: La topología diferencial de CAN y su capacidad para detectar y resolver colisiones lo hacen idóneo para entornos espaciales donde las interferencias son comunes.
- Eficiencia en el ancho de banda: La estructura eficiente de las tramas y el protocolo CSMA/CR contribuyen a una utilización óptima del ancho de banda.
- Adaptabilidad: La selección de velocidades de transmisión permite adaptarse a distancias variables entre los nodos.
- Priorización de mensajes: La capacidad de asignar prioridades a los mensajes asegura una comunicación jerárquica.
- Identificador: El ID puede ser utilizado para dar información adicional en el mensaje.

La figura número 12 proporciona una representación gráfica de la señal CAN, en donde se observa la forma de onda de la señal, y la composición de la trama del mensaje, en donde vemos los bits del identificador, de Control, los 8 bytes de datos y el CRC.

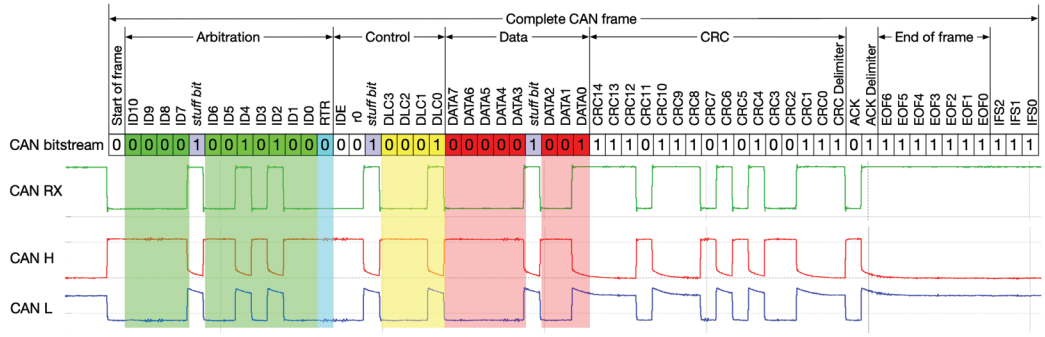


Figura 12: Frame de un mensaje CAN

Su robustez, eficiencia e inmunidad a las interferencias, lo convirtieron en un protocolo muy utilizado en la industria espacial, ya que asegura una comunicación fluida entre los distintos subsistemas.

3.5.2. Ethernet

El protocolo de comunicación Ethernet, inicialmente concebido como un protocolo para redes de área local (LAN), ha evolucionado y se ha expandido para abordar diversas aplicaciones, incluidas las comunicaciones satelitales. Su popularidad radica en su versatilidad, capacidad de transmisión de datos de alta velocidad y eficiencia en la utilización del ancho de banda.

Ethernet se basa en una topología de red de tipo estrella o bus, donde múltiples dispositivos se conectan a través de un concentrador o switch central. Utiliza un método de acceso al medio conocido como CSMA/CD (Carrier Sense Multiple Access with Collision Detection) para gestionar la transmisión de datos entre los nodos de la red.

Ethernet ofrece una variedad de velocidades de transmisión, desde 10 Mbps hasta varios Gbps. Esto permite adaptar la red a los requisitos específicos de la aplicación satelital, ya sea para la transferencia eficiente de datos científicos o para la gestión de sistemas a bordo.

El protocolo TCP/IP (Transmission Control Protocol/Internet Protocol) es fundamental en las comunicaciones Ethernet. Proporciona un stack de protocolos que facilitan la transmisión de datos de manera confiable y eficiente, permitiendo la conexión de dispositivos en una red global.

Algunas de sus ventajas son:

- Alta velocidad de transferencia: Las velocidades de transmisión de Ethernet, especialmente en sus variantes más avanzadas, garantizan la transferencia eficiente de grandes volúmenes de datos generados por instrumentos científicos a bordo del satélite.
- Flexibilidad y escalabilidad: La arquitectura de Ethernet permite la expansión y escalabilidad de la red para incorporar nuevos dispositivos.
- Confiabilidad: La capacidad de detección de colisiones de Ethernet y los mecanismos de corrección de errores en el nivel de transporte (TCP) contribuyen a la confiabilidad de las comunicaciones.
- Interoperabilidad: La amplia adopción de Ethernet en aplicaciones terrestres se traduce en la disponibilidad de componentes y tecnologías compatibles.

Dado su elevado rendimiento en la transmisión de datos a altas velocidades, Ethernet se ha consolidado como un protocolo ampliamente empleado en la comunicación con las cargas útiles. Su capacidad para transmitir de manera rápida los datos recolectados permite reducir la necesidad de tiempos prolongados de actividad, haciendo de Ethernet una elección destacada en entornos espaciales.

3.6. Subsistema de comunicación

Los subsistemas de comunicación son uno de los componentes claves en cualquier misión satelital, ya que permiten la transmisión de datos y el control del satélite desde la Tierra. Existen diferentes sistemas de comunicación que pueden ser utilizados en nanosatélites, cada uno con sus propias ventajas y desventajas.

Los sistemas de comunicación se dividen generalmente en dos categorías: subsistemas de comunicación de enlace ascendente (uplink) y subsistemas de comunicación de enlace descendente (downlink).

Los subsistemas de comunicación de enlace ascendente se encargan de la transmisión y recepción de datos, comunicándo el Payload con la estación terrena. Generalmente, estos datos se transmiten a través de ondas de radio que son captadas por una antena en el satélite. El sistema de comunicación de enlace ascendente debe ser lo suficientemente potente como para enviar una señal clara y fuerte al satélite, incluso a una distancia de cientos o miles de kilómetros.

Por otro lado, los enlaces descendentes se encargan de enviar los datos de Payload a la estación terrena. Estos datos se transmiten desde el satélite a la Tierra a través de la misma antena utilizada para el enlace ascendente. El sistema de comunicación de enlace descendente debe ser capaz de recibir una señal débil y distorsionada y convertirla en datos útiles.

Existen varios sistemas de comunicación que se utilizan en los nanosatélites, entre ellos los sistemas de RF (Radiofrecuencia) y otras tecnologías como óptica y comunicación por láser. A continuación se detallan algunos de los sistemas más utilizados en la comunicación con nanosatélites:

- Sistemas de RF: Los sistemas de RF son los más comunes en la comunicación con nanosatélites. Estos sistemas utilizan ondas de radio para transmitir datos desde el satélite a la Tierra y viceversa. Existen diferentes bandas de frecuencia que se utilizan en la comunicación con nanosatélites, incluyendo UHF, VHF, L, S, C, X y Ku.

- Óptica: La comunicación óptica utiliza la luz para transmitir datos. Esta tecnología se ha utilizado en algunos nanosatélites para la comunicación a alta velocidad y largas distancias. En este caso, el satélite utiliza un láser para transmitir datos a la Tierra.

- Comunicación por láser: Esta tecnología también utiliza la luz para la comunicación, pero en lugar de transmitir los datos en forma de ondas de radio, utiliza pulsos de láser. La comunicación por láser se utiliza a menudo para la comunicación inter-satélite en sistemas de constelaciones de nanosatélites.

Existen diferentes bandas de frecuencia que se utilizan para la comunicación con nanosatélites. A continuación se describen algunas de las bandas de frecuencia más comunes utilizadas:

1. Banda UHF (Ultra High Frequency): Esta banda se encuentra entre los 300 MHz y los 3 GHz y se utiliza comúnmente para la comunicación con nanosatélites. La banda UHF es popular debido a que las antenas y los equipos necesarios son relativamente pequeños y livianos, lo que los hace ideales para su uso en nanosatélites.

2. Banda VHF (Very High Frequency): Esta banda se encuentra entre los 30 MHz y los 300 MHz y se utiliza para comunicaciones de corto alcance. Aunque la banda VHF tiene una menor capacidad de transmisión de datos que la banda UHF, es menos susceptible a la atenuación de la señal debido a la lluvia o la neblina, lo que la hace útil para la comunicación terrestre-satélite.

3. Bandas L y S: Estas bandas se encuentran entre los 1 GHz y los 2 GHz y los 2 GHz y los 4 GHz, respectivamente. Se utilizan a menudo para la comunicación con satélites geoestacionarios y también son adecuadas para la comunicación con nanosatélites.

4. Banda X: Esta banda se encuentra entre los 8 GHz y los 12 GHz y se utiliza a menudo en sistemas de comunicaciones militares y gubernamentales. También se utiliza en la comunicación con nanosatélites.

5. Banda Ku: Esta banda se encuentra entre los 12 GHz y los 14 GHz y se utiliza a menudo en sistemas de televisión por satélite y telecomunicaciones.

Por lo general, todos los sistemas de comunicación tienen una arquitectura similar. Están constituidos por las antenas (pueden ser una para Tx y otra para Rx, o una sola para ambas funciones), el receptor (en donde se amplifica la señal y se la cambia a una frecuencia más baja) y el demodulador (en el cual se extrae la información, y se la pasa a la POBC), el transmisor (en el cual se modula y codifica la señal a transferir), y una etapa de potencia, en donde se amplifica la señal antes de su transmisión a Tierra. Esta arquitectura la podemos observar en la imagen número 13.

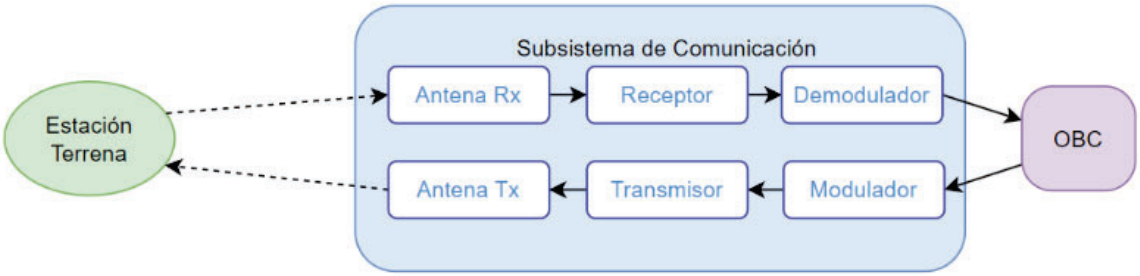


Figura 13: Diagrama del Subsistema de Comunicación satelital

La comunicación en los nanosatélites no es solo una transmisión de información, sino es más bien una coordinación de objetivos interconectado, que se rigen como pilares fundamentales para el éxito de la misión. En caso de no poder obtener comunicación con un satélite que está en órbita, se lo puede considerar como completamente perdido, por más que el resto de sus subsistemas estén funcionando correctamente. Es por eso, que el Subsistema de Comunicación está dividido en las siguientes partes:

- Confirmación de supervivencia del satélite: En el espacio, la confirmación de que el satélite está operativo es vital. Este propósito se logra mediante un 'beacon' o señales CW Morse. Estas señales transmiten la existencia continua del satélite, estableciendo un enlace vital con la Tierra.
- Transmisión de datos: Datos digitales, como temperatura, tensiones, corriente y parámetros de altitud, son encapsulados en paquetes y transmitidos hacia la estación terrena.
- Control del satélite: Además del envío de telemetría, deben tener la capacidad de recibir telecomandos. A través de este canal bidireccional, es posible modificar los modos operativos, activar la Carga Útil de la misión y registrar el cronograma de la misión. Esta capacidad de control desde la Tierra potencia la flexibilidad de las operaciones en el espacio.
- Obtención de información de distancia: Para misiones que exploran los confines más lejanos, obtener información de distancia es esencial. Este propósito, conocido como 'ranging', permite calcular la distancia entre el satélite y la Tierra.

3.6.1. Consideraciones de diseño

Para la elección de un Subsistema de comunicación, las consideraciones de diseño desempeñan un papel esencial en el desarrollo y operación exitosa de los sistemas de comunicación en nanosatélites. Estas consideraciones son cruciales debido a varios motivos clave:

- Requisitos de la misión: La elección del sistema de comunicación depende de los objetivos de la misión. Si la misión requiere alta velocidad de transmisión, como en misiones de imágenes en tiempo real, la tecnología óptica podría ser preferible. Para misiones de recopilación de datos científicos a largo plazo, la eficiencia energética y la estabilidad podrían ser más importantes.

- Limitaciones de energía y espacio: Los nanosatélites tienen restricciones de energía y espacio, lo que influye en la selección de sistemas de comunicación. Las tecnologías de baja potencia como LoRa pueden ser ideales para prolongar la vida útil de la batería. Las antenas deben ser compactas y tener una ganancia adecuada para maximizar la eficiencia de la comunicación.

- Condiciones atmosféricas y Frecuencias: Las frecuencias de comunicación se seleccionan considerando las condiciones atmosféricas y la atenuación de la señal. Por ejemplo, la banda UHF es menos susceptible a la atenuación debido a la lluvia. Las bandas de frecuencia más altas, como la banda Ku, ofrecen más ancho de banda pero pueden ser más afectadas por la atmósfera.

- Integración con equipos de a bordo: Los sistemas de comunicación deben integrarse con otros equipos de a bordo, como sensores y sistemas de propulsión. La compatibilidad y la interferencia electromagnética deben considerarse en el diseño para garantizar un funcionamiento certero.

- Adaptabilidad y flexibilidad: Los sistemas de comunicación deben ser adaptables a diferentes órbitas y condiciones operativas. Algunos nanosatélites pueden operar en constelaciones o en misiones interplanetarias, lo que requiere sistemas de comunicación capaces de mantenerse conectados a largas distancias.

3.6.2. Presupuesto de enlace

En todos los sistemas de comunicación, uno de los pasos más importantes es el cálculo del presupuesto del enlace. El presupuesto de enlace de una comunicación por RF es un conjunto de parámetros que describen un enlace en términos de niveles de potencia necesarios para establecer una comunicación fiable entre transmisor y el receptor. Una medida del rendimiento de un enlace por satélite es la relación de la intensidad de la señal medida a la potencia de ruido en la entrada del receptor, y los cálculos del balance del enlace a menudo se ocupan de determinar esta relación. Aquí que se considera el escenario de comunicación de enlace ascendente. Por lo

tanto, para medir el balance de enlace de este tipo de comunicación, proporcionamos a continuación algunos detalles de los componentes o factores que contribuyen a la ganancia a lo largo de este enlace de comunicación. En los siguientes puntos.

- Potencia Isotrópica Radiada Equivalente (EIRP): es una medida de la potencia transmitida.

$$EIRP = P_{\mathrm{Tx}}|_{\mathrm{dBW}} - L|_{\mathrm{dB}} + G_{\mathrm{Tx}}|_{\mathrm{dBi}} \tag{1}$$

  En donde PTx es la potencia transmitida, L es la pérdida en la línea de transmisión y GTx es la ganancia de la antena del transmisor.

- Pérdidas de espacio libre: es la pérdida de intensidad de una señal electromagnética cuando viaja a través del espacio libre en un trayecto con visibilidad directa desde una antena transmisora hasta una antena receptora. Su expresión viene dada por la siguiente ecuación:

$$L_{\mathrm{SF}} = 20 \log_{10} \left(\frac{4\pi d f_{\mathrm{c}}}{c} \right) \tag{2}$$

  En donde fc es la frecuencia de la portadora.

- Atenuación atmosférica: A medida que la señal viaja a través de la atmósfera, las diferentes moléculas que componen el aire absorben su energía a diferentes velocidades dependiendo de la frecuencia de la portadora. Por lo tanto, la pérdida atmosférica puede clasificarse como atenuación de la energía de la señal. Una de las pérdidas atmosférica es dada por la lluvia, que depende de la frecuencia y suele ser mayor para las frecuencias más altas. Las bandas utilizadas comercialmente, en particular las bandas Ku, K y Ka, se ven mucho más afectadas que las bandas por debajo de 1 GHz.

La elección de la banda de frecuencia para la comunicación con nanosatélites dependerá de varios factores, incluyendo el tipo de misión, el tamaño y peso del nanosatélite, y las condiciones específicas de la ubicación del satélite y del centro de control terrestre.

Dentro de estos sistemas de comunicación, se ha observado una leve tendencia creciente en el uso de sistemas de radio de largo alcance (LoRa), debido a su capacidad de proporcionar un bajo consumo de energía, bajo costo y alto rendimiento en términos de distancia y penetración de obstáculos. La implementación de LoRa en nanosatélites permite una comunicación bidireccional con una potencia de transmisión baja, lo que resulta en una mayor vida útil de la batería del nanosatélite. Además, su baja tasa de datos es adecuada para la transmisión de pequeñas cantidades de datos de telemetría, que son comunes en misiones de nanosatélites.

En cuanto a los estudios y experimentos previos sobre sistemas de comunicación en nanosatélites que utilizan LoRa, se han reportado varias misiones exitosas, como el

Proyecto HaloSat de la NASA, el CubeSail de la Universidad de Michigan y la misión ESEO de la Agencia Espacial Europea. En general, estos estudios muestran que LoRa es un sistema de comunicación viable y efectivo para las misiones de nanosatélites, y es especialmente útil para las aplicaciones que requieren una transmisión de datos de baja tasa.

La elección del sistema de comunicación para un nanosatélite depende de una serie de factores, incluyendo los requisitos de la misión, la disponibilidad de recursos y el presupuesto. Aunque existen diferentes sistemas de comunicación disponibles, el uso de LoRa se ha vuelto cada vez más popular en misiones de nanosatélites debido a su bajo consumo de energía, bajo costo y alto rendimiento en términos de distancia y penetración de obstáculos.

3.7. LoRa

Este capítulo se centra en la tecnología de comunicación LoRa (Long Range), una tecnología inalámbrica que demostró ser especialmente valiosa en aplicaciones de comunicaciones a larga distancia y bajo consumo de energía. Se analizarán los principios fundamentales de LoRa, sus características claves y su relevancia en el campo de las comunicaciones.

3.7.1. Redes LPWAN

Las redes LPWAN ('Low Power Wide Area Network') son redes inalámbricas de largo alcance que permiten la comunicación entre dispositivos (nodos finales) y una estación base ('gateway'). Pueden cubrir entre 1 y 10 Km en zonas urbanas y de 10 a 40 Km en zonas rurales (Bajic, Chaxel, Meyer, 2019). Están diseñadas para conectar objetos (sensores) o máquinas entre sí, sin requerir de la asistencia de un usuario. La conexión entre objetos es denominada IoT ('Internet of Things'), y la conexión entre máquinas se denomina M2M ('Machine to Machine). Además de ser una comunicación de largo alcance, las LPWAN tienen un bajo consumo y bajo costo. Su bajo consumo de energía permite una autonomía del dispositivo durante varios años, mientras que su bajo costo favorece su comercialización. Algunos tipos de LPWAN son Sigfox, LoRa, Wize, NB-IoT.

La tecnología LoRa ha sido exitosamente implementada en la comunicación terrestre con 256 millones de dispositivos finales instalados alrededor del mundo, y se están comenzando a explorar sus capacidades de comunicación entra la Tierra y el espacio. En el año 2017, usando un globo meteorológico se logró el récord mundial de distancia máxima que puede recorrer un paquete de datos LoRaWAN, llegando a 702,67 Km. En el año 2019, el récord fue batido por otra conexión globo-Tierra de 766 Km y en el 2020 el récord aumentó a 832 Km (The Things Network, 2020). En 2019, Lacuna Space y Fossa Systems probaron exitosamente la comunicación Lora con satélites en LEO. Estas dos empresas, junto a varios otras como Astrocast, Orbcomm y Kepler Communications pretenden proporcionar cobertura IoT mundial

a través de constelaciones de satélites en LEO. En 2019, un grupo de desarrolladores programó un firmware para placas ESP32 compatibles con los módulos LoRa SX126x y SX127x con el objeto de crear una red abierta de estaciones distribuida alrededor del mundo. De esta manera, cualquiera que lo deseara podría construir una estación terrestre y recibir paquetes LoRa de los satélites que contaban con esta tecnología abierta, ayudando así al seguimiento de los mismos. El proyecto se llamó TinyGS y actualmente cuenta con 1054 estaciones terrestres activas en todo el mundo, como podemos observar en la figura número 14. Hasta la fecha, hay unos 27 CubeSats en LEO que pertenecen a universidades y pequeñas empresas que quieren hacer pruebas con LoRa (TinyGS, 2022c). Por lo tanto, LoRa está ganando cada día más impulso como una opción de bajo coste, la que resulta muy atractiva para los sistemas de comunicación por satélite.

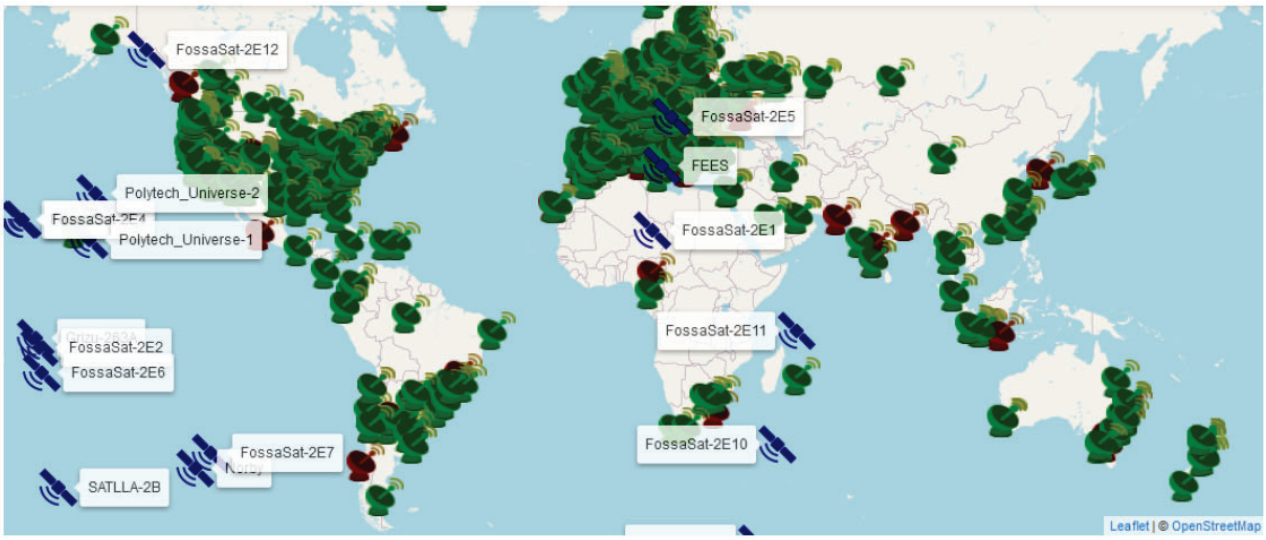


Figura 14: Estaciones terrenas desplegadas en el mundo por el proyecto TinyGS

Existen muchas tecnologías disponibles de comunicación inalámbrica que pueden utilizarse. Cada uno de ellas se adapta mejor a diferentes aplicaciones dependiendo de los requisitos de velocidad de datos y rango de comunicación y de las restricciones de consumo y coste.

Las LPWAN no son la mejor opción para monitorizar variables que cambian a un ritmo rápido o para transmitir bloques de datos pesados (como vídeos) porque sólo son capaces de enviar pequeños paquetes de datos a velocidades de transmisión bajas. Sin embargo, ofrecen comunicación de largo alcance con bajo consumo de energía y a bajo coste, por lo que son una opción muy atractiva cuando se necesitan datos de sensores y se dispone de poca energía. Con las LPWAN es posible optimizar la producción y el funcionamiento, minimizar las pérdidas, reducir costes, tomar decisiones basadas en datos y mejorar la gestión.

Existen LPWAN con y sin licencia. y LoRa es parte del grupo que no requiere de licencia para su uso. Opera en la banda libre de Industria, Ciencia y Medicina (ISM), la cual es usada en todo el mundo de manera libre. Las frecuencias de la banda ISM varían dependiendo de la región del mundo en dónde se encuentre.

Hay dos tipos de dispositivos LoRa disponibles:

- Nodos LoRa: Son módulos con una antena y un microprocesador, típicamente conectados a una batería que los alimente. Colectan información de sensores y la transmiten. Pueden recibir datos. Es capaz de recibir varios paquetes LoRa al mismo tiempo usando factores de dispersión (SF - 'Spreading Factor') en distintos canales.

- LoRa Gateway: Son módulos de radio con antenas y microprocesador. Se encuentran alimentados por la red eléctrica. Es el puente entre los nodos y los dispositivos finales. Se conecta a los nodos finales a través de una red de baja potencia, y se conectan al servidor de red a través de una red de gran ancho de banda. Es un transistor con múltiples canales. Un canal se refiere a una frecuencia de portadora con información.

LoRa es un tipo de modulación, perteneciente a la familia del espectro ensanchado. La modulación LoRa está patentada y pertenece a Semtech, quienes han otorgado una licencia a otros fabricantes de chips para utilizarla. LoRa se hace referencia a la capa física (PHY) de la red. La capa de control de acceso al medio (MAC) y la capa de aplicación se engloban en lo que se denomina el estándar LoRaWAN.

3.7.2. Espectro Ensanchado

Cuando se utiliza una sola frecuencia para la transmisión, es muy fácil interceptarla. Puede hacerse simplemente sintonizando una radio hasta detectar la frecuencia portadora de la señal. Sin embargo, cuenta con la desventaja de sufrir interferencias, ya que si se transmite otra señal con la misma frecuencia que la portadora (o una frecuencia muy cercana a la de la portadora), el receptor escuchará ambas señales simultáneamente y no entenderá el mensaje original.

Una forma de reducir estos problemas es usando la modulación de espectro ensanchado, en la que la señal se envía utilizando muchas frecuencias. En el espectro ensanchado, la señal que contiene la información se modifica de tal manera que la señal final que se transmite tiene un ancho de banda mucho mayor que la señal original. Existen varias técnicas de espectro ensanchado como: Espectro Ensanchado de Secuencia Directa (DSSS), Espectro Ensanchado por Salto de Frecuencia (FHSS) y Espectro Ensanchado Chirp (CSS).

- Espectro Ensanchado de Secuencia Directa (DSSS): cada bit de la señal original de una frecuencia baja es multiplicado por una señal de alta frecuencia denominada código de propagación. Su resultado es una señal modulada con

mayor ancho de banda. Los bits del código de dispersión se denominan 'chips'. Además, la secuencia de bits del código de dispersión no cambia, y es sólo conocida por el transmisor y el receptor. El receptor necesita conocerlo para poder decodificar el mensaje. Cuanto más largo es el código de propagación, más tarda la demodulación, por lo que no es una técnica ideal cuando se busca velocidad y bajo consumo. Además, necesita de una fuente de reloj de alta precisión. Esta modulación la podemos observar en la imagen número 15

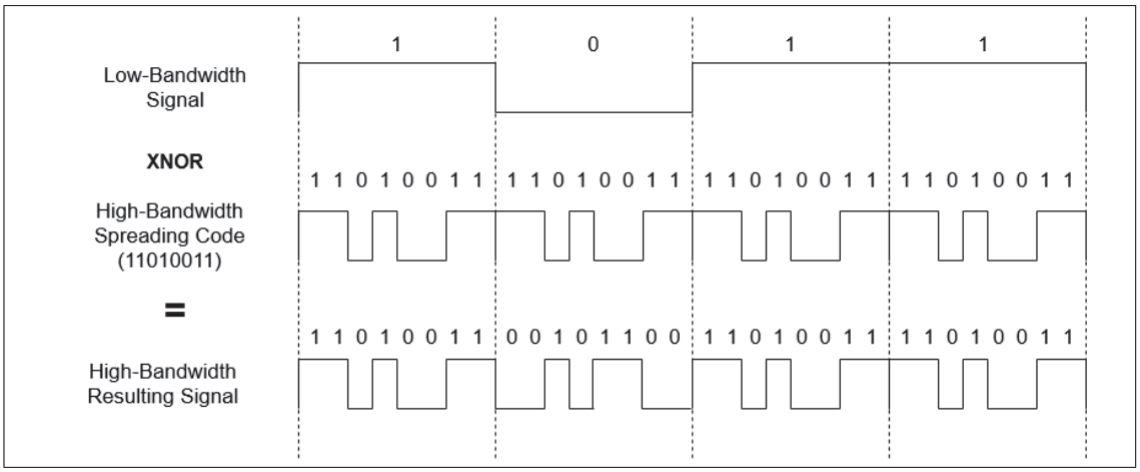


Figura 15: Ejemplo de Espectro Directo Ensanchado

- Espectro Ensanchado con Salto de Frecuencia (FHSS): En FHSS, la señal se envía a través de diferentes frecuencias portadoras durante un tiempo fijo, llamado tiempo de permanencia. Así, la señal se transmite primero utilizando una frecuencia portadora determinada durante el tiempo de permanencia y, a continuación, salta a otra frecuencia portadora y la utiliza hasta que finaliza el tiempo de permanencia. Y así sucesivamente. El tiempo que se tarda de cambiar de una portadora a otra se denomina tiempo de salto, y la secuencia de salto de frecuencia que se va a utilizar se denomina código de dispersión. Esta técnica la podemos observar en la imagen número 16.

- Espectro Ensanchado Chirp (CSS): En CSS, la señal portadora está formada por chirps. Un chirp (también conocido como señal de barrido) es un pulso de frecuencia variable. En una onda sinusoidal en donde la frecuencia cambia linealmente. Si la frecuencia aumenta linealmente ascendente en el tiempo, se la denomina chirp ascendente, y si disminuye en el tiempo se llama chirp descendente. En la imagen número 17 podemos observar cómo son los chirps ascendentes y descendentes. El ancho de banda de un chirp es igual al ancho de banda espectral de la señal, lo que se representa como la frecuencia más alta menos la frecuencia más baja. En resumen, en CSS la información es codificada

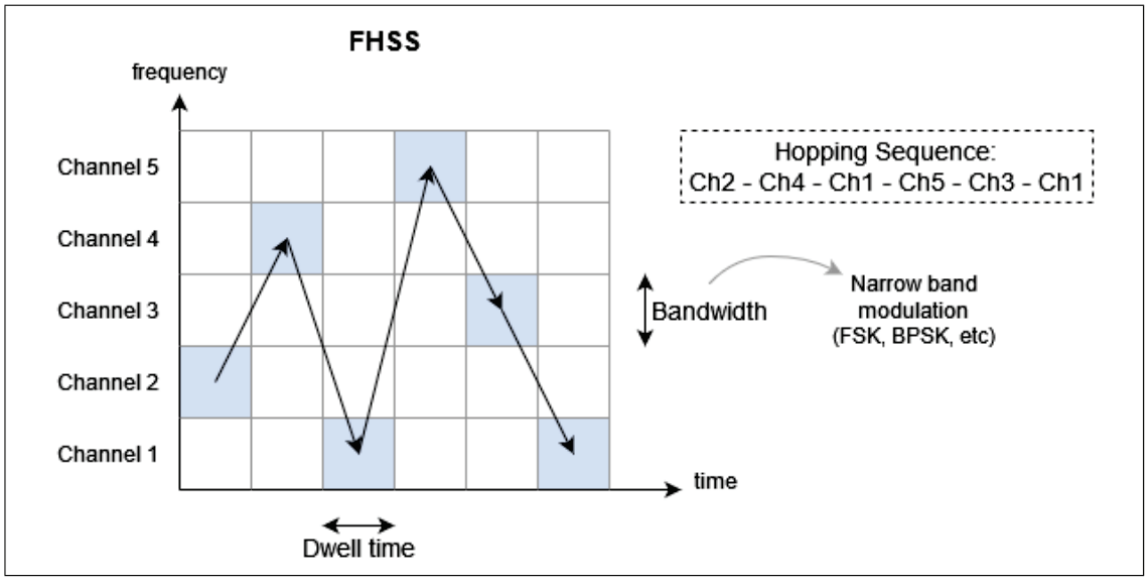


Figura 16: Ejemplo de Espectro Ensanchado con Salto de Frecuencia

en pulsos de chirp. En la figura número 18 podemos ver cómo es que cambia la frecuencia de la señal en el tiempo, con los chirps ascendentes y descendentes.

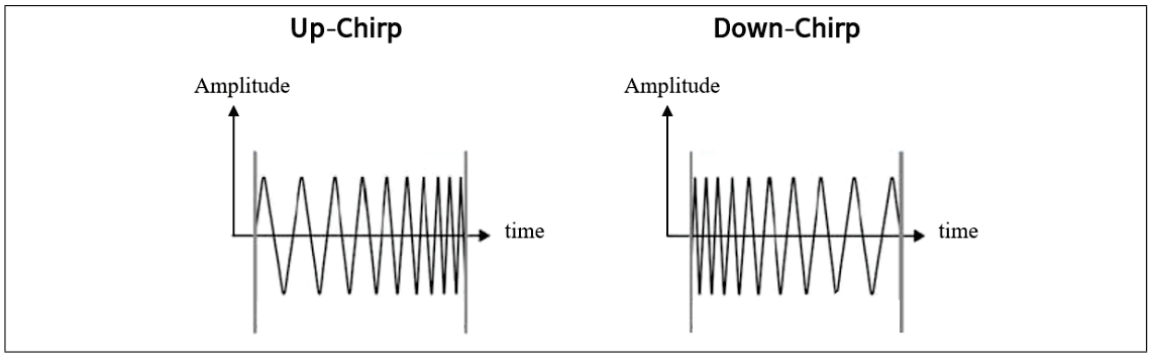


Figura 17: Amplitud vs Tiempo de la señal de barrido

3.7.3. Modulación LoRa: Modulación Chirp por desplazamiento de frecuencia

La modulación LoRa se representa como saltos de frecuencia en las señales de chirp, como puede observarse en la figura 19, en donde un símbolo es el elemento que transporta la información en la modulación digital, y cada símbolo representa un número. Así, el transmisor envía una secuencia de símbolos que quiere transmitir. Por

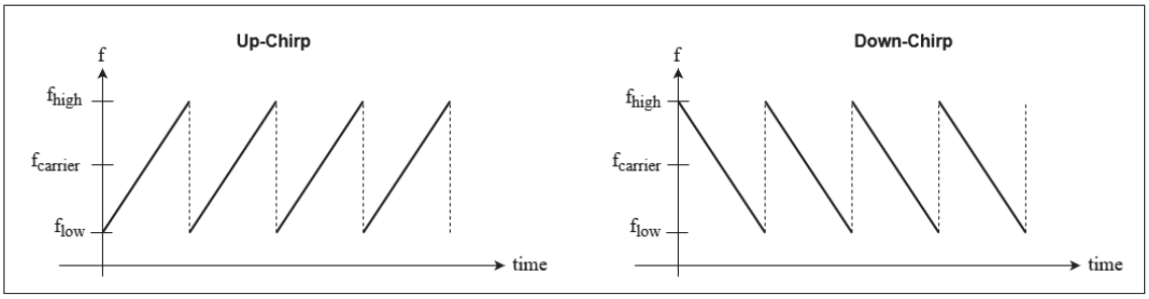


Figura 18: Frecuencia vs Tiempo de la señal portadora

otro lado, el receptor debe ser capaz de identificar qué símbolos ha de recibir para comprender el mensaje.

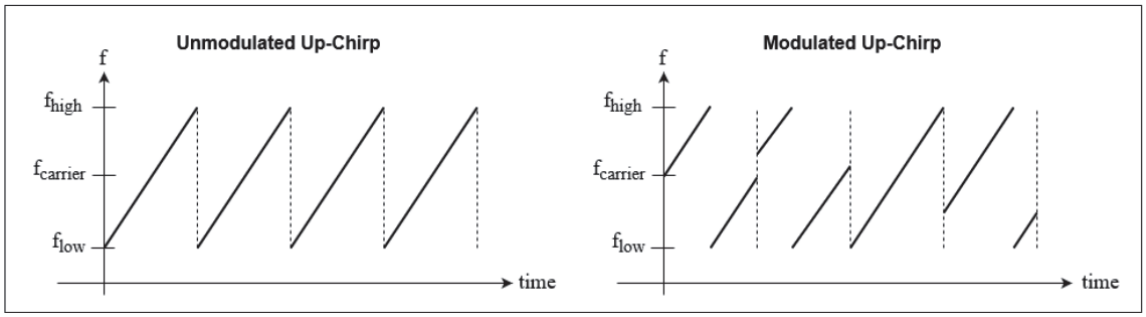


Figura 19: Izquierda: Portadora sin modular. Derecha: Portadora modulada.

Por ejemplo, en la modulación ASK ('Amplitud Shift Keying'), va a modularse por la amplitud de cada símbolo. Si sólo se consideran dos amplitudes (alta y baja), cada símbolo contiene un bit.

- Amplitud baja = 0
- Amplitud alta = 1

En la modulación PSK ('Phase Shift Keying'), la fase es el símbolo. Entonces, si hay cuatro fases diferentes para codificar la información, entonces cada símbolo contiene 2 bits.

En la modulación LoRa, el símbolo es el desplazamiento de frecuencia al principio de la señal chirp, como se muestra en la figura 20. Es por esto, que la modulación LoRa es conocida FSCM ('Frequency Shift Chirp Modulation').

En esta técnica de modulación, el número de bits que se pueden codificar un símbolo, es denominado factor de dispersión (SF, por 'Spreading Factor'). Este parámetro

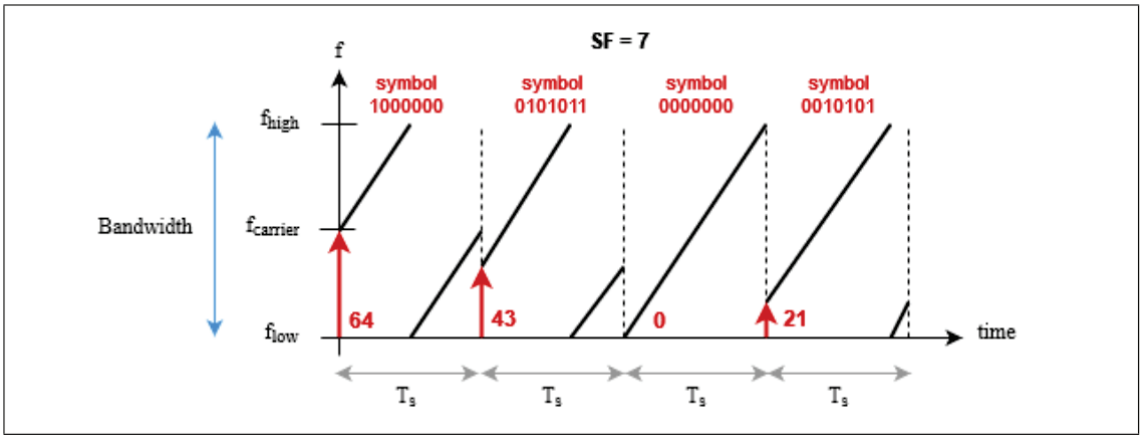


Figura 20: FSCM: Cada símbolo se codifica en la señal chirp ascendente como un desplazamiento de frecuencia

puede tomar un valor entre 6 y 12, el cual va a depender de las necesidades de a aplicación y las condiciones del entorno. Por lo tanto, el número de símbolos *s*. utilizados en la modulación LoRa es:

$$\#s = 2^{SF} \qquad \text{for } SF = \{6, 7, 8, 9, 10, 11, 12\}$$

En donde cada símbolo (en base decimal) pude representar cualquier valor en *s=0, 1, 2, 3, ...,(2SF - 1)*. Matemáticamente, esto se representa cómo:

$$s(nT_s) = \sum_{h=0}^{SF-1} w_h(nT_s) \cdot 2^h$$

En dónde Ts es el periodo del símbolo (el tiempo que se tarda en transmitir un símbolo compelo), n es el contador de símbolos y $w(nTs)$ es un vector de dígitos binarios de *SF*.

Por ejemplo, si el tercer símbolo que se transmite ($n = 3$) es 43, y el $SF = 7$, entonces:

$$s(3T_s) = \begin{pmatrix} w_0(3T_s) \\ w_1(3T_s) \\ w_2(3T_s) \\ w_3(3T_s) \\ w_4(3T_s) \\ w_5(3T_s) \\ w_6(3T_s) \end{pmatrix} \cdot \begin{pmatrix} 2^0 \\ 2^1 \\ 2^2 \\ 2^3 \\ 2^4 \\ 2^5 \\ 2^6 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 4 \\ 8 \\ 16 \\ 32 \\ 64 \end{pmatrix} = 43$$

En cuando al ancho de banda, es un parámetro mismo del diseño de la modulación. Según la hoja de datos, los ancho de bandas disponibles para la modulación LoRa son: 7,8 kHz, 10,4 kHz, 15,6 kHz, 20,8 kHz, 31,25 kHz, 41,7 kHz, 62,5 kHz, 125 kHz, 250 kHz o 500 kHz.

El ancho de banda va a determinar la frecuencia con la que se transmite una muestra de la señal chirp. Según Semtech (Semtech, 2015), cada muestra se envía cada:

$$T[s] = \frac{1}{\text{BW}} \tag{3}$$

En donde cada señal de chirp tiene 2^{SF}, por lo que el periodo del símbolo se calcula como:

$$Ts[s] = 2^{SF} \cdot T = \frac{(2^{SF})}{\text{BW}} \tag{4}$$

LoRa se ha convertido en una opción popular para los nanosatélites debido a su capacidad de larga distancia, bajo consumo de energía y alta sensibilidad, lo que lo hace una opción atractiva y justificada para su implementación.

Algunas de las ventajas sobre su uso son:

1. Larga distancia: Puede transmitir datos a largas distancias, lo que hace que sea una buena opción para las misiones de nanosatélites que requieren comunicación a grandes distancias.

2. Bajo consumo de energía: Utiliza una técnica de modulación de baja potencia que permite a los nanosatélites enviar datos a la Tierra con un bajo consumo de energía, lo que es esencial en misiones de nanosatélites con limitaciones de energía.

3. Sensibilidad: Es capaz de detectar señales débiles en entornos con mucho ruido, lo que es importante para la transmisión de datos en el espacio donde hay interferencias y ruido electromagnético.

4. Flexibilidad: Puede ser configurada para diferentes rangos de frecuencia y ancho de banda, lo que lo hace una opción flexible para los nanosatélites con diferentes requerimientos de transmisión de datos.

5. Ancho de banda ajustable: La tecnología LoRa permite ajustar el ancho de banda para adaptarse a diferentes entornos de comunicación y requisitos de datos.

6. Inmunidad al ruido: La modulación de espectro ensanchado utilizada en LoRa proporciona una alta inmunidad al ruido y la interferencia, lo que garantiza una alta fiabilidad de la comunicación.

7. Red de malla: La arquitectura de red de malla permite que los dispositivos se comuniquen entre sí, sin necesidad de una conexión directa a una estación base o antena central, lo que proporciona una mayor flexibilidad en la distribución de la red.

En cuanto a la información sobre nanosatélites que utilizan LoRa, hay varios proyectos y misiones en los que se ha utilizado esta tecnología. Por ejemplo, el proyecto 'The Things Network' ha desarrollado una red de satélites que utiliza LoRa para proporcionar comunicación de Internet de las cosas (IoT) en todo el mundo. Otro ejemplo es la misión 'Swarm B' de la Agencia Espacial Europea, que utiliza LoRa para la comunicación de satélite a satélite en una formación de satélites. También hay varios proyectos de satélites educativos que utilizan LoRa para la comunicación, como el proyecto 'PocketQube' y 'CubeSatSim'.

LoRa cuenta con varias ventajas en términos de larga distancia, bajo consumo de energía, sensibilidad y flexibilidad. LoRa se ha utilizado en varios proyectos de nanosatélites y ha demostrado ser una tecnología prometedora para la comunicación en el espacio.

En cuanto a la justificación del uso de LoRa, hay varios factores que lo hacen atractivo para su implementación en nanosatélites. En primer lugar, LoRa es un sistema de comunicación de larga distancia, lo que significa que es posible enviar señales desde el nanosatélite a la Tierra sin necesidad de grandes antenas o sistemas de transmisión de alta potencia. Además, LoRa es un sistema de baja potencia, lo que significa que el consumo de energía es menor en comparación con otros sistemas de comunicación, lo que es esencial en los nanosatélites, donde el consumo de energía es un factor crítico. Por último, LoRa ofrece una alta sensibilidad, lo que significa que es capaz de detectar señales débiles en entornos con mucho ruido, lo que es una ventaja en el espacio donde hay interferencias y ruido electromagnético.

Finalizando este capítulo, observamos como ésta tecnología demuestra ser un recurso invaluable para establecer comunicaciones confiables a larga distancia, gracias

a su capacidad de comunicación y bajo consumo de energía. En el siguiente capítulo, nos adentraremos en cómo aplicamos esta tecnología en nuestro proyecto y cómo contribuye a nuestros objetivos de investigación y comunicación.

4. Proyecto

En el siguiente capítulo, vamos a explicar el proyecto mismo en el cual hemos trabajado. Este capítulo se divide en dos secciones principales. En la primera mitad, explicaremos el Firmware de la computadora de a bordo de Payload, desglosando su arquitectura, sistema operativo, diseño y desarrollo. En la segunda mitad, nos adentraremos en el Subsistema de Comunicación de Payload, examinando su diseño funcional, la elección de protocolos, el desarrollo de código y el presupuesto de enlace.

4.1. Firmware de la POBC

Esta primera sección nos enfocamos en el desarrollo del Firmware de la POBC. Desglosaremos minuciosamente la arquitectura, el sistema operativo, el diseño y el desarrollo. Veremos cómo la POBC desempeña un papel central al guiar y controlar cada función, siendo el núcleo inteligente que controla las operaciones de nuestro Payload y contribuye significativamente a la investigación en la tecnología satelital.

4.1.1. Arquitectura Principal

La arquitectura de la POBC abarca la disposición y el diseño de sus componentes. La elección y disposición estratégica de los puertos y protocolos de comunicación, junto con la inteligencia del firmware, forman un sistema robusto y adaptable. Esta arquitectura garantiza que la operación del satélite se realice de manera segura y eficaz. El hardware utilizado es perteneciente a Texas Hércules, y reconocido en distintos proyectos espaciales. Por acuerdos firmados, no se divulgará el nombre exacto del mismo.

Continuando con el firmware de la computadora de a bordo de Payload, se define como el cerebro tecnológico de nuestro Payload, encargado de orquestar cada aspecto de su funcionamiento. Este software se despliega en múltiples modos y capas, para garantizar la operación segura y efectiva del satélite. Para su implementación, se lo dividió en cuatro modos principales, debido a la necesidad de gestionar diferentes aspectos de la operación del Payload de manera eficiente y organizada. Cada una de estas partes cumple una función específica y esencial para garantizar el éxito de la misión.

1. Modo de Inicialización: Este modo es crucial porque prepara el Payload para su operación en el espacio. Durante esta fase, se realizan las comprobaciones iniciales de integridad de los sistemas, se configuran las interfaces de comunicación y se garantiza que todo esté en condiciones óptimas para el funcionamiento subsiguiente.

2. Modo de Operación Nominal: En este modo, el Sistema Operativo (OS) toma el control y gestiona todas las operaciones regulares del satélite. Es el estado en el que el satélite realiza sus tareas científicas, recopila datos y se comunica

con la estación terrestre. Esta parte es esencial para el funcionamiento diario y normal del Payload.

3. Modo de Backup: La redundancia y la capacidad de recuperación son fundamentales en cualquier misión espacial. El modo de respaldo es necesario para garantizar la continuidad de la misión en caso de que ocurran eventos inesperados o anomalías. Esta parte proporciona una capa adicional de seguridad al permitir la conmutación rápida a sistemas de respaldo en caso de problemas.

4. Modo de Actualización: La tecnología y las condiciones en el espacio pueden cambiar con el tiempo. Para mantener al Paylaod actualizado y capaz de abordar nuevos desafíos, el modo de actualización permite la introducción de mejoras y parches en el sistema operativo y las aplicaciones. Esto garantiza que el satélite pueda adaptarse a las condiciones cambiantes y aprovechar oportunidades emergentes.

En el ámbito de la comunicación, la POBC cuenta con una arquitectura versátil, que incorpora dos puertos CAN (Controller Area Network) y un puerto Ethernet, cada uno de ellos desempeñando un papel fundamental en el intercambio de información y datos con diferentes componentes del satélite.

Los dos puertos CAN, cuidadosamente integrados en la POBC, son utilizados para la comunicación con el PSC. El protocolo CAN es conocido por su confiabilidad y eficiencia, lo que lo convierte en la elección ideal para establecer una comunicación sólida entre la POBC y el subsistema encargado de las transmisiones de datos. A través de estos puertos, la POBC recibe los comandos de control y datos, permitiendo una comunicación bidireccional sin problemas.

Además de su conexión CAN, la POBC está equipada con un puerto Ethernet. Este puerto, que se asemeja a la red de comunicación de una computadora convencional, cumple un papel esencial en la interacción con la Carga Útil, que suele estar compuesta por instrumentos científicos sofisticados, y requiere una comunicación de alta velocidad y de gran ancho de banda. El puerto Ethernet proporciona esta capacidad, permitiendo la transmisión adecuada de datos entre la POBC y la Carga Útil. Esta interface de comunicación de alto rendimiento es fundamental, ya que garantiza que los datos científicos se recopilen y transmitan de manera eficiente y precisa.

En la figura 21 mostramos cómo está conformada la arquitectura principal del Payload, y cómo es que se encuentran comunicados cada uno de los subsistemas. Podemos observar en la parte central a nuestra POBC, la cual va a estar comunicándose con el PSC y con la Carga Útil. El Subsistema de Comunicación de Payload, que cuenta con dos transmisores y un receptores LoRa, los cuales se comunican por SPI, contiene dos módulos CAN BUS, de los cuales mencionaremos en detalle más adelante. Por otro lado, la POBC ya cuenta con dos CAN BUS integrados, y el puerto de comunicación Ethernet, con el cual va a comandar a la Carga Útil, la cual simulamos mediante una computadora.

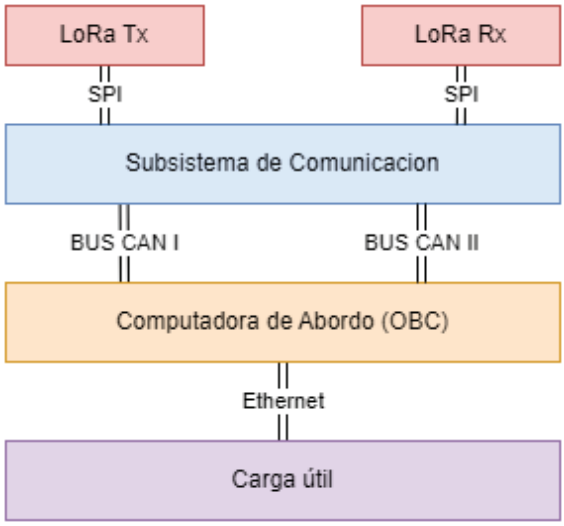


Figura 21: Arquitectura principal del Payload

Esta sólida arquitectura principal del firmware de la POBC sienta las bases de un funcionamiento confiable, adaptable y escalable para nuestro Payload. Con sus cuatro modos distintivos, cada uno desempeñando un papel crítico, con el objetivo de cumplir la misión a la cual fuese destinado.

4.1.2. Sistema Operativo

Luego de un minucioso análisis de las opciones de sistemas operativos compatibles con nuestro kit de desarrollo, tomamos la decisión de implementar nuestro Modo Nominal de operación en FreeRTOS. Esta elección se basó en una serie de factores críticos, y fue seleccionado debido a su probada capacidad en la gestión de tareas en tiempo real, su eficiencia en el uso de recursos y su adaptabilidad a sistemas embebidos. En el ámbito espacial, donde la eficiencia energética y la confiabilidad son fundamentales, estas características se vuelven esenciales para garantizar un funcionamiento responsable y eficiente.

Un aspecto significativo que respalda nuestra elección fue la compatibilidad de FreeRTOS con el kit de desarrollo que utilizamos. Esto asegura que el sistema operativo funcione en correctamente con nuestro hardware, brindando estabilidad y coherencia en el rendimiento de la POBC.

Para la configuración y desarrollo de la POBC de nuestro nanosatélite, hicimos uso de una herramienta de generación de controladores y otros recursos esenciales. Su perfecta integración con FreeRTOS nos permitió generar el código necesario sin problemas. Además, pudimos aprovechar la activa comunidad de usuarios y desarrolladores que se encuentran en los foros de este sistema operativo. Esta comunidad no solo proporcionó valiosos recursos y soporte técnico, sino que también ofreció solucio-

nes a los desafíos técnicos que surgieron durante el desarrollo.

La elección de FreeRTOS representó una decisión sólida y estratégica que respalda la fiabilidad de nuestro Payload, dos aspectos cruciales para alcanzar el éxito en nuestro objetivo. Este sistema operativo es una base sólida sobre la cual construimos y optimizamos la funcionalidad de la POBC, asegurando un funcionamiento sin contratiempos y eficiente en el entorno espacial. A continuación, explicaremos con mayor detalle el diseño del firmware, donde aprovechamos las capacidades de ese sistema operativo.

4.1.3. Diseño del Firmware

Como observamos en el capítulo anterior, el firmware se encuentra compuesto por cuatro modos principales. Para el diseño de cada uno de ellos, utilizamos distintos diagramas en bloques, que nos sirvieron de base para diseñar el comportamiento y codificar nuestro kit. La base principal de nuestro firmware puede observarse en la figura 22. En esta figura pueden verse los cuatro modos por los que está compuesta nuestra POBC (Modo Inicialización, Actualización del Firmware, Modo Backup, y el Modo principal), y las tareas que componen al sistema operativo, con sus distintas funciones. Al ser un sistema sincrónico, requiere de una fuente de sincronización, por lo cual en la figura también se encuentra el GPS. En las siguientes secciones entraremos en detalle en cada uno de los modos y de las tareas que programamos para nuestro firmware.

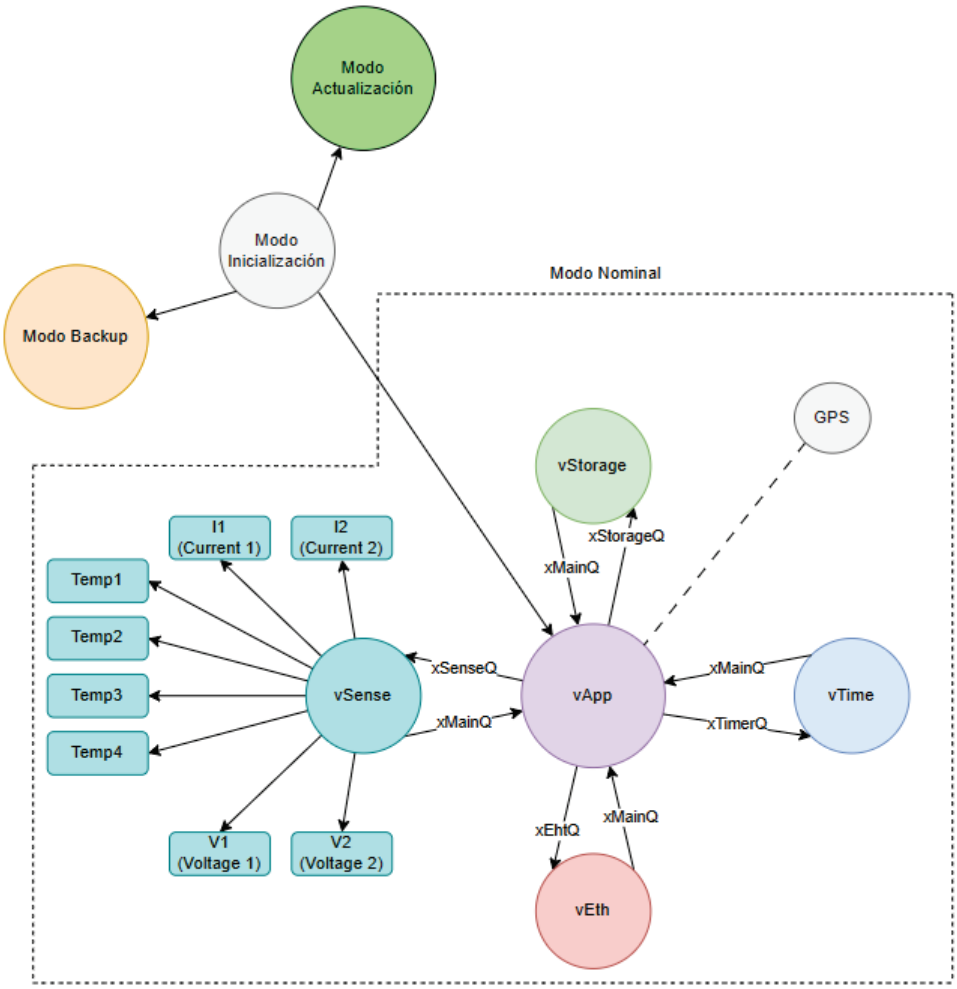


Figura 22: Topología del Firmware de la POBC

4.1.3.1. Modo Inicialización o System

Este modo actúa como el punto de partida crítico en la operación de nuestra POBC. Una vez que la POBC se enciende, automáticamente ingresa al *Modo Inicialización*, que es el centro de control desde el cual se toman decisiones cruciales. En este estado, la POBC aguarda indicaciones desde la la Estación Terrena, desde donde se le proporcionará instrucciones precisas sobre qué aplicación debe cargar y ejecutar, desde las diversas posiciones de memoria disponibles. Las opciones disponibles en este punto incluyen la ejecución del *Modo Nominal*, que es el programa principal de nuestra misión, el *Modo Backup*, que actúa como un modo seguro en caso de contingencias, y el *Modo Actualización*, en caso de considerar que el Modo Nominal deba ser actualizado.

En este modo, la POBC inicia su proceso de inicialización y, simultáneamente, envía un comando 'System', seguido de un comando 'Alive' a través de cada uno de los puertos CAN hacia el PSC. El comando 'System' tiene como objetivo indicar que la POBC se encuentra en Modo Inicialización, mientras que el comando 'Alive' indica a la Estación Terrena que el Payload está operativo y esperando instrucciones. Si no hay respuesta de confirmación por parte de la Estación Terrena, este proceso de envío de comandos 'Alive' se va a repir tres veces por cada puerto CAN. Si después de estos intentos, la POBC no recibe respuesta de la Estación Terrena, se activa automáticamente el *Modo Principal*.

En la figura 23 podemos observar el diagrama en bloques el *Modo Inicialización*. Para comenzar, va a ejecutar la función 'SysInit' la cual se encarga de inicializar los periféricos y las interrupciones para poder usar la comunicación CAN. Luego, va crear una variable llamada 'Frame', la cual va a iniciar con el valor en cero, para tener un registro de los comandos 'Alive' que se envían hacia la Estación Terrena. Las funciones 'Write canREG1' y 'Write canREG2' se encargan de cargar el comando Alive en los registros de CAN y de transmitir el comando por cada uno de los puertos. La función 'Read canREG1' se encarga de corroborar si hay algún mensaje CAN recibido en ese puerto. Este proceso lo va a realizar hasta enviar tres comandos 'Alive' por cada uno de los puertos CAN. En caso de recibir un comando desde la Estación Terrena en el que se indique que debe ejecutar el Modo Nominal o el Modo Actualización, va a enviar un ACK como comando de respuesta y procederá a la ejecución del modo solicitado.

En caso de haber completado los intentos de comunicación hacia la Estación Terrena y de no recibir ningún tipo de respuesta, por default va a ejecutar el *Modo Nominal*, y procederá a la ejecución de dicho modo de manera habitual.

Durante esta fase, la POBC opera en un modo de bajo consumo, minimizando el uso de recursos para preservar la energía y prolongar la vida útil de la batería. La Carga Útil, en este punto, permanece apagada, enfocándose en establecer una conexión segura con la Tierra y en la espera de las próximas indicaciones.

De esta manera, concluimos con el desarrollo del Modo Inicialización, pasando ahora al desarrollo del Modo Nominal.

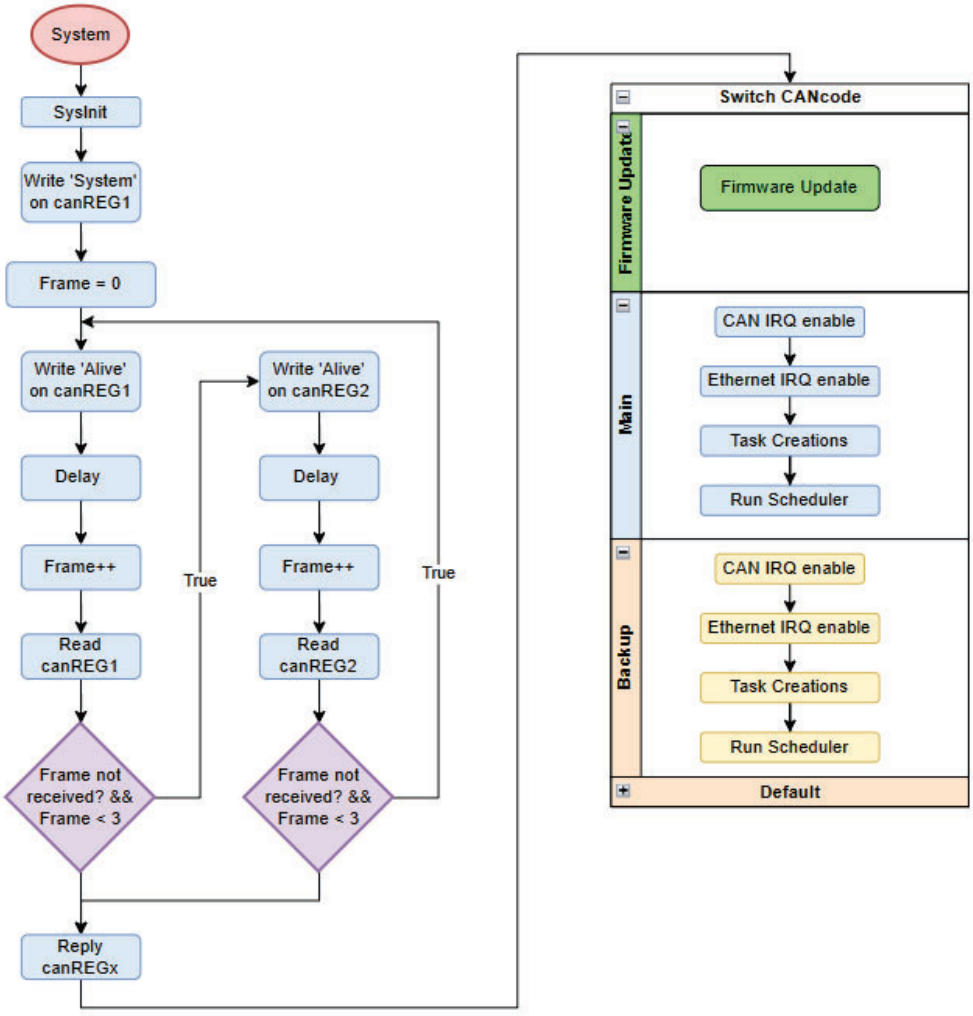


Figura 23: Diagrama funcional del Modo Inicialización

4.1.3.2. Modo Nominal

Este modo es una fase crucial en la operación. Es en donde la POBC ejecuta la última versión del sistema operativo cargada en su memoria, el cual ha sido cuidadosamente desarrollado en FreeRTOS. Este sistema operativo de tiempo real es la pieza clave de la POBC, siendo la aplicación principal encargada de gestionar y ejecutar una serie de tareas críticas que permiten que la misión del Payload se desarrolle en tiempo y forma. Quien va a coordinar todas las operaciones del satélite va a ser el FreeRTOS, desde la adquisición de datos de los sensores hasta el procesamiento de información, la gestión de comunicaciones y la activación de sistemas de respaldo en caso de contingencias. Este sistema operativo es quien asegura que cada aspecto de la misión se desarrolle de manera precisa y adecuada. Para lograr este objetivo, desarrollamos distintas tareas, cada una destinada a una función y misión específica.

Dentro del **Modo Nominal**, la Aplicación Principal (llamada vApp) es la tarea principal de la POBC, y la encargada de llevar el control de las operaciones. Esta tarea crítica, ya que se encarga de la ejecución y supervisión de las diversas tareas que componen al sistema operativo. Desde el monitoreo de sensores hasta la gestión de la Carga Útil, esta aplicación opera en un entorno de tiempo real para garantizar que todos los sistemas funcionen en armonía y de acuerdo con los parámetros de la misión.

En total, este modo está compuesto por seis tareas distintas, cada una con su función única y esenciales en el funcionamiento del Payload, las cuales pasaremos a explicar en detalle a continuación:

- **vApp**: Esta es la tarea principal del programa, y es quien está encargada de controlar al resto de las tareas. Es considerada la mente de la POBC, encargada de coordinar todas las operaciones del Payload. vApp actúa como un supervisor inteligente, garantizando la precisión y eficiencia de la misión en curso. Esta tarea es altamente versátil y adaptable, desempeñando múltiples roles estratégicos en la operación de la POBC. Al recibir un comando desde la Estación Terrena, esta tarea va a ser la encargada de decodificar la información y ejecutar la instrucción recibida. Sus responsabilidades se pueden describir en los siguientes ítem:

  1. Recepción y decodificación de comandos: vApp decodifica los comandos recibidos a través de los puertos CAN y los interpreta para determinar la acción necesaria.
  2. Transmisión de comandos: Además de recibir, vApp puede transmitir comandos y mensajes a otros componentes o la Estación Terrena.
  3. Control de tareas: Gestiona el ciclo de uso de las tareas, habilitándolas o deshabilitándolas según las necesidades de la misión.
  4. Inteligencia operativa: Agrega inteligencia operativa al Payload, tomando decisiones basadas en comandos y datos recibidos.

5. Fácil actualización: Para actualizar el funcionamiento del Modo Nominal a una nueva versión, sólo deberíamos trabajar sobre esta tarea, ya que es la tarea principal, y se encarga de las comunicaciones y el manejo de las demás tareas.

En la figura 24 se puede observar cómo diagramamos la tarea vApp. Diseñamos meticulosamente su flujo de trabajo para garantizar el control y una respuesta eficiente a las condiciones de trabajo. Al ejecutarse por primera vez esta tarea, está programada para activar las interrupciones de los dos puertos CAN, por medio de las funciones 'EnableErrorNotification(canREG1)' y 'EnableErrorNotification(canREG2). Una vez activadas estas interrupciones, va a ingresar a un bucle 'for'. Dentro de este bucle, se encuentra un 'if' el cual va a tener una respuesta positiva en caso de recibir un comando por CAN, o si se supera el tiempo de suspensión de esta tarea. En caso de no recibir ningún comando, va a ingresar en el otro 'if' en donde va a corroborar si hay algún experimento pendiente para que realice la Carga Útil. En caso de una respuesta positiva, va a corroborar con el OBT si es el tiempo correcto de ejecución y va a proceder a comunicarse con la Carga Útil, enviándole todos los comandos correspondientes para establecer el experimento. Este segundo 'if' lo pensamos para poder ejecutar los experimentos en caso de no tener visibilidad con el nanosatélite. En caso de que no se reciba ningún comando por CAN, de que el FreeRTOS no considere activarla y de que no hallan experimentos pendientes, la tarea va a permanecer suspendida, para así lograr un menor consumo de energía y de memoria de procesamiento de la CPU.

Siguiendo el diagrama en bloques de la figura 24, al recibir un comando, desarrollamos un gran 'switch case', en el cual se contemplan todos los posibles comandos que se puedan recibir por CAN. Dependiendo de cuál sea el comando recibido, la vApp va a ejecutar una función particular, respondiendo a las necesidades del mismo comando. En cada caso, la POBC va a responder a la Estación Terrena enviando el mismo comando que recibió, de manera que desde la Estación Terrena puedan corroborar que el comando que enviaron fue recibido y ejecutado correctamente.

Luego de pasar por el 'switch case', la tarea vApp va a corroborar si es necesario el envío de telemetría hacia la Estación Terrena. El envío de la telemetría puede solicitarse a través del comando 'CAN_TELE_REQUEST', y en caso de ser activado, la POBC va a enviar datos propios de su subsistema cada vez que la vApp sea activada por el sistema operativo. Lo programamos para un tiempo de activación cada dos segundos, por lo que en caso de querer telemetría de la POBC, la Estación Terrena va a tener un refresco de datos cada dicho tiempo.

Cuando un mensaje CAN es detectado, vApp utiliza una estructura de selección (Switch Case) para determinar las acciones a tomar según el identificador (ID) del mensaje, permitiendo un procesamiento específico y adaptativo de los

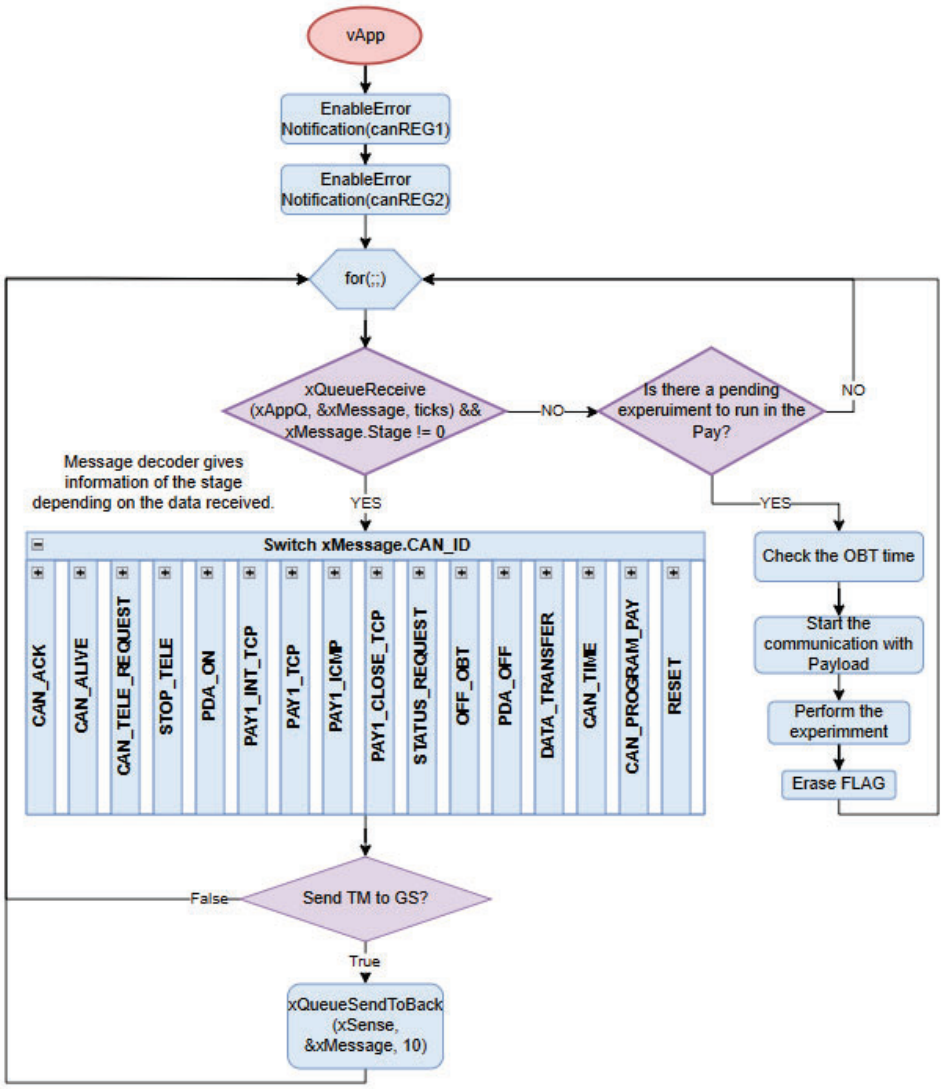


Figura 24: Composición de la tarea vApp

comandos recibidos. Además, supervisa si se ha solicitado la transmisión de telemetría desde la Estación Terrena y, en caso afirmativo, activa la tarea vSense y coordina la transmisión de la información necesaria hacia la Tierra. Este minucioso proceso asegura que la tarea vApp se comunique con otros subsistemas y tome decisiones inteligentes en tiempo real. Otra de las funciones de la vApp, es guardar en la memoria EEPROM la cantidad de mensajes recibidos por CAN, el OBT, las mediciones fuera del rango establecido por vSense, y los posibles problemas que puedan llegar a surgir.

- **vSense**: Esta tarea juega un papel esencial en la misión de nuestro Payload al ser la encargada de recopilar información vital sobre el estado y el rendimiento de la POBC. Inicialmente, al arrancar el sistema operativo, vSense se encuentra en un estado de inactividad, y es la tarea vApp la que tiene la capacidad de habilitarla mediante comandos específicos provenientes de Tierra. Una vez activada, vSense comienza a recolectar telemetría propia de la POBC. En la figura 25 podemos observar el diagrama funcional de esta tarea, la cual resulta de una complejidad menor que la tarea vApp. Al ejecutarse, lo que va a hacer es leer la información de tensiones, de temperaturas y de corrientes de los sensores que dispone. Como esa información se encuentra en forma analógica, va a convertirla a digital por medio de los conversores ADC, para luego cargarla y enviarla por el comando CAN_STATUS hacia la Estación Terrena.

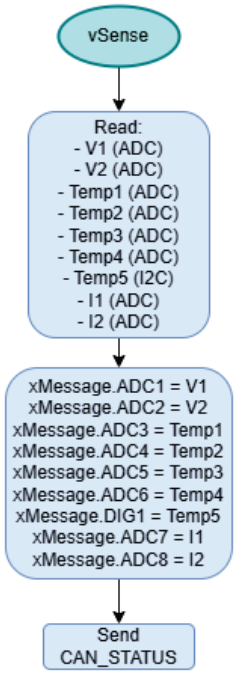


Figura 25: Arquitectura de la tarea vSense

La telemetría recopilada abarca una amplia gama de parámetros, como la temperatura de la POBC, los niveles de tensión y corriente, el estado de los sistemas de control y otros indicadores clave de rendimiento. Estos datos se capturan a intervalos regulares de 2 segundos, proporcionando una visión en tiempo real del funcionamiento del Payload en el entorno espacial.

Un aspecto notable de vSense es su capacidad de respuesta a las necesidades de la misión. Puede ser activada selectivamente para recopilar información específica cuando se requiere una supervisión detallada o durante eventos críticos de la

misión. Esta adaptabilidad garantiza un uso efectivo de los recursos y permite la obtención de datos en momentos estratégicos.

- **vTime**: La tarea vTime desempeña un rol crítico en la sincronización y el control temporal de todas las operaciones realizadas por nuestro Payload. Su función principal radica en calcular y transmitir el OBT (Tiempo de a bordo) de la POBC hacia la Tierra. La singularidad de esta tarea reside en su capacidad para operar de manera autónoma, independientemente de la visibilidad del nanosatélite por parte de la Estación Terrena.

  El OBT es una referencia temporal fundamental que guía la ejecución de procedimientos, la activación de sensores y la toma de decisiones en todo el sistema. Dado que el nanosatélite puede pasar largos períodos fuera del rango de comunicación directa con la Estación Terrena, el OBT se convierte en la pieza clave de la sincronización. Cada acción y evento programado se realiza en función del tiempo marcado por esta tarea, asegurando una coordinación precisa de las operaciones, incluso en condiciones de comunicación interrumpida.

  La transmisión del OBT hacia la Estación Terrena es realizada mediante solicitudes específicas, permitiendo a los operadores en Tierra mantener un registro del tiempo a bordo del Paylaod. Esto facilita la planificación y la ejecución de comandos, así como el seguimiento de eventos a lo largo de la misión. La tarea vTime opera con una eficiencia excepcional, garantizando que el tiempo sea un aliado en lugar de un desafío en el complejo entorno espacial.

  Utilizamos el OBT para marcar el tiempo en que se llevan a cabo las operaciones críticas del satélite, como las operaciones, la recolección de datos de telemetría y la transmisión de datos a tierra. También, es de utilidad al querer ejecutar algún procedimiento en un tiempo específico que no se tenga visibilidad del nanosatélite.

  Es importante que el OBT esté sincronizado con el tiempo absoluto en tierra, para lo cual utilizamos procedimiento de sincronización. Esta técnica se basa en la transmisión de un mensaje con un offset, y utilizado para corregir la derivación del OBT.

  El OBT es fundamental para el correcto funcionamiento del satélite, y su cálculo y mantenimiento son responsabilidad de la tarea vTime. Es importante que el OBT se mantenga actualizado en todo momento para evitar errores en la sincronización y el control de la misión del satélite.

- **vStorage**:

  La tarea vStorage se encarga de la gestión y el almacenamiento de datos críticos en la memoria de la POBC, y provenientes desde la Carga Útil o de la POBC. Su función principal es asegurar la integridad y disponibilidad de datos esenciales durante toda la misión del nanosatélite.

Algunas de sus funciones claves son:

- Almacenamiento Seguro: La tarea vStorage se encarga de garantizar que los datos importantes, como la telemetría (TM) y otros registros críticos, se almacenen de manera segura en la memoria no volátil de la POBC. Esto incluye la gestión de archivos y la organización de datos para su fácil recuperación y análisis.
- Respaldo y Recuperación: La tarea vStorage establece rutinas de respaldo periódicas para asegurarse de que no se pierdan datos críticos en caso de un reinicio o un evento inesperado. Además, facilita la recuperación de datos en situaciones de falla o reinicio.

- **vEth**: La tarea vEth se encarga de gestionar la interface Ethernet de la POBC con la Carga Útil. Esto incluye el intercambio de datos y comandos críticos para el funcionamiento de la Carga Útil.

  Algunas de sus funciones y responsabilidades son:

  - Generación del cliente TCP: Es la responsable de la creación y gestión de un cliente TCP. Esto implica establecer conexiones TCP salientes con otros nodos, para poder comandar a la Carga Útil. La tarea se encarga de la inicialización del cliente TCP y la gestión de las conexiones activas.
  - Manejo de Protocolo TCP: La tarea vEth implementa el protocolo TCP para garantizar la entrega segura de datos entre la POBC y la Carga Útil. Esto incluye el seguimiento del estado de las conexiones, la retransmisión de datos si es necesario y la administración de ventanas de recepción y envío.
  - Manejo de Protocolo ICMP: Además del protocolo TCP, la tarea vEth maneja el protocolo ICMP (Internet Control Message Protocol). ICMP se utiliza para el control y la señalización de errores en la capa de red. La tarea puede enviar y recibir mensajes ICMP, lo que es esencial para la comunicación de diagnóstico y control.
  - Gestión de Colas y Temporizadores: Para garantizar un funcionamiento adecuado y sincronizado, la tarea vEth utiliza colas y temporizadores para la gestión de eventos y datos. Esto permite una comunicación eficaz con otras tareas y la coordinación de operaciones de red.

  La tarea vEth la diseñamos y programamos meticulosamente utilizando el stack de bibliotecas de lwIP para garantizar un rendimiento adecuado y un uso efectivo de los recursos en un entorno altamente limitado. lwIP es una implementación de código abierto del conjunto de protocolos de Internet (IP) diseñada específicamente para sistemas embebidos y de recursos limitados. Proporciona las funciones básicas de comunicación en red, como TCP, UDP (Protocolo de

Datagramas de Usuario) e IP, pero está diseñado para ser eficiente en cuanto al uso de recursos como la memoria y la CPU

Para poder comunicarnos con la Carga Útil, hemos desarrollado y programado las funciones de TCP, desarrollando distintas funciones para su uso, ya que estas capacidades no estaban disponibles de forma nativa en el stack de bibliotecas para ser usadas en nuestro kit de desarrollo. Esta adaptación ha sido fundamental para habilitar la comunicación crítica requerida, lo que demostró nuestra capacidad de innovación y resolución de problemas.

Esta tarea fue diseñada con un enfoque en la eficiencia y la optimización de recursos. Dado que los recursos en un nanosatélite son limitados, es fundamental minimizar el consumo de energía y la carga de CPU. La programamos para utilizar los recursos de manera organizada y garantizar un rendimiento sólido.

- **vIdle**:

  La tarea vIdle, o 'Idle Task', es una tarea fundamental en el sistema operativo FreeRTOS la cual se ejecuta cuando no hay otras tareas en estado activo para ejecutarse. Los sistemas Operativos siempre necesitan tener una tarea que se encuentre activa, aún cuando no se esté realizando ninguna acción en concreto. Es por eso que creamos esta tarea ya que sin ella el sistema operativo no podría funcionar. Algunas de sus funciones claves son:

  - Gestión de recursos: vIdle ayuda en la gestión productiva de los recursos de la POBC cuando no se están utilizando para tareas críticas. Reduce el consumo de componentes no esenciales para ahorrar energía.
  - Ahorro de energía: Cuando no hay tareas críticas para ejecutar, vIdle puede poner en modo de bajo consumo ciertos componentes de hardware, lo que contribuye al ahorro de energía en el nanosatélite.

De esta manera, concluimos con el desarrollo del Modo Nominal, pasando a explicar ahora al desarrollo del Modo Backup.

4.1.3.3. Modo Backup

Este modo es esencial y cumple la función de otorgarle robustez, garantizando la integridad y continuidad de las operaciones incluso en circunstancias adversas. Lo programamos como una versión segura y protegida del *Modo Nominal*, alojada en una región de memoria especialmente resguardada y de solo lectura. El principio fundamental detrás del *Modo Backup* fue proporcionarle una capa de redundancia que garantice que el Payload pueda continuar su misión en el improbable pero crítico escenario en el que el *Modo Nominal* se corrompa o quede inoperable. Esta región de memoria inmutable alberga la versión Backup del programa.

Inicialmente, la versión Backup y la versión del *Modo Nominal* cumplen con los mismos objetivos, el cual es el de brincar la comunicación CAN con el PSC y de

controlar la Carga Útil. Sin embargo, el modo Backup no contiene un sistema operativo, sino que está constituido en Bare Metal. Este no es un detalle menor, y la decisión de esta diferencia fue la gran complejidad de realizar los saltos de memoria hacia dos programas que contengan sistema operativos. Igualmente, este modo ofrece funcionalidades similares a las del Modo Nominal, ya que permite la misma recepción de comandos por CAN, el uso de memoria SD, EEPROM, capacidad de guardar y enviar telemetría propia de la POBC, y la capacidad de comunicarse por Ethernet con la Carga Útil. Al no contar con un sistema operativo, la gestión de recursos, como asignaciones y liberación de memoria puede ser más compleja, y no garantiza los tiempos de respuesta que sí se puede hacer en un sistema operativo. Por otro lado, al no contar con la capa adicional de complejidad asociada con la gestión que sí tiene los sistemas operativos, tiene un tiempo de arranque más rápido y una interacción con periféricos y hardware más directa y precisa (lo cual es ideal para un modo Backup o de Emergencia).

En la figura 26 podemos observar la arquitectura de este modo. Al iniciar, va a enviar un comando CAN_BACKUP, el cual tiene como objetivo indicarle a la Estación Terrena que ya se encuentra en este modo de operación. Luego, inicia las interrupciones por CAN de ambos puertos e inicia los periféricos y las funciones que van a ser utilizadas. Cada vez que se reciba un mensaje por CAN, va a ingresar el switch case. Ahí, va a discriminar el mensaje recibido por el ID, y va a realizar una función determinada dependiendo del tipo de mensaje. En caso de recibir un mensaje por Ethernet, el manejo va a ser por la interrupción, en donde se guarde la información recibida, y se envía un mensaje por CAN hacia la Estación Terrena, con la información propicia al mismo.

La robustez del *Modo Backup* radica en su capacidad para mantenerse inalterable y completamente funcional a pesar de cualquier eventualidad que afecte a su modo complementario. Esta característica se convierte en un seguro contra fallas, permitiendo a la POBC seguir operando de manera efectiva y cumplir con su cometido incluso en situaciones críticas. La existencia del *Modo Backup* garantiza la fortaleza de nuestra misión y el aprovechamiento máximo de las oportunidades.

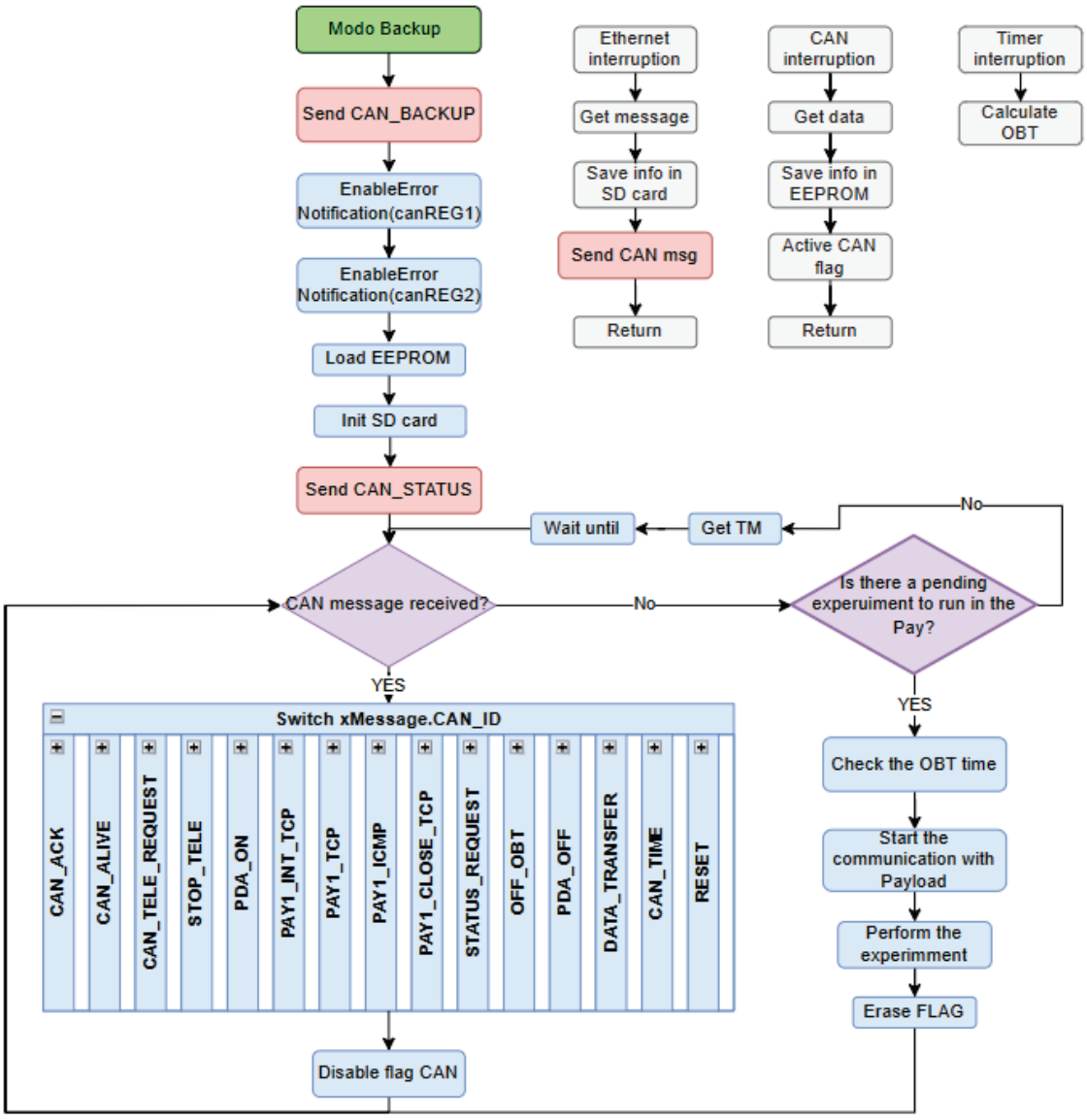


Figura 26: Arquitectura del Modo Backup

4.1.3.4. Modo Actualización del Firmware

Dentro del conjunto de modos de operación diseñados, uno de los aspectos cruciales es el *Modo Actualización del Firmware*. Este modo no sólo garantizamos la versatilidad de nuestro firmware, sino que también desempeña un papel fundamental en la capacidad de adaptación y mejora continua de nuestro satélite. En este modo, la POBC se encarga de grabar, en la posición de memoria en donde se encuentra el Modo Nominal, la nueva imagen binaria proporcionada a través del protocolo de comunicación CAN. Es importante destacar que el área de la memoria en donde se encuentra el Modo Actualización es una de las principales posiciones de la memoria flash y está protegida contra escritura, ya que es un modo fundamental para cambiar el funcionamiento de la POBC una vez que se encuentra en vuelo. Este modo no trabaja bajo un sistema operativo, sino que su implementación es a través de Bare Metal. Su desarrollo se puede observar en la figura 27, en donde podemos ver cómo es su estructura.

Al iniciar, lo primero que va a hacer es enviar un comando por CAN hacia la Estación Terrena, en el cual indica que ya se encuentre en Modo Actualización. Luego de esto, va a estar esperando el comando CAN_GET_ADDR con los datos de la posición de memoria a grabar y el largo de la nueva imagen binaria. Si la dirección y el tamaño de la nueva imagen son correctos, va a proceder a iniciar los bancos de memoria para grabar, y va a calcular la cantidad de mensajes que debe grabar. En caso de que la dirección de memoria o el largo de la imagen binaria sean incorrectos (posición de memoria fuera del rango de grabación, o largo de imagen más largo de lo permitido), va a enviar un comando CAN_NOK, con el dato incorrecto (sea la dirección o el tamaño de la imagen), y va a esperar por un nuevo comando CAN_GET_ADDR con nuevos datos. En caso de que este evento ocurra tres veces consecutivas, como medida de protección la POBC va a enviar un comando CAN_NOK con la última información inválida recibida, va a enviar un comando CAN_STATUS con la posición de memoria y el largo máximo de la imagen binaria disponibles, y va a proceder a saltar al Modo de Inicialización.

En caso de que sea valida la dirección y largo de imagen, va a enviar el comando CAN_GET_ADDR con los mismos datos recibidos, va a iniciar los bancos de memoria para grabar, y va a activar las funciones para manejar la memoria flash. Luego, va a entrar al switch case, en donde va a esperar por los datos a grabar. Dentro del switch case, los siguientes comandos van a ser admitidos: CAN_ALIVE, CAN_GET_DATA, CAN_BL_END, CAN_RESET y CAN_GET_STATUS.

Los datos a grabar van a ser recibidos dentro del comando CAN_GET_DATA. Este comando indica que dentro del mismo se encuentran los datos a grabar. Dentro de los 8 bytes disponibles del comando, los primeros 4 bytes van a indicar la posición de memoria a grabar, y los otros 4 bytes van a ser datos de la imagen binaria. El primer mensaje a grabar que espera la POBC, debe tener la posición P1 (primera posición de memoria grabar, pasada en el mensaje CAN_GET_ADDR) y los 4 bytes de la imagen binaria (Data1, haciendo referencia a los datos del primer mensaje a grabar) destinados a grabar en esa posición. Al recibirlo, comprueba que la posición de

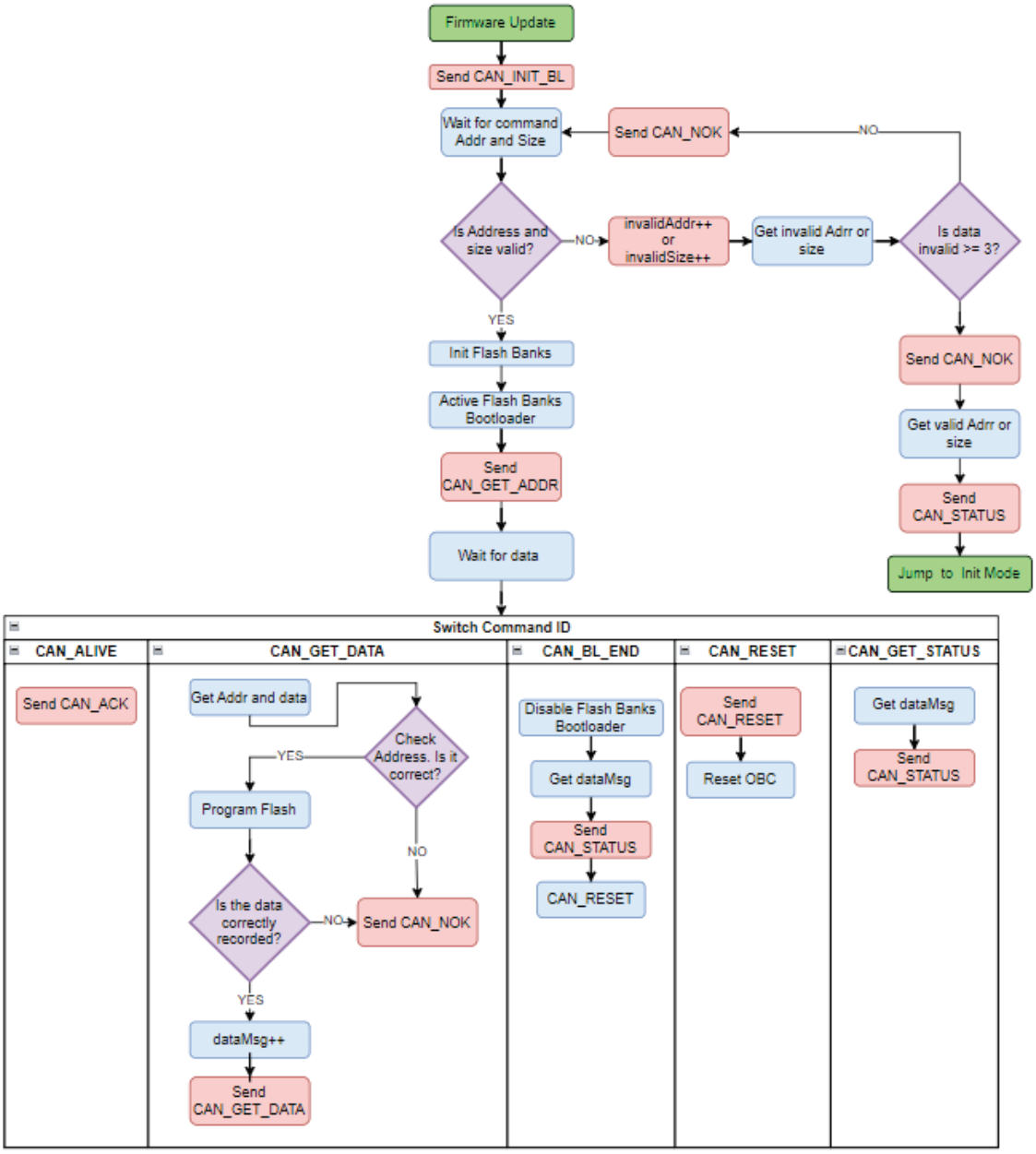


Figura 27: Arquitectura del firmware del Modo Actualización

memoria P1 sea correcta, y mediante las funciones de flash graba los 4 bytes de datos en esa posición. Luego, envía un comando con los mismos 8 bytes recibidos (posición de memoria e información grabada), de manera de que se corrobore en tierra los datos grabados. Para continuar con el siguiente comando a grabar, la POBC va a esperar grabar en la siguiente posición de memoria, es decir, P1+4B. Por lo que, el siguiente comando deberá estar compuesto en los primeros 4 bytes por la información de la

siguiente posición de memoria (P2 = P1+4B), y los correspondientes datos a grabar en esa posición (Data2). Si ese comando tiene una posición de memoria distinta a la esperada, la POBC va a enviar un mensaje de error (CAN_NOK), con los datos recibidos. Si es correcta, va a grabar los datos recibidos en la posición correspondiente, y va responder a modo de confirmación con un mensaje con la misma información recibida en ese segundo mensaje (P2+Data2). El proceso de actualización continúa hasta grabar toda la nueva imagen binaria.

El switch case está compuesto de la siguiente manera:

- CAN_ALIVE: La POBC con este comando aumenta el contador de comandos recibidos en la memoria EEPROM, y envia un ACK.
- CAN_GET_DATA: Comando utilizado para recibir los datos a grabar en la memoria flash.
- CAN_BL_END: Con este comando, se le indica a la POBC que la operación de actualización ya terminó. Al recibirlo, deshabilita los bancos de memoria para que no se pueda seguir escribiendo en ellos, y envía un comando CAN_STATUS con la información de la cantidad de comandos que fueron efectivamente grabados, proporcionada por dataMsg.
- CAN_RESET: Al recibir este comando, se procede al reseteo de la POBC, iniciando en el Modo Inicialización.
- CAN_GET_STATUS: Al recibir este comando, la POBC va a a enviar un comando CAN_STATUS con con la información de la cantidad de comandos que fueron efectivamente grabados hasta ese momento, proporcionada por dataMsg.

Los pasos a seguir para realizar la actualización son los siguientes:

1. Inicio del Modo Inicialización: Al encenderse la POBC, ingresa en lo que se denomina el 'Modo Inicialización' o System. En este estado, espera durante un tiempo determinado a que se le indique qué modo debe iniciar: Modo Nominal, Modo Backup o Modo Actualización del Firmware.
2. Ingreso al Modo actualización: Desde la Estación Terrena, se debe enviar el comando CAN_BOOT para indicarle a la POBC que ingrese al Modo de Actualización.
3. Modo Actualización: La POBC envía el comando CAN_INIT_BL para confirmarle a la Estación Terrena que ya ingresó en el Modo Actualización.
4. Confirmación de memoria y largo de imagen: Antes de enviar los datos del nuevo firmware, desde la Estación Terrena se debe enviar el comando CAN_GET_ADDR, indicando la posición de la memoria y cuánto espacio ocupará el nuevo programa.

5. Transferencia de la imagen binaria: Una vez confirmado el comando CAN_GET_ADDR, la POBC espera a que se le envíen los comandos con la imagen binaria del nuevo software. Esta imagen binaria se transfiere mediante el comando CAN_GET_DATA. Para la grabación en la memoria flash, la POBC recibe la imagen binaria de manera secuencial. Los primeros 4 bytes se graban en la posición más baja de memoria, seguidos de los segundos 4 bytes en la siguiente ubicación y así sucesivamente. Por cada comando recibido, la POBC va a responder con un comando que tenga los mismos datos, de manera de verificar la recepción correcta de cada uno de los paquetes recibidos.

6. Finalización de la actualización: Una vez que se ha transferido completamente la imagen binaria del nuevo firmware, la POBC debe reiniciarse con el comando RESET. Este reinicio asegura que el nuevo firmware se active correctamente.

7. Regreso al Modo Inicialización: Después de reiniciarse, la POBC regresa al Modo Inicialización. Desde este punto, se le puede indicar a la POBC que ejecute el Modo Nominal, que ahora está actualizado con la nueva versión del firmware.

El proceso de actualización del Firmware de la POBC garantiza que el Payload esté siempre equipado con la última versión del software necesario para cumplir con su misión. La capacidad de actualizar el firmware en órbita es esencial para adaptarse a cambios en los requisitos de la misión o para solucionar problemas inesperados, y garantiza el éxito continuo de la misión en el espacio.

En este capítulo explicamos cómo desarrollamos el firmware de la POBC, mostrando los diagramas funcionales de cada uno de los modos y las tareas que programamos. En el siguiente capítulo, explicaremos cómo procedimos con el desarrollo del firmware.

4.1.4. Desarrollo

Este proceso abarcó una serie de etapas, cada una de las cuales contribuyó al logro de un sistema de control confiable y eficiente.

En primer lugar, comenzamos con el proceso de familiarización y configuración de la placa de desarrollo utilizada para la POBC. Esta etapa nos permitió adquirir un profundo conocimiento de las capacidades y limitaciones de la plataforma, lo que resultó fundamental para el diseño y la implementación del firmware.

Una vez que aprendimos cómo programar la placa de desarrollo, nos enfocamos en la implementación del protocolo de comunicación CAN. Inicialmente, configuramos y probamos la comunicación entre los dos puertos CAN de la placa, nos aseguramos de que los mensajes se transmitan y reciban de manera efectiva. Esta fase de desarrollo proporcionó la base para la futura integración con otros subsistemas y sensores. Definimos todos los comandos CAN que necesitábamos, utilizando el ID del frame de CAN como nuestro identificador de mensajes. Para eso, definimos comandos personalizados utilizando ese ID, los cuales se pueden observar en la imagen 28.

Fuente	Nombre	ID	Largo	Comentario
GS to OBC	CAN_BOOT	0x03	0	Indicación para que ingresa el Modo Actualización
GS to OBC	CAN_ALIVE	0x06	0	Comando de Alive
GS to OBC	CAN_MAIN	0x060	0	Indicación para que ingrese al Modo Nominal
GS to OBC	CAN_BACKUP	0x30	0	Indicación para que ingrese al Modo Backup
GS to OBC	CAN_GET_ADDR	0x66	8	Posición y largo de BI para grabar en Modo Actualización
GS to OBC	CAN_GET_DATA	0x6C	8	Indica posición y datos a grabar
OBC to GS	CAN_ERR_HI	0x78	8	Error Hi
OBC to GS	CAN_ERR_MED	0x83	8	Error medio
OBC to GS	CAN_ERR_LOW	0x103	8	Error bajo
OBC to GS	DATA_TRANSFER	0x202	8	Transferencia de información
GS to OBC	PAY1_EXP0	0x204	8	Ejecución experimento 1
GS to OBC	PAY1_EXP1	0x208	8	Ejecución experimento 2
GS to OBC	PAY1_EXP2	0x210	8	Ejecución experimento 3
GS to OBC	PAY1_EXP3	0x220	8	Ejecución experimento 4
GS to OBC	STORAGE	0x384	8	Pedido de información SD
GS to OBC	PAY1_CLOSE_TCP	0x388	0	Pedido de cierre comunicación TCP
GS to OBC	PAY1_TCP	0x390	8	Mensaje TCP
GS to OBC	PAY1_INIT_TCP	0x3A0	0	Inicio de comunicación TCP
GS to OBC	PAY1_ICMP	0x5C1	0	Mensaje ICMP - Alive
GS to OBC	PDA_ON	0x61D	0	Prendido de la Carga Útil
GS to OBC	PDA_OFF	0x61E	0	Apagado de la Carga Útil
GS to OBC	STOP_TELE	0x67D	0	Pedido
GS to OBC	TELE_REQUEST	0X67E	8	Pedido de telemetría de la OBC
OBC to GS	CAN_STATUS	0x7F7	8	Mensaje de información de estado
GS to OBC	CAN_STATUS_REQUEST	0x7FB	0	Pedido de estado
GS to OBC	CAN_HARD_RESET	0x7FC	0	Reset por interrupción
GS to OBC	CAN_RESET	0x7FD	0	Reset por software
OBC to GS	CAN_ACK	0x7FE	0	Mensaje de confirmación de arribo
GS to OBC	CAN_TIME_REQUEST	0x602	0	Pedido de OBT
OBC to GS	CAN_TIME	0x604	8	Envío de OBT
GS to OBC	CAN_OFFSET_TIME	0x610	8	Offset para OBT

Figura 28: Tabla de comandos CAN

Posteriormente, nos dirigimos a la configuración y prueba de la comunicación Ethernet. Para este propósito, creamos un servidor TCP en una computadora, simulando la Carga Útil. Esto nos permitió verificar que los mensajes se transmitieran sin problemas desde la POBC hacia la Carga Útil a través de la conexión Ethernet y sentó las bases para la futura comunicación con la estación terrena.

Una vez que los protocolos de comunicación estaban en funcionamiento, nuestro enfoque de desarrollo prosiguió con la creación de diagramas funcionales detallados para cada componente y tarea de la POBC. Estos diagramas nos permitieron visualizar y comprender claramente las interacciones y dependencias entre las partes del sistema.

Una vez que finalizamos la fase de diseño basado en diagramas, procedimos a la etapa de implementación. Utilizamos estos diagramas funcionales como guía para la escritura de código en lenguaje C. Cada componente del sistema, ya sea una tarea, un módulo de comunicación o una función específica, lo programamos siguiendo fielmente la estructura y la lógica representada en los diagramas anteriormente mostrados.

Cada tarea la programamos cuidadosamente en C utilizando las funciones proporcionadas por FreeRTOS. Consideramos los aspectos críticos de la sincronización y la gestión de recursos para garantizar la solidez y la estabilidad del sistema. Además, utilizamos distintas funcionalidades de FreeRTOS como colas, semáforos y otros mecanismos de comunicación proporcionados para garantizar la coordinación efectiva entre las distintas tareas. En la imagen número 29 se puede ver la creación de cada una de las tareas previa ejecución del sistema operativo, junto con las colas que utilizan para comunicarse.

A medida que avanzábamos en la programación de las tareas y módulos individuales, fuimos testeando su comportamiento. Esto implicó garantizar que cada tarea funcionara de manera coherente y sin problemas con las demás tareas del sistema. También nos aseguramos de que los módulos de comunicación se comunicaran correctamente con las tareas correspondientes. En la imagen 30, podemos ver un fragmento del código de la vApp, dentro del switch case al recibir un mensaje a través del puerto CAN.

El desarrollo del firmware de la POBC fue un proceso gradual que abarcó desde la familiarización con la plataforma hasta la implementación de protocolos de comunicación, la integración en el sistema operativo y la verificación final. Esta metodología de desarrollo basada en diagramas funcionales y programación detallada nos resultó una base solida y efectiva para garantizar la calidad y fidelidad de nuestro proyecto.

Esta sección nos permitió adentrarnos en las tecnológicas del Firmware de la POBC y desvelar su importancia en el contexto de nuestro proyecto. A medida que cerramos este capítulo dedicado al Firmware de la POBC, es crucial recordar que este componente no es simplemente una entidad técnica, sino el cerebro detrás de la operación exitosa de nuestro Payload. El próximo capítulo explicaremos el PSC, donde veremos cómo estas dos piezas se relacionan para contribuir al funcionamiento del nanosatélite y éxito de la misión.

```
void vApp(void *pvParameters);
void vTimer(void *pvParameters);
void vTaskIDLE(void *pvParameters);
void vSense(void *pvParameters);

void RTOSFunction()
{
      sciSend(scilinREG, 15, (unsigned char *)"FIRMWARE\n\r");
      vSemaphoreCreateBinary(xSemaphoreTime);

      /*Create Queues*/
      xMainQ  = xQueueCreate(10, sizeof (struct xMessage));
      xSense  = xQueueCreate(10, sizeof (struct xMessage));
      xErrQ   = xQueueCreate(10, sizeof (struct xMessage));
      xComIn  = xQueueCreate(10, sizeof(int));
      xComRx  = xQueueCreate(10, sizeof(int));

      /* Create Tasks */
      xTaskCreate(vTimer,           "TIMER", configMINIMAL_STACK_SIZE, NULL, 5, &xTask1Handle);
      xTaskCreate(vApp,             "APP",   configMINIMAL_STACK_SIZE, NULL, 5, &xTask1Handle);
      xTaskCreate(vTaskIDLE,        "IDLE",  configMINIMAL_STACK_SIZE, NULL, 4, &xTask1Handle);
      /*Create Aperiodic Tasks*/
      xTaskCreate(vSense,           "HK",    configMINIMAL_STACK_SIZE, NULL, 4 , &xTask5Handle);
      xTaskCreate(vTCP,             "TCP",   configMINIMAL_STACK_SIZE, NULL, 4, &xTask4Handle);

      /* Start Scheduler */
      vTaskStartScheduler();
      /* Run forever */
      while(1);
}
```

Figura 29: Codificación de las tareas de la vApp

```
if(xMessage.Interface == CAN1Rx)
{
    commandsReceived++;

    switch (xMessage.CAN_ID)
    {
        /*************************************************************/
        case CAN_ALIVE:
            /*OBC sends ACK message */
            sciSend(scilinREG, 7, (unsigned char *)"ALIVE\n\r");
            /* Sends a CAN_ACK command though CAN 1 to the GS */
            xMessage.Interface = CAN1Tx;
            xMessage.CAN_ID = CAN_ACK;
            /* Puts the FOV in 1, to send the timer*/
            FOV = 1;
            PacketWrite(canREG1, xMessage.CAN_ID, &xMessage.Data, 0);
            break;
        /*************************************************************/
        case CAN_TELE_REQUEST:
            /* OBC starts to send  TM */
            sciSend(scilinREG, 7, (unsigned char *)"TL_ON\n\r");
            /* Set the Flag in true*/
            FLAG_TM = true;
           /* Sends a TELE_REQUEST command though CAN 1 to the GS */
            xMessage.Interface = CAN1Tx;
            xMessage.CAN_ID = CAN_TELE_REQUEST;
            PacketWrite(canREG1, xMessage.CAN_ID, &xMessage.Data, 0);
            break;
        /*************************************************************/
        case STOP_TELE:
            /*OBC stops to send TM to the HPCA*/
            sciSend(scilinREG, 8, (unsigned char *)"TL_OFF\n\r");
            /* Set Flag in false */
            FLAG_TM = false;
            /* Sends a  STOP_TELE command through CAN 1 to the GS*/
            xMessage.Interface = CAN1Tx;
            xMessage.CAN_ID = STOP_TELE;
            PacketWrite(canREG1, xMessage.CAN_ID, &xMessage.Data, 0);
            break;
        /*************************************************************/
```

Figura 30: Codificación de los comandos recibidos por CAN en vApp

4.2. Subsistema de Comunicación de Payload

En esta segunda sección nos enfocamos en el Subsistema de Comunicación de Payload. Se detalla en profundidad la arquitectura funcional, la elección de protocolos, el desarrollo del código y el presupuesto de enlace que conforman este subsistema.

4.2.1. Diseño funcional

En este capítulo, describimos en detalle el diseño funcional del PSC basado en LoRa para nuestro Payload, el cual es crucial para comprender cómo se planificó y estructuró la comunicación en nuestro sistema. En esta sección proporcionamos una visión general de los componentes y la arquitectura.

La topología básica del PSC está compuesta de dos módulos LoRa SX1278, los cuales actúan como transmisores y receptores, y una placa de control STM32F429 que supervisa y controla dichos módulos. La elección de utilizar dos módulos LoRa se basa en la necesidad de establecer una comunicación bidireccional estable y redundante.

Cada módulo LoRa se comunica con la placa de control STM32F429 a través de la interface SPI (Serial Peripheral Interface), lo que permite una transferencia rápida y segura de datos. Además, la placa de control está equipada con dos interfaces físicas de CAN, las cuales se conectan a dos módulos TJA1050, para establecer el uso del protocolo CAN y poder comunicarse correctamente con la POBC. Esta configuración proporciona una comunicación efectiva entre los subsistemas. En la imagen número 31 se muestra el diagrama completo de la comunicación entre la estación terrena y el Paylaod. Como puede observarse, para la arquitectura de la Estación Terrena utilizamos dos módulos Arduino Uno, los cuales conectamos por medio de SPI a los módulos LoRa SX1278. Utilizamos una computadora para programar las plaquetas de Arduino, y mediante la cual coordinamos el envío y recepción de paquetes de información hacia el Payload.

Algunas de las funcionalidades y características de nuestro subsistema son:

- Transmisión y recepción de datos: Cada módulo LoRa tiene la capacidad de transmitir y recibir información. Esta funcionalidad bidireccional permite que el satélite envíe datos a la estación terrestre y reciba comandos desde la misma. Además, al habilitar ambos módulos para transmitir y recibir, se logra una redundancia efectiva.

- Redundancia y tolerancia a fallas: La redundancia es fundamental en las misiones espaciales, donde las condiciones pueden ser extremas y las fallas son posibles. Habilitar ambos módulos LoRa para transmitir y recibir garantiza que, en caso de una falla en uno de los módulos, el otro pueda asumir el control y mantener la comunicación.

- Control de la Placa STM32F429: Esta placa desempeña un papel crucial al controlar los módulos LoRa. Gestiona la configuración de los módulos LoRa,

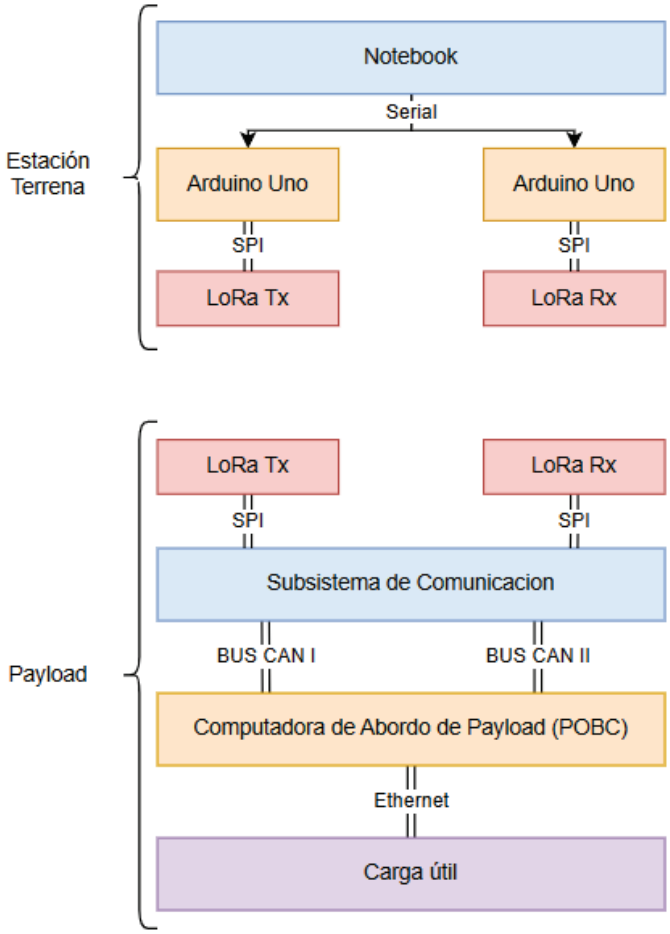


Figura 31: Comunicación entre la estación terrena y el Payload

incluyendo la frecuencia de operación, el ancho de banda y la velocidad de transmisión, supervisa el estado de la comunicación y coordina las transferencias de datos. Además, está equipada con interfaces CAN para comunicarse con la POBC.

- Redundancia de CAN: La redundancia se extiende a las interfaces CAN utilizadas para comunicarse con la POBC. La placa de control STM32F429 tiene la capacidad de enviar comandos y datos a través de ambos puertos CAN (CAN 1 y CAN 2) de forma redundante por ambos, lo que agrega una capa adicional de seguridad y tolerancia a fallas.

- Gestión de mensajes: La placa de control STM32F429 es responsable de gestionar los mensajes entre la POBC y la estación terrestre. Actúa como un puente de comunicación bidireccional, asegurando que los comandos y datos enviados

desde la Tierra lleguen a la POBC y que los datos científicos generados por el Payload se transmitan de manera precisa a la estación terrestre.

- Optimización de energía: En el entorno de un nanosatélite, donde los recursos son limitados, se ha prestado especial atención a la eficiencia energética. El PSC LoRa se configura para minimizar el consumo de energía, asegurando una operación sostenible y duradera.

La comunicación LoRa se integra de manera crítica en el funcionamiento general del Payload. Permite la transferencia de datos científicos, comandos de control y actualizaciones de software, lo que garantiza que el Payload pueda realizar sus tareas científicas y responder a las necesidades de la misión en tiempo real. El diseño funcional del Subsistema de Comunicación de Payload, LoRa también se centró en la adaptabilidad a condiciones cambiantes en el espacio. La capacidad de cambiar entre módulos LoRa y adaptar protocolos de comunicación garantizó que nuestro sistema responda ante eventuales problemas.

En esta sección vimos cómo es el diseño funcional del PSC. En la siguiente sección justificaremos la elección de los protocolos seleccionados.

4.2.2. Selección de protocolos

En esta sección, detallamos la selección de protocolos utilizados en el proyecto LoRa. Explicamos el por qué elegimos el protocolo LoRa y el resto de protocolos relacionados.

La elección del protocolo de comunicación en un nanosatélite fue una decisión importante que relacionó la eficacia de la misión y la capacidad del satélite para cumplir sus objetivos. En este contexto, la elección de utilizar la tecnología LoRa como protocolo de comunicación para nuestro nanosatélite se basó en una serie de razones sólidas y consideraciones fundamentales. A continuación, se presenta una justificación detallada de por qué LoRa se ha convertido en la elección predilecta para este proyecto espacial.

1. Rendimiento en largo alcance: Uno de los aspectos más destacados de la tecnología LoRa es su capacidad para ofrecer comunicaciones a larga distancia con un consumo de energía relativamente bajo. En el entorno del espacio, donde las distancias son largas y los recursos energéticos son limitados, esta característica se convierte en un factor determinante. La tecnología LoRa permite que nuestro Payload pueda comunicarse con la estación terrestre a distancias considerables, lo que es esencial para el éxito de la misión.

2. Eficiencia energética: Los nanosatélites, en su mayoría, dependen de fuentes de energía limitadas, como paneles solares y baterías. En este contexto, la eficiencia energética es esencial para garantizar una operación continua y sostenible. LoRa se destaca por su capacidad para transmitir datos de manera eficiente, lo que

significa que consume menos energía en comparación con otros protocolos de comunicación. Esta eficiencia energética contribuye directamente a la duración de la misión y a la confiabilidad operativa del nanosatélite.

3. Comunicación bidireccional: En muchas misiones espaciales, es fundamental tener la capacidad de establecer comunicaciones bidireccionales, es decir, tanto para enviar datos desde el satélite a la estación terrestre como para recibir comandos y actualizaciones desde la Tierra. La tecnología LoRa, con su capacidad de transmisión y recepción de datos, permite esta comunicación bidireccional. Además, como se mencionó anteriormente, habilitar ambos módulos LoRa para transmitir y recibir agrega una capa adicional de redundancia y tolerancia a fallas.

4. Resistencia a interferencias: En el entorno espacial, las interferencias electromagnéticas pueden ser un desafío significativo. LoRa se ha demostrado como una tecnología resistente a las interferencias, lo que garantiza una comunicación constante incluso en presencia de ruido y perturbaciones electromagnéticas. Esto es esencial para garantizar que el nanosatélite pueda mantener la comunicación en condiciones adversas.

5. Flexibilidad y adaptabilidad: La tecnología LoRa ofrece una flexibilidad notable en términos de configuración y adaptabilidad. Los parámetros de comunicación, como la frecuencia de operación, el ancho de banda y la velocidad de transmisión, pueden ajustarse según las necesidades específicas de la misión. Esto permite que el nanosatélite se adapte a diferentes condiciones y entornos, lo que es especialmente valioso en el espacio, donde las condiciones pueden cambiar rápidamente.

6. Costo: Además de sus ventajas técnicas, LoRa también se destaca por ser una tecnología económica. En comparación con algunas alternativas, la implementación de LoRa resulta en un costo total más bajo, lo que es beneficioso para proyectos con recursos financieros limitados.

7. Éxito en misiones espaciales anteriores: La elección de LoRa como protocolo de comunicación para Payload se basó en el éxito demostrado en misiones espaciales anteriores. Numerosas agencias espaciales y organizaciones han utilizado LoRa en misiones exitosas, lo que respalda su idoneidad y confiabilidad en el espacio. Algunas de las misiones espaciales exitosas son:

    a) FossaSat-1: También conocido como 'PocketQube 6' o 'FossaSat', es un nanosatélite de bolsillo desarrollado por Fossa Systems. Fue uno de los primeros nanosatélites en utilizar la tecnología LoRa para comunicaciones en órbita. FossaSat-1, lanzado en 2019, llevaba a bordo sensores y equipos de comunicación que utilizaban LoRa para transmitir datos y telemetría a

la Tierra. Esta misión demostró con éxito la viabilidad de LoRa en aplicaciones espaciales.

b) UPSat: Desarrollado por la Universidad de Patras y la Asociación de Usuarios de Software Libre de Grecia. Fue lanzado en 2017 y se convirtió en el primer nanosatélite de código abierto de Europa. UPSat utilizó la tecnología LoRa para la comunicación descendente y ascendente con la Tierra. La misión fue un éxito y contribuyó a validar la idoneidad de LoRa en entornos espaciales.

c) Prometheus: Desarrollado por la Universidad de Ciencias Aplicadas de Zurich, utilizó LoRa como parte de su sistema de comunicación. Fue lanzado en 2018 como parte de la misión QB50, que involucró a múltiples nanosatélites. Prometheus demostró con éxito la capacidad de LoRa para mantener la comunicación en la órbita terrestre baja.

d) GomX-4B: Es un nanosatélite de la Agencia Espacial Europea (ESA) lanzado en 2018 como parte de la misión 'Greenhouse Gases Observing Satellite' (GOSAT). GomX-4B incluyó un módulo de comunicación LoRa que se utilizó para probar la tecnología en el espacio. Esta misión contribuyó a la comprensión de la capacidad de LoRa para la comunicación espacial

Estos ejemplos destacan cómo varios nanosatélites han utilizado con éxito la tecnología LoRa como parte de sus sistemas de comunicación en órbita. Estas misiones han contribuido a la validación de LoRa como una opción confiable y efectiva para la comunicación en el entorno espacial, respaldando así su elección en proyectos de nanosatélites.

En cuanto a la elección de la placa STM32F429 como el controlador principal para los módulos LoRa en nuestro Payload nos basamos en una serie de consideraciones críticas que influyeron significativamente en nuestra decisión. En la figura número 32 se puede observar cómo es que está compuesto el PSC, con la placa de control, sus dos módulos de comunicación CAN y sus dos módulos LoRa.

El rendimiento confiable de la placa STM32F429 fue un factor clave. Está equipada con un procesador ARM Cortex-M4 de 32 bits, ofrece la capacidad de procesamiento necesaria para gestionar la comunicación de datos y garantizar una operación fluida y efectiva. Cuenta con interfaces SPI, fundamentales para la comunicación bidireccional con los módulos LoRa SX1278. La capacidad de comunicación eficiente y segura que proporcionan fue un requisito crucial para nuestras necesidades de transmisión y recepción de datos. La versatilidad y programabilidad de la plataforma STM32 fueron factores determinantes en nuestra elección. La capacidad de adaptar el firmware de la placa para satisfacer las necesidades específicas de nuestro nanosatélite y los requisitos de comunicación LoRa nos brindó la flexibilidad necesaria para lograr nuestros objetivos de comunicación de manera efectiva. La confiabilidad comprobada de la placa STM32F429 en aplicaciones críticas y misiones espaciales anteriores también pesó en nuestra decisión.

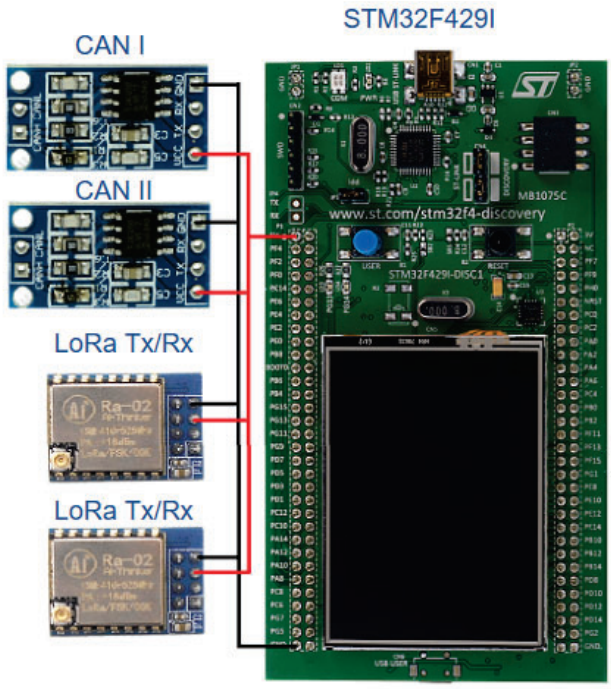


Figura 32: Prototipo del Subsistema de Comunicación de Payload

Saber que esta placa ha demostrado su rendimiento exitoso en el espacio proporcionó una mayor confianza en su capacidad para funcionar en el entorno riguroso de la órbita terrestre baja. Además, la amplia disponibilidad de documentación, recursos de desarrollo y soporte técnico simplificó en gran medida el proceso de diseño y desarrollo. Esto aceleró nuestra capacidad para integrar esta placa en nuestro sistema de comunicación LoRa de manera eficaz y dentro de los plazos previstos.

En conclusión, la elección de la tecnología LoRa como protocolo de comunicación para nuestro Payload se fundamenta en su probada eficacia en misiones espaciales previas, su capacidad para superar los desafíos inherentes a la comunicación en el espacio, y su versatilidad para satisfacer nuestras necesidades de transmisión de datos y comandos de manera efectiva y confiable.

4.2.3. Desarrollo

La fase de desarrollo desempeñó un papel fundamental en la implementación de la comunicación LoRa en nuestro Payload. En esta sección, abordaremos los aspectos técnicos clave, el proceso de programación en C y las pruebas exhaustivas realizadas durante esta fase para garantizar la solidez y eficiencia de la comunicación en el espacio.

En esta etapa, llevamos a cabo una serie de pasos críticos que incluyeron la selección de herramientas y lenguaje de programación, la arquitectura del software, la

interface de la placa STM32F429, el desarrollo del firmware y la creación del prototipo de Estación Terrena.

La codificación de la plaqueta de nuestro subsistema la hicimos en lenguaje de programación C. Este lenguaje demostró ser altamente virtuoso y capaz de gestionar los recursos limitados disponibles en el entorno de un nanosatélite, al tiempo que garantiza un consumo de energía mínimo. Además, C ofreció una versatilidad necesaria para implementar las funciones específicas requeridas para la comunicación LoRa.

Para el desarrollo, utilizamos herramientas de codificación estándar y ampliamente disponibles que facilitaron enormemente el proceso de programación y depuración. Una de estas herramientas fue STM32Cube IDE, la cual podemos observar en la figura 33. Este es un entorno de desarrollo integrado especialmente diseñado para microcontroladores STM32, el cual nos proporcionó las siguientes ventajas clave durante el proceso de desarrollo:

- **Integración de periféricos**: STM32Cube IDE incluyó una amplia biblioteca periférica que simplifica la configuración y el manejo de los periféricos de la placa STM32. Esto permitió una rápida integración de componentes como sensores y actuadores en el sistema.
- **Depuración avanzada**: El IDE ofrece un conjunto completo de herramientas de depuración, como el depurador integrado ST-Link, que permitió identificar y resolver rápidamente problemas de software.
- **Generación de código**: STM32Cube IDE proporciona la capacidad de generar código de inicio y configuración automáticamente, lo que aceleró el proceso de configuración inicial de la placa STM32.
- **Compatibilidad con plataformas**: El IDE es compatible con una amplia variedad de placas y microcontroladores STM32, lo que garantiza una mayor flexibilidad y opciones de desarrollo.

Comenzamos el desarrollo del Subsistema de Comunicación de Payload analizando y probando la placa de desarrollo seleccionada. Esto implicó sumergirnos en la comprensión de sus especificaciones técnicas y capacidades, así como familiarizarnos con la interface de programación, el STM32Cube IDE. Esta fase inicial nos permitió establecer una base sólida antes de adentrarnos en el desarrollo de protocolos de comunicación personalizados.

Uno de los componentes centrales de nuestro PSC es el protocolo CAN. Desarrollamos y codificamos este protocolo desde el principio, adaptándolo a nuestras necesidades específicas. Una vez que el protocolo CAN se encontraba en su etapa de desarrollo, pasamos a realizar distintas pruebas. Evaluamos minuciosamente su capacidad para transmitir y recibir comandos CAN de manera confiable, verificando que cumplía con los estándares de rendimiento y seguridad necesarios. La siguiente fase consistió en integrar nuestro PSC con la POBC del Payload. Esto no fue un proceso

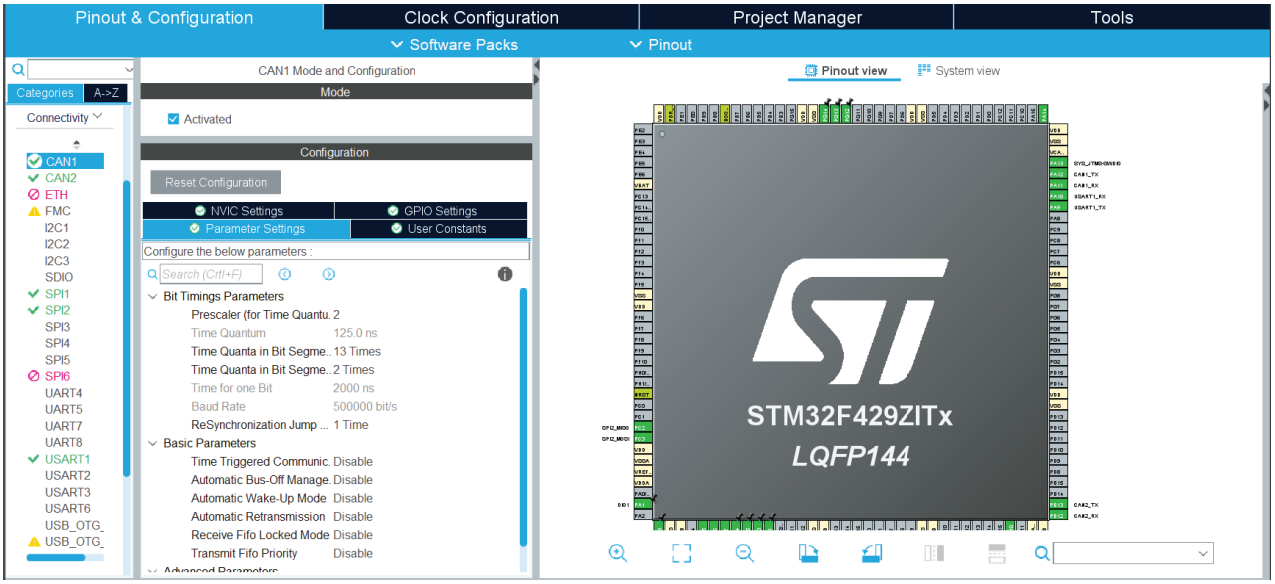


Figura 33: interface del STM32CubeIDE

trivial, ya que requería la sincronización perfecta entre dos sistemas independientes. Inicialmente, establecimos una comunicación entre la POBC y el PSC a través de un puerto. Esta prueba inicial nos permitió evaluar la capacidad de ambos sistemas para intercambiar información de manera efectiva, a distintas velocidades de transmisión.

Para garantizar la redundancia y la confiabilidad de las comunicaciones, continuamos desarrollando la comunicación con la POBC a través del segundo puerto. Este paso adicional fue crucial, ya que aseguraba que, en caso de una contingencia o un problema en uno de los puertos, el otro seguiría siendo funcional, permitiendo la continuidad de la misión. Durante esta fase, superamos desafíos técnicos y ajustamos cuidadosamente las configuraciones para garantizar una comunicación sin problemas.

Una vez que logramos una comunicación efectiva entre la POBC y nuestro PSC a través de los puertos CAN, avanzamos hacia la implementación y codificación de los módulos LoRa en nuestro kit de desarrollo. Esta fase del desarrollo implicó una serie de pasos cruciales para habilitar la comunicación a larga distancia tanto con la estación terrena como entre los propios módulos LoRa.

Inicialmente, nos concentramos en la configuración y el control de los módulos LoRa en nuestro kit de desarrollo. Esto incluyó la adaptación de los puertos SPI para que fueran capaces de gestionar estos módulos específicos. Este proceso nos permitió establecer una base sólida para la comunicación con los módulos LoRa y asegurarnos de que pudieran operar de manera coherente y predecible en el entorno del nanosatélite.

Paralelamente, desarrollamos la comunicación en nuestra estación terrena. Comenzamos con pruebas de recepción, lo que nos permitió verificar si los comandos enviados desde la estación terrena podían ser captados de manera confiable por nues-

tro PSC. Esta fase inicial de pruebas fue fundamental para garantizar que nuestra estación terrena pudiera interactuar de manera efectiva con el Payload.

La etapa siguiente involucró la transmisión de comandos desde el Payload hacia la estación terrena. Esta parte del proceso resultó ser especialmente desafiante y requirió ajustes y optimizaciones. Superamos estos obstáculos mediante pruebas rigurosas y refinamientos en la codificación y la configuración. Y una vez que logramos una comunicación confiable con un módulo LoRa, duplicamos este proceso para configurar y probar el segundo módulo LoRa. Esto fue importante para garantizar una comunicación redundante y respaldar la continuidad de la misión en caso de problemas técnicos. Repetimos las mismas pruebas de recepción y transmisión para el segundo módulo, asegurándonos de que ambos fueran completamente funcionales y complementarios.

Con el exitoso establecimiento de comunicaciones a través de los puertos CAN y los módulos LoRa de nuestra plaqueta, avanzamos hacia la creación de un sistema de comunicación completamente integrado y funcional para nuestro prototipo de Payload. Esta fase crucial del proyecto involucró la programación y las pruebas del flujo completo de datos, desde la Estación Terrena hasta la POBC y viceversa.

Comenzamos elaborando un diagrama en bloques que representaba de manera precisa la arquitectura de nuestro sistema de comunicación, el cual puede observarse en la figura 34. Este diagrama sirvió como guía durante la etapa de programación. La programación de esta estructura implicó la creación de un código robusto y eficiente que permitiera la transferencia bidireccional de información entre la Estación Terrena, el PSC y la POBC. Además de los comandos que se transmiten desde la POBC, incorporamos un beacon, propio del PSC. Este beacon se envía una vez cada un segundo, y tiene el objetivo de avisar a la estación terrena que el Payload se encuentra dentro de su zona de cobertura. El beacon transmite un identificador, el cual es de utilidad para identificar a nuestro nanosatélite. El funcionamiento detallado del diagrama en bloques es el siguiente:

1. Configuración del Sistema (System Configuration):

    a) La función HAL_Init() inicializa los periféricos del microcontrolador y la interface Flash.

    b) La función SystemClock_Config() configura el reloj del sistema.

2. Inicialización de Periféricos (Peripheral Initialization):

    a) Las funciones MX_GPIO_Init(), MX_CAN1_Init(), MX_CAN2_Init() inicializan los periféricos GPIO y los puertos CAN.

    b) Y las funciones MX_SPI1_Init(), y MX_SPI2_Init() inicializan los periféricos para la comunicación SPI.

3. Inicialización de LoRa (LoRa Initialization):

a) Se configuran los parámetros de un dispositivo LoRa llamado myLoRaTx para la transmisión y myLoRaRx para la recepción.

b) Se establecen valores como la frecuencia, el factor de esparcimiento, el ancho de banda, la tasa de corrección de errores, la potencia de transmisión, etc., para la comunicación LoRa.

c) Se verifica si la inicialización de LoRa es exitosa.

4. Bucle Principal (Main Loop):

   a) El bucle principal (while(1)) se ejecuta continuamente y contiene la lógica principal del programa.

   b) Realiza la recepción de datos LoRa utilizando la función LoRa_receive() y procesa los datos recibidos.

   c) Realiza la recepción de datos CAN utilizando HAL_CAN_GetRxMessage() y procesa los mensajes CAN recibidos, respondiendo a ellos según corresponda.

   d) Transmisión del beacon, mediante una interrupción por tiempo cada un segundo.

5. Manejo de Interrupciones (Interrupt Handling):

6. Bucle Principal (Main Loop):

   a) El programa configura la interrupción EXTI0 para manejar eventos de interrupción externa en el pin GPIO 0.

Con el objetivo de facilitar la supervisión y la corroboración de los datos transmitidos y recibidos en la Estación Terrena, decidimos llevar a cabo un desarrollo adicional: la creación de una interface gráfica personalizada. Esta interface gráfica nos permitiría visualizar de manera clara y concisa tanto los parámetros de telemetría generados por la POBC como los valores de señal recibidos a través de la tecnología LoRa. Los datos de telemetría proporcionados por la POBC, que abarcaban aspectos esenciales como el estado del sistema, la temperatura interna y otros parámetros críticos, se presentaron de manera organizada y accesible en la interface. Además, la interface gráfica también proporcionaba una representación visual de los valores de señal recibidos a través de la comunicación LoRa. Esta función resultó especialmente valiosa para el monitoreo de la calidad de la comunicación y la potencia de la señal durante los testeos.

Una vez finalizada la programación, procedimos a realizar pruebas integrales de comunicación. Estas pruebas comprendieron el envío de comandos desde la Estación Terrena hacia la POBC, siguiendo la ruta a través del PSC. Además, evaluamos la funcionalidad inversa, es decir, el flujo de datos desde la POBC hasta la Estación

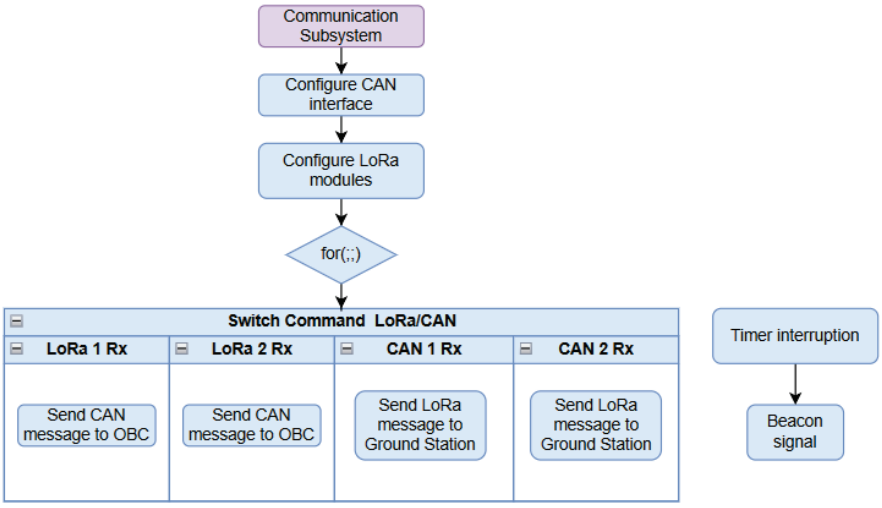


Figura 34: Arquitectura del firmware del Subsistema de Comunicación de Payload

Terrena. Este proceso de prueba nos permitió verificar que todos los componentes estaban sincronizados y operativos de manera coherente.

La realización exitosa de estas pruebas de comunicación validó la eficacia de nuestro sistema integrado. Cada mensaje enviado desde la Estación Terrena fue recibido de manera confiable por la POBC, y las respuestas y datos generados por la POBC fueron transmitidos de regreso a la Estación Terrena de manera precisa.

El desarrollo de nuestro sistema de comunicación LoRa se basó en una sólida investigación y decisiones estratégicas. Ahora procederemos al cálculo del presupuesto de enlace para la comunicación LoRa.

4.2.4. Presupuesto de enlace

El presupuesto de enlace de comunicación es un componente fundamental en la planificación y operación de un sistema de comunicación satelital. En el contexto de nuestro proyecto, este presupuesto se encargó de determinar la viabilidad y la calidad de la comunicación entre nuestro satélite y la estación terrestre en la Tierra.

La capacidad de mantener una comunicación fiable a lo largo de la misión del nanosatélite es esencial para el éxito de la operación. Este presupuesto tomó en consideración una serie de factores cruciales, como la distancia entre el satélite y la estación terrestre, la potencia transmitida, las ganancias de las antenas, las pérdidas del sistema y el nivel de ruido.

En esta sección, presentamos en detalle los cálculos y los resultados obtenidos a partir del presupuesto de enlace de comunicación. Estos resultados son esenciales para garantizar una comunicación eficaz y fiable, lo que a su vez respalda el cumplimiento de los objetivos de nuestra misión.

Para garantizar una comunicación estable y segura, fue esencial calcular la potencia recibida (Prx). Esto nos permitió determinar si la señal transmitida desde el Payload era lo suficientemente fuerte para ser detectada y decodificada de manera efectiva. El cálculo de Prx se basó en la siguiente ecuación del presupuesto de enlace:

$$Prx = Pt + Gt - Lc - Lfs - Gr \tag{5}$$

Donde:
- Ptx es la potencia de transmisión.
- Gtx es la ganancia de la antena de transmisión en la estación terrena.
- L representa las pérdidas de los conectores y cables.
- Lfs es la pérdida en espacio libre.
- Grx es la ganancia de la antena receptora en el satélite.

Considerando que LoRa ofrece la flexibilidad de operar en tres frecuencias distintas, realizamos el cálculo de la potencia recibida en las siguientes bandas: 868 MHz, 915 MHz y 433 MHz. Los resultados de nuestros cálculos de Prx se presentan en las tablas números 35 para la frecuencia de 433 MHz, en la número 36 para la frecuencia de 868 MHz y en la tabla número 37 para la frecuencia de 915 MHz. Estos resultados fueron fundamentales para determinar la viabilidad de la comunicación entre el Payload y la estación terrestre en diferentes escenarios.

433							
Distancia [Km]	Ptx [dBm)]	L [dB]	Gtx [dBi]	FSL [dB]	Grx [dBi]	L [dB]	Prx [dBm]
0,1	20	1	1,2	65,18	1,2	1	-44,78
1	20	1	1,2	85,18	1,2	1	-64,78
100	20	1	1,2	125,18	1,2	1	-104,78
500	20	1	1,2	139,16	1,2	1	-118,76
1000	20	1	1,2	145,18	1,2	1	-124,78
2450	20	1	1,2	152,96	1,2	1	-132,56
2750	20	1	1,2	153,97	1,2	1	-133,57

Figura 35: Cálculo de la potencia recibida en 433 MHz

Fue evidente al observar las tablas de potencia recibida, que la frecuencia de 433 MHz exhibe la mayor potencia recibida en comparación con las otras dos frecuencias (868 MHz y 915 MHz). Esta diferencia se atribuye a las pérdidas en el espacio libre, donde la menor frecuencia experimenta una menor atenuación de la señal a lo largo de distancias largas, lo que resulta en una mayor potencia recibida por nuestro PSC. Este fenómeno es coherente con los principios de propagación de radiofrecuencia

Otro aspecto esencial en la planificación de nuestro sistema de comunicación LoRa fue la determinación de las tasas de transferencia de datos, o tasas de bits, que podíamos lograr. Esto fue fundamental para garantizar una comunicación eficiente con nuestro Payload. En este contexto, calculamos las tasas de bits para distintos anchos de banda, factores de propagación (SF) y una tasa de codificación (CR) de

868							
Distancia [Km]	Ptx [dBm)]	L [dB]	Gtx [dBi]	FSL [dB]	Grx [dBi]	L [dB]	Prx [dBm]
0,1	20	1	3	71,22	3	1	-47,22
1	20	1	3	91,22	3	1	-67,22
100	20	1	3	131,22	3	1	-107,22
500	20	1	3	145,20	3	1	-121,20
1000	20	1	3	151,22	3	1	-127,22
2450	20	1	3	159,00	3	1	-135,00
2750	20	1	3	160,01	3	1	-136,01

Figura 36: Cálculo de la potencia recibida en 868 MHz

915							
Distancia [Km]	Ptx [dBm)]	L [dB]	Gtx [dBi]	FSL [dB]	Grx [dBi]	L [dB]	Prx [dBm]
0,1	20	1	3	71,68	3	1	-47,68
1	20	1	3	91,68	3	1	-67,68
100	20	1	3	131,68	3	1	-107,68
500	20	1	3	145,66	3	1	-121,66
1000	20	1	3	151,68	3	1	-127,68
2450	20	1	3	159,46	3	1	-135,46
2750	20	1	3	160,47	3	1	-136,47

Figura 37: Cálculo de la potencia recibida en 915 MHz

4/5. Estos cálculos nos permitió evaluar y optimizar la capacidad de transferencia de datos de nuestro sistema de comunicación.

Se calculó utilizando la siguiente fórmula:

$$Rb = \frac{SF \cdot \left(\frac{4}{4+CR}\right)}{\frac{2^{SF}}{BW}} \cdot 1000 \tag{6}$$

En donde:

- Rb es el Bit Rate en bits por segundos (bps)
- SF es el factor de propagación (Spreading Factor), que determina la duración del símbolo.
- CR es la tasa de codificación (Coding Rate), expresada como una fracción, por ejemplo, 4/5.
- BW es el ancho de banda (Bandwidth) de la señal en hertz (Hz). Esta fórmula nos permite calcular el bit rate en función de varios parámetros, como el Spreading Factor, la tasa de codificación y el ancho de banda, lo que nos permite optimizar la comunicación LoRa para nuestras necesidades específicas.

Después de calcular el bit rate para diversas configuraciones, el siguiente paso crucial en nuestro análisis de enlace fue determinar la sensibilidad del receptor (S).

La sensibilidad del receptor es el nivel mínimo de potencia de señal que el receptor puede detectar y decodificar de manera efectiva.

Calculamos la sensibilidad del receptor para cada combinación de ancho de banda y SF que podríamos utilizar en nuestro sistema de comunicación LoRa. Este es un parámetro crítico en la evaluación de la capacidad de un sistema de comunicación para recibir señales débiles en presencia de ruido. Es el nivel mínimo de potencia de la señal que el receptor puede detectar y decodificar de manera confiable. Esta sensibilidad la calculamos utilizando la siguiente ecuación:

$$S = -174 + 10 \cdot \log_{10}(BW) + NF + SNR$$

Donde:

- S es la sensibilidad del receptor en dBm, que es una medida de potencia.
- BW representa el ancho de banda del receptor en Hz. Este valor se refiere al rango de frecuencias que el receptor es capaz de recibir y procesar.
- NF es el factor de ruido (Noise Figure) del receptor en dB. El factor de ruido se relaciona con la cantidad de ruido adicional que el receptor introduce en el sistema. Cuanto menor sea el valor de NF, mejor será la sensibilidad del receptor.
- SNR es la relación señal-ruido (Signal to Noise Ratio) deseada en dB. Esta es la relación entre la potencia de la señal deseada y la potencia del ruido en el sistema. Cuanto mayor sea el valor de SNR, mejor será la capacidad del receptor para distinguir la señal de los niveles de ruido.

Es importante destacar que tanto el factor de ruido (NF) como la relación señal-ruido (SNR) son parámetros críticos que dependen de la configuración y las características del receptor. Estos datos los obtuvimos de la hoja de datos del módulo SX1278.

Utilizamos la información anteriormente calculada, que incluía la potencia recibida y la sensibilidad del receptor, para evaluar la robustez y la viabilidad de la comunicación en nuestro sistema. Uno de los aspectos críticos en esta evaluación fue el cálculo del margen de enlace. El margen de enlace representó la diferencia entre la potencia recibida y la sensibilidad del receptor, la cual calculamos para el peor escenario posible, es decir, la mayor distancia entre la Estación Terrena y el Payload.

Saber si existía un margen de enlace adecuado era esencial, ya que determinaba la capacidad del sistema para tolerar la atenuación de la señal debida a la propagación a larga distancia y otros factores que podían degradar la calidad de la señal.

Si los cálculos mostraban que no había margen de enlace, esto indicaba que no sería posible establecer la comunicación con el Payload en esa distancia específica utilizando la configuración actual. En tales casos, se habrían requerido ajustes en los parámetros de comunicación, como aumentar la potencia de transmisión, reducir la distancia o considerar un cambio en la configuración de la antena.

Por otro lado, un margen de enlace positivo proporcionaba una mayor confianza en la capacidad del sistema para mantener una comunicación precisa en condiciones adversas. Este margen también permitía anticipar la calidad de la comunicación en

diversas distancias y condiciones atmosféricas.

Al final de esta evaluación, generamos tablas en las que los valores con fondo de color violeta claro representaban los casos en los que el margen era positivo. Estos valores se convirtieron en referencias clave para establecer la comunicación con el Payload, ya que indicaban las distancias y las configuraciones de comunicación en las que podíamos llevar a cabo el proyecto.

En las tablas 38, 39 y 40 se pueden ver los resultados obtenidos para los distintos anchos de banda:

BW = 125 KHz									
SF	Chips/ Symbol	SNR min [dB]	NF [dB]	CR = 4/5	Bit Rate [bps]	Sensibilidad [dBm]	Margen [433 MHz]	Margen [868 MHz]	Margen [915 MHz]
6	64	-5,00	6	0,8	9.766	-122,03	-7,94	-13,98	-14,43
7	128	-7,50	6	0,8	5.697	-124,53	-5,44	-11,48	-11,93
8	256	-10,00	6	0,8	3.255	-127,03	-2,94	-8,98	-9,43
9	512	-12,50	6	0,8	1.831	-129,53	-0,44	-6,48	-6,93
10	1024	-15,00	6	0,8	1.017	-132,03	2,06	-3,98	-4,43
11	2048	-17,50	6	0,8	559	-134,53	4,56	-1,48	-1,93
12	4098	-20,00	6	0,8	305	-137,03	7,06	1,02	0,57

Figura 38: Cálculo de la sensibilidad y márgenes disponibles para un BW de 125 KHz

BW = 250 KHz									
SF	Chips/ Symbol	SNR min [dB]	NF [dB]	CR = 4/5	Bit Rate [bps]	Sensibilidad [dBm]	Margen [433 MHz]	Margen [868 MHz]	Margen [915 MHz]
6	64	-5,00	6	0,8	19.531	-119,02	-10,95	-16,99	-17,44
7	128	-7,50	6	0,8	11.393	-121,52	-8,45	-14,49	-14,94
8	256	-10,00	6	0,8	6.510	-124,02	-5,95	-11,99	-12,44
9	512	-12,50	6	0,8	3.662	-126,52	-3,45	-9,49	-9,94
10	1024	-15,00	6	0,8	2.035	-129,02	-0,95	-6,99	-7,44
11	2048	-17,50	6	0,8	1.119	-131,52	1,55	-4,49	-4,94
12	4098	-20,00	6	0,8	610	-134,02	4,05	-1,99	-2,44

Figura 39: Cálculo de la sensibilidad y márgenes disponibles para un BW de 250 KHz

El margen de enlace nos proporcionó información valiosa sobre qué configuraciones eran viables para nuestro sistema. Identificamos las combinaciones de BW y SF que ofrecían un margen de enlace suficiente para garantizar una comunicación sólida, incluso en condiciones desafiantes. Estas configuraciones nos permitieron determinar en qué frecuencia podríamos operar de manera efectiva, qué combinación de BW y

BW = 500 KHz									
SF	Chips/ Symbol	SNR min [dB]	NF [dB]	CR = 4/5	Bit Rate [bps]	Sensibilidad [dBm]	Margen [433 MHz]	Margen [868 MHz]	Margen [915 MHz]
6	64	-5,00	6	0,8	39.063	-116,01	-13,96	-20,00	-20,45
7	128	-7,50	6	0,8	22.786	-118,51	-11,46	-17,50	-17,95
8	256	-10,00	6	0,8	13.021	-121,01	-8,96	-15,00	-15,45
9	512	-12,50	6	0,8	7.324	-123,51	-6,46	-12,50	-12,95
10	1024	-15,00	6	0,8	4.069	-126,01	-3,96	-10,00	-10,45
11	2048	-17,50	6	0,8	2.238	-128,51	-1,46	-7,50	-7,95
12	4098	-20,00	6	0,8	1.221	-131,01	1,04	-5,00	-5,45

Figura 40: Cálculo de la sensibilidad y márgenes disponibles para un BW de 500 KHz

SF sería óptima y las velocidades de transmisión que podríamos alcanzar en nuestro Payload.

Después de analizar los cálculos de margen de enlace, determinamos la configuración óptima para nuestras comunicaciones con el Payload. Esta elección se basó en la búsqueda de un equilibrio entre el alcance necesario y la calidad de la señal.

En última instancia, decidimos operar en la frecuencia de 433 MHz, con un ancho de banda (BW) de 125 kHz y un factor de propagación (SF) de 11. Esta decisión se basó en varios factores clave:

1. Margen de enlace sólido: Con esta configuración, logramos un margen de enlace de 4.56 dB, lo que garantizó una comunicación robusta y precisa con nuestro Payload en órbita.

2. Disponibilidad del espectro: La frecuencia de 433 MHz ofrecía ventajas en términos de disponibilidad del espectro radioeléctrico para nuestras comunicaciones. Esta banda estaba menos congestionada y era menos propensa a interferencias, lo que mejoraba la integridad de nuestras señales.

3. Alcance adecuado: La elección de un SF de 11 y un BW de 125 kHz nos permitía alcanzar distancias considerables sin sacrificar la eficiencia del canal de comunicación. Esto era esencial para garantizar que el Payload pudiera comunicarse de manera confiable a través de su órbita.

4. Ahorro de energía: El SF de 11 y el BW de 125 kHz también contribuyeron a minimizar el consumo de energía del sistema, una consideración crítica para un Payload donde los recursos energéticos son limitados.

5. Capacidad de penetración: La frecuencia de 433 MHz también tenía una buena capacidad de penetración a través de obstáculos, lo que resultaba valioso en escenarios donde la línea de visión directa con la estación terrestre no siempre se mantenía.

Esta configuración se convirtió clave para nuestro propósito, ofreciendo un compromiso sólido entre rendimiento y eficiencia energética. A lo largo de la misión, esta elección demostró ser acertada, permitiéndonos mantener comunicaciones confiables y de alta calidad con nuestro Payload en órbita.

Es importante destacar que estos cálculos se basaron en los parámetros específicos de nuestro sistema de comunicación y las condiciones de la misión. Los valores reales pueden variar según la ubicación orbital del nanosatélite y las condiciones atmosféricas.

Concluimos esta sección dedicada al Subsistema de Comunicación de Payload, en la cual hemos explorado en detalle su diseño funcional, la elección de protocolos, el desarrollo experto de código y el minucioso análisis del presupuesto de enlace.

Habiendo explorado con profundidad las dos secciones fundamentales de nuestro proyecto, el Firmware de la POBC y el PSC, hemos establecido las bases de este proyecto tecnológico. Desde la arquitectura hasta el desarrollo, desde el diseño funcional hasta la selección de protocolos, cada detalle ha sido meticulosamente examinado. En el próximo capítulo vamos a adentrarnos en los testeos que realizamos en ambos subsistemas y los resultados que obtuvimos.

5. Testeo

Esta sección nos adentramos en las pruebas fundamentales que garantizan la robustez y confiabilidad de nuestro Payload. Está sección está dividida en dos secciones principales, en donde testearemos primero el firmware de la POBC y segundo los protocolos de comunicación, evaluando su rendimiento en los kit de desarrollo utilizados.

5.1. Firmware de la POBC

Nuestro firmware de la POBC cuenta con cuatro modos de funcionamiento distintos, los cuales explicamos en la sección anterior. Cada uno de los distintos modos tiene distintas especificaciones y características, por lo cual su testeo va a depender de cada uno de ellos. Utilizamos distintas herramientas para poder validar los distintos modos, y algunas funcionalidades tanto de nuestro kit de desarrollo como del IDE utilizado para programarlo. El IDE utilizado para programar nuestro kit de desarrollo de la POBC fue el Code Composer Studio, en la versión 10. A continuación, explicaremos los testeos en cada uno de los modos.

- Testeo del Modo de Inicialización: Para este modo, corroboramos la correcta ejecución del mismo, y que cumpla con los saltos hacia los distintos modos de funcionamiento.
- Testeo del Modo Nominal: En este caso, utilizamos una herramienta para poder testear el funcionamiento del Sistema Operativo. Elegimos una tarea, y corroboramos que funcione de acuerdo a lo planificado según los diagramas previamente diseñados.
- Testeo del Modo Backup: Comprobamos su ejecución desde el Modo de Inicialización, y corroboramos que cumplía con todas las funciones programadas.
- Testeo del Modo Actualización: Para este testeo, realizamos un dispositivo (gadget) para corroborar que nuestro programa de actualización funcione, grabe y lo ejecute correctamente.

5.1.1. Testeo del Modo de Inicialización

Para la verificación y prueba de este modo operativo, no fue necesario recurrir a la utilización de software especializado adicional. Nos centramos en la validación de la ejecución del firmware, asegurándonos de que se cumpla el salto de memoria al ejecutar el Modo Nominal y la correcta operación de los diferentes modos funcionales. En la imagen número 41 pudimos observar en la consola de comandos, la correcta inicialización de este modo. Realizamos distintas comprobaciones para garantizar que cada uno de los modos operativos se ejecutaron de forma correcta, partiendo siempre desde el Modo de Inicialización. Verificamos que el programa funcione tal cual como

lo habíamos pensado, iniciando todas las funciones y periféricos, y transmitiendo los mensajes de ALIVE indicando que la POBC estaba encendida. Se probaron todos los comandos CAN que se podrían recibir y transmitir desde este modo.

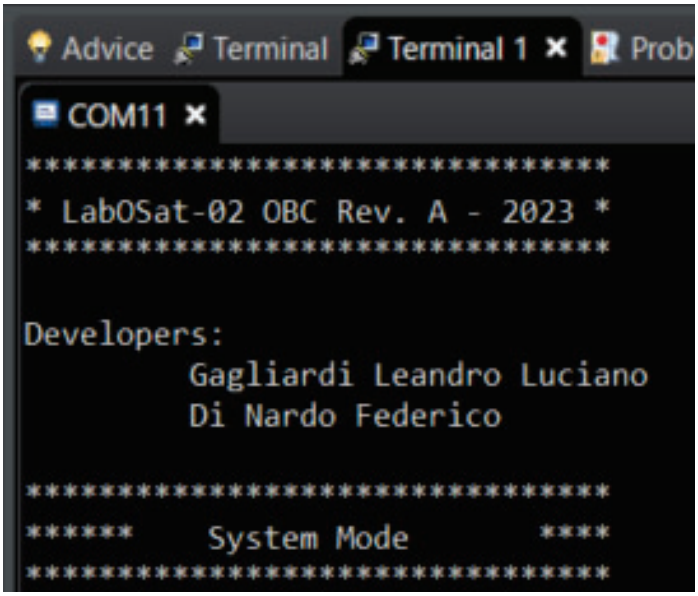


Figura 41: Mensajes en consola al comenzar el Modo Inicialización

5.1.2. Testeo del Modo Nominal

Uno de los grandes desafíos que abordamos en nuestro proyecto fue el testeo de nuestro Modo Nominal, ya que el mismo esta compuesto por un Sistema Operativo, y la ejecución de las tareas y su funcionamiento no es algo observable a simple vista, sino que requiere de distintos métodos para poder comprobar su funcionamiento.

Para comenzar y poder corroborar si nuestras tareas se estaban ejecutando correctamente, decidimos utilizar un testeo visual, y así poder corroborar que las tareas se estén ejecutando. Para eso, decidimos utilizar los LEDs integrados que tiene nuestro Kit de desarrollo. De esta manera, asociamos cada una de las tareas a un LED específico, el cual se mantenía encendido durante todo el tiempo que la tarea estaba en funcionamiento. De esta forma, nos aseguramos de que las tareas estaban ejecutándose. Este enfoque visual inicial nos permitió identificar y corregir problemas evidentes y verificar el inicio y la finalización de las tareas. Sin embargo, es importante destacar que este método no nos brindó información detallada sobre los tiempos de ejecución de cada tarea.

Para continuar con nuestro análisis de depuración, decidimos utilizar la consola de comandos que nos proporciona nuestro IDE de Texas, con el cual programamos nuestro kit de desarrollo. Esto nos permitió detectar ciertos problemas en las ejecuciones

de algunas tareas, y poder hacer una corrección más precisa de nuestro sistema. En la imagen número 42 podemos ver la ejecución de las distintas tareas del sistema. Implementamos mensajes de depuración en diferentes etapas de cada tarea programada, lo que facilitó la observación detallada del flujo de ejecución. Este método nos permitió asegurarnos que cada tarea completara correctamente sus distintas etapas y ejecutara todas las líneas de código esperadas. De esta manera, nos aseguramos de la completa ejecución de cada una de las tareas que programamos, y nos sirvió para poder detectar y corregir ciertos problemas que había en la ejecución de algunas partes. Si bien es una forma de poder garantizar que se ejecuten todas las líneas de código de nuestras tareas, no es un método para poder terminar de validar nuestro sistema, ya que no nos ofrece información precisa sobre tiempos de ejecución y consumos.

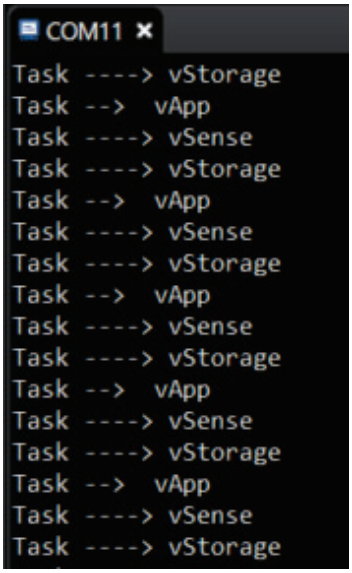


Figura 42: Análisis de la ejecución de las tareas por la consola de comandos

Por lo que, una vez que nos aseguramos que nuestro Sistema Operativo estaba funcionando, decidimos hacer un testeo más profundo, para poder validarlo correctamente.

Lo que hicimos fue corroborar que todas las tareas se crearan y se ejecutaran correctamente, que el 'Scheduler' asigne a las tareas las prioridades correspondientes, que se crearan y ejecutaran los semáforos y las colas que programamos para poder comunicar a las tareas, y que las interrupciones recibidas no interfirieran en la ejecución de las tareas. Para lograr ese nivel de validación, utilizamos un software especial de FreeRTOS llamado Tracealyzer. Este software es una herramienta de análisis y visualización de sistemas en tiempo real, utilizada para depurar y optimizar dichos sistemas. Esto nos permitió observar y analizar el comportamiento de nuestro sistema en tiempo real, lo cual fue algo fundamental para poder testear y validar nuestro sistema. Desde Tracealyzer nos han otorgado una licencia de uso, por ser parte de un

proyecto correspondiente a la Universidad de San Martín. Algunas de las funcionalidades que ofrece son:

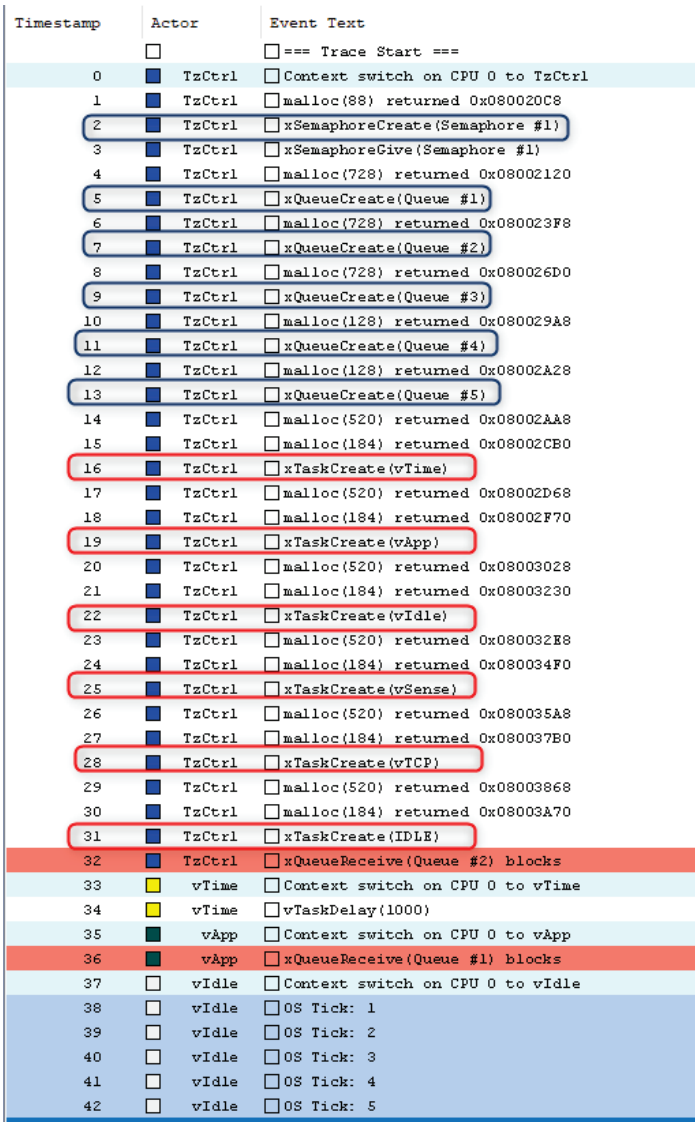
- Proporciona una representación visual de las tareas, interrupciones, eventos y recursos compartidos en el sistema.
- Permite medir los tiempos de ejecución de tareas, tiempos de espera y tiempos de suspensión.
- Identifica condiciones en las que múltiples tareas acceden a recursos compartidos simultáneamente, lo que podría provocar errores.
- Ayuda a identificar problemas y errores en el sistema, lo que simplifica la tarea de depuración y optimización.
- Permite observar el comportamiento del sistema mientras se ejecuta en tiempo real, lo que es esencial para aplicaciones críticas.

Para poder utilizarlo y analizar nuestro sistema, agregamos todos los archivos correspondientes del software en nuestro proyecto e hicimos las modificaciones correspondientes para que funcione con nuestro kit de desarrollo. Para que este software pudiera obtener los datos de las demás tareas, el Tracealyzer generó una tarea nueva, llamada TzCtrl, con la cual guardó la información sobre el resto de las tareas. Una vez configurado, comenzamos con las pruebas.

1. Creación de tareas, semáforo y colas.

   Iniciamos nuestro proceso de validación al asegurarnos que todas las tareas, semáforos y colas que diseñamos se crearan adecuadamente. Este primer paso fue fundamental para garantizar la correcta inicialización de nuestro Sistema Operativo. Ejecutamos nuestro proyecto en el Modo Nominal, y configuramos el Tracealyzer para registrar datos desde el inicio del Sistema Operativo.

   Comenzamos por comprobar que todas las tareas, semáforos y colas que programamos se crearan correctamente. Para eso, ejecutamos nuestro proyecto en el Modo Nominal, y configuramos al Tracealyzer para que grabe los datos desde el comienzo del Sistema Operativo. En la imagen número 43 apreciamos cómo inicia el sistema, en donde podemos observar el 'Timestamp', que es la referencia de tiempo con la que van sucediendo los eventos, el 'Actor', donde indica el nombre de las tareas que se están ejecutando en ese momento, y el 'Event Text', en donde nos proporcionó cierta información de lo que estaba ocurriendo en ese instante.



Timestamp	Actor	Event Text
		=== Trace Start ===
0	TzCtrl	Context switch on CPU 0 to TzCtrl
1	TzCtrl	malloc(88) returned 0x080020C8
2	TzCtrl	xSemaphoreCreate(Semaphore #1)
3	TzCtrl	xSemaphoreGive(Semaphore #1)
4	TzCtrl	malloc(728) returned 0x08002120
5	TzCtrl	xQueueCreate(Queue #1)
6	TzCtrl	malloc(728) returned 0x080023F8
7	TzCtrl	xQueueCreate(Queue #2)
8	TzCtrl	malloc(728) returned 0x080026D0
9	TzCtrl	xQueueCreate(Queue #3)
10	TzCtrl	malloc(128) returned 0x080029A8
11	TzCtrl	xQueueCreate(Queue #4)
12	TzCtrl	malloc(128) returned 0x08002A28
13	TzCtrl	xQueueCreate(Queue #5)
14	TzCtrl	malloc(520) returned 0x08002AA8
15	TzCtrl	malloc(184) returned 0x08002CB0
16	TzCtrl	xTaskCreate(vTime)
17	TzCtrl	malloc(520) returned 0x08002D68
18	TzCtrl	malloc(184) returned 0x08002F70
19	TzCtrl	xTaskCreate(vApp)
20	TzCtrl	malloc(520) returned 0x08003028
21	TzCtrl	malloc(184) returned 0x08003230
22	TzCtrl	xTaskCreate(vIdle)
23	TzCtrl	malloc(520) returned 0x080032E8
24	TzCtrl	malloc(184) returned 0x080034F0
25	TzCtrl	xTaskCreate(vSense)
26	TzCtrl	malloc(520) returned 0x080035A8
27	TzCtrl	malloc(184) returned 0x080037B0
28	TzCtrl	xTaskCreate(vTCP)
29	TzCtrl	malloc(520) returned 0x08003868
30	TzCtrl	malloc(184) returned 0x08003A70
31	TzCtrl	xTaskCreate(IDLE)
32	TzCtrl	xQueueReceive(Queue #2) blocks
33	vTime	Context switch on CPU 0 to vTime
34	vTime	vTaskDelay(1000)
35	vApp	Context switch on CPU 0 to vApp
36	vApp	xQueueReceive(Queue #1) blocks
37	vIdle	Context switch on CPU 0 to vIdle
38	vIdle	OS Tick: 1
39	vIdle	OS Tick: 2
40	vIdle	OS Tick: 3
41	vIdle	OS Tick: 4
42	vIdle	OS Tick: 5

Figura 43: Análisis de la creación de tareas usando el Tracealyzer

Podemos observar a la tarea **TzCtrl** en la figura anterior, que nos muestra la información precisa de la creación de semáforos, colas y tareas que estarán en ejecución. De esta manera es que corroboramos la creación de cada una de las tareas, colas y semáforos. Una vez que todas las tareas estaban creadas, comenzó la ejecución de las tareas mismas del sistema.

2. Ejecución y alternancia de tareas.

   Tras la fase inicial de creación de tareas y recursos, continuamos verificando la ejecución de las tareas, como **vTimer**, **vApp** y **vSense**, que hemos identificado en la observación previa. Para eso usamos la función 'Traces view', y analizamos un fragmento de tiempo.

   La tarea **vTimer** se encarga de gestionar el tiempo a bordo (OBT), esencial para la sincronización de las operaciones del nanosatélite. Su ejecución es periódica, ya que debe calcular el tiempo a bordo.

   La tarea **vApp**, responsable de las comunicaciones y de procesar los datos recibidos de **vTimer** y **vSense** se ejecuta aperiódicamente. Al ser una tarea principal, su tiempo activo es mayor que el del resto de las tareas.

   Por su parte, la tarea **vSense** es activada por la tarea **vApp**, por lo cual se activa luego de la ejecución de la tarea principal.

   Fue fundamental verificar que estas tareas se ejecutasen según lo planificado y que la alternancia entre ellas fuera coherente. Para ello, la herramienta Tracealyzer nos proporcionó una representación gráfica detallada, permitiéndonos observar los tiempos de ejecución, los eventos y la alternancia entre tareas en tiempo real.

   La interacción coordinada entre **vTimer** y **vApp**, evidenciada por la comunicación mediante la cola, reflejó una implementación eficiente y cooperativa de tareas para cumplir con los objetivos del sistema.

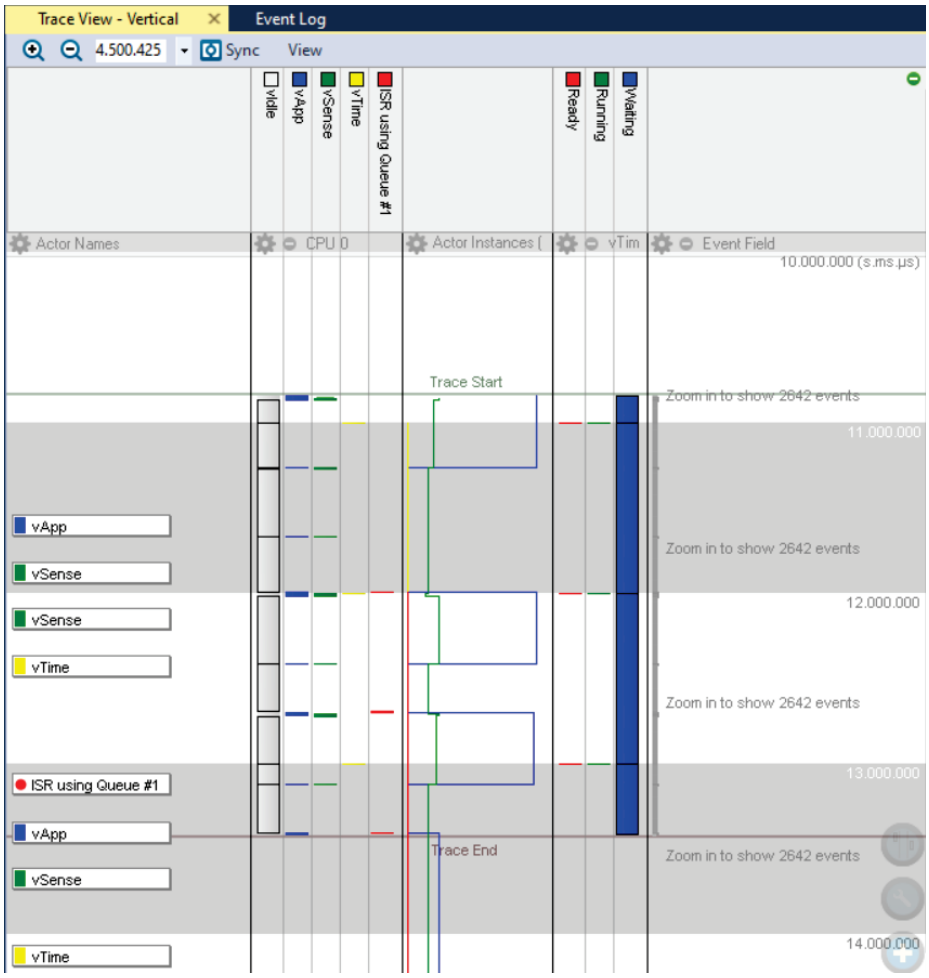


Figura 44: Fragmento de tiempo analizado con Tracealyzer

3. Consumo de las tareas

   Para analizar el consumo de memoria de las tareas, utilizamos los gráficos de análisis propios del Tracealyzer, llamados 'CPU Load Graph'. Para eso, guardamos los datos de la ejecución de las tareas en un intervalo de tiempo y analizamos los consumos de las mismas. En la imagen 45 podemos ver el porcentaje de uso de la CPU en el intervalo de tiempo comprendido. A simple vista, podemos observar que el pico máximo de consumo de la CPU fue del 14 %, y la tarea vApp quien realizó el mayor consumo de memoria.

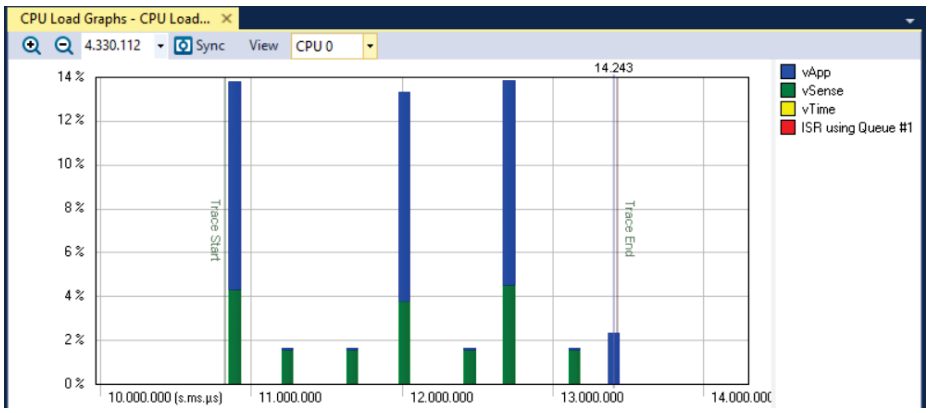


Figura 45: Fragmento de tiempo analizado con Tracealyzer

Realizando un análisis más detallado sobre cada una de las tareas, observamos la figura 46 que la tarea vApp tuvo un consumo de entre el 9 y el 10 % de la memoria. Este nivel de consumo fue aceptable, ya que esta tarea es la tarea responsable de comunicar el resto de las tareas, y de recibir y transmitir los mensajes por el puerto CAN.

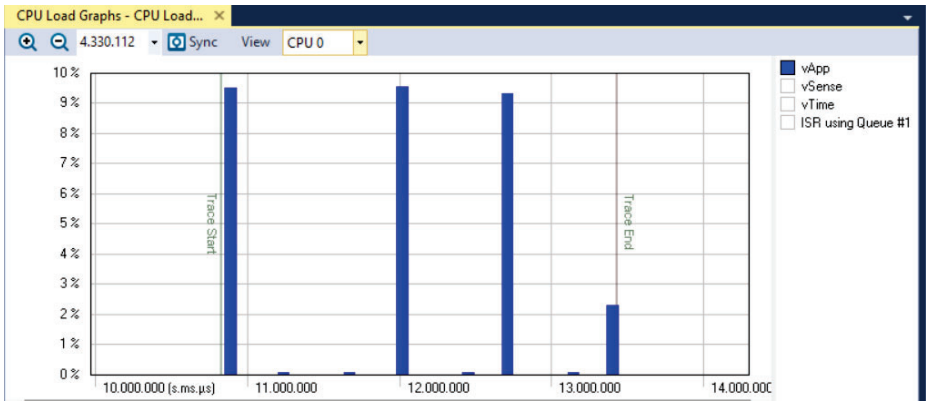


Figura 46: Consumo de memoria de la tarea vApp

Analizando en detalle la tarea vSense, cuyo comportamiento visualizamos en la imagen número 47, pudimos apreciar que esta tarea presentaba un consumo de recursos de la CPU en el rango del 3 al 4 % del total. La tarea vSense desempeñó un papel crucial en la operación de la POBC, encargándose de recopilar datos de telemetría específicos de la POBC y de transmitirlos hacia la estación terrestre. El hecho de que la tarea vSense mostró un consumo relativamente bajo de CPU fue coherente, ya que la recopilación y transmisión de datos de telemetría no necesariamente implica operaciones intensivas en términos de procesamiento.

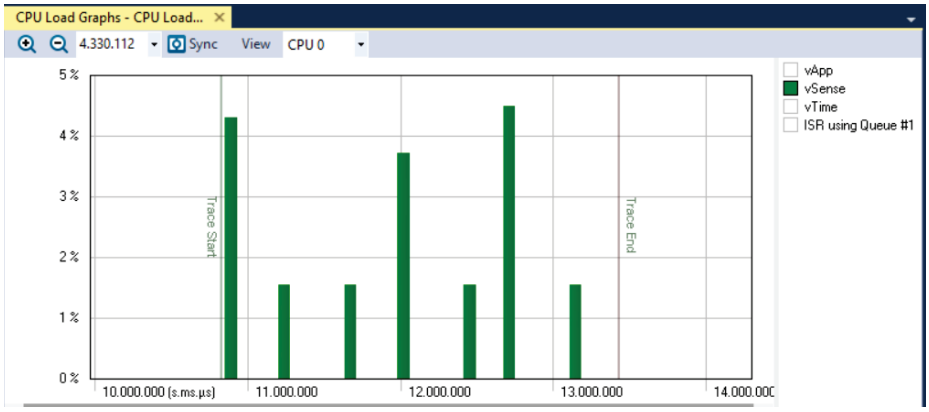


Figura 47: Fragmento de tiempo analizado con Tracealyzer

En lo que respecta a la tarea vTime, observamos su mínimo consumo de recursos, como se evidencia en la imagen 48, donde se registró un consumo inferior al 0.05 % de la memoria de la CPU. Este bajo consumo es totalmente entendible, ya que su ejecución se produce únicamente una vez por segundo. La principal responsabilidad de la tarea vTime es calcular el OBT, una operación que, aunque es esencial para la sincronización del sistema, no requiere una carga significativa de procesamiento. Esta tarea, al contribuir a la coordinación temporal de las operaciones del satélite, demostró un diseño eficaz al minimizar su impacto en la capacidad de procesamiento general del sistema.

4. Interrupción por CAN

   Para realizar un análisis del funcionamiento de las interrupciones externas, implementamos un procedimiento con la herramienta Tracealyzer en un intervalo de tiempo específico. Durante este intervalo, generamos una interrupción enviando un mensaje a la POBC a través del bus CAN. La visualización en la imagen 49 nos muestra la aparición de la interrupción externa identificada como 'ISR using Queue 1' en la vista detallada de eventos de Tracealyzer. Este evento específico señaló la llegada de un mensaje CAN a la POBC, y su representación en el Tracealyzer ofreció información valiosa sobre la temporización y la ejecución de las operaciones asociadas con esa interrupción externa. La

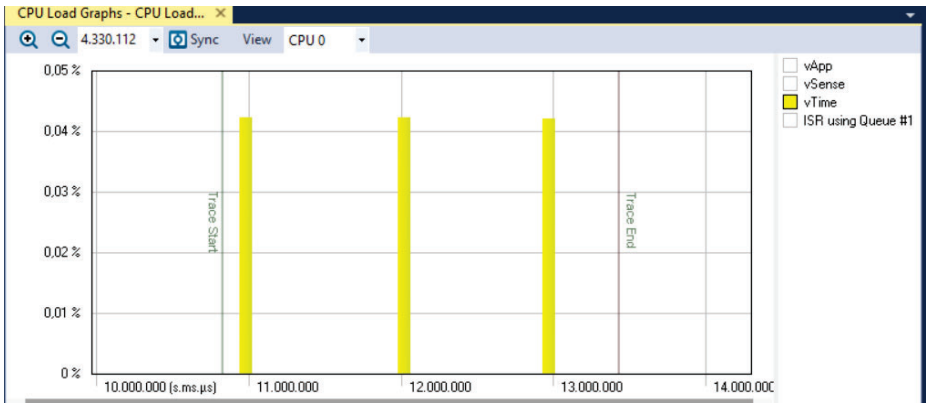


Figura 48: Fragmento de tiempo analizado con Tracealyzer

capacidad de rastrear y entender la secuencia temporal de estos eventos críticos fue crucial para garantizar la sincronización adecuada entre las interrupciones y las tareas correspondientes.

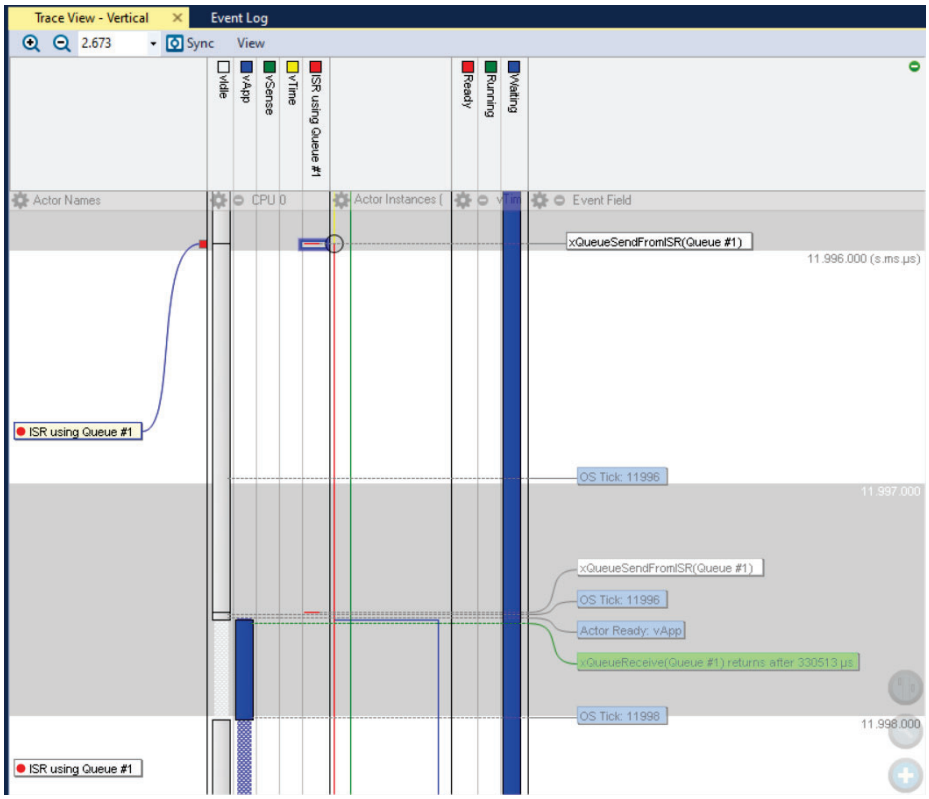


Figura 49: Interrupción por CAN

La imagen 50 proporciona información de cómo esta interrupción externa ac-

tivó la tarea vApp. La cola 'xQueueSendFromISR' se utilizó como el medio de comunicación entre la interrupción y la tarea, asegurando una transferencia de datos fluida y eficiente. Este enfoque de comunicación basado en colas fue fundamental para mantener la coherencia en la transmisión y procesamiento de datos en el sistema.

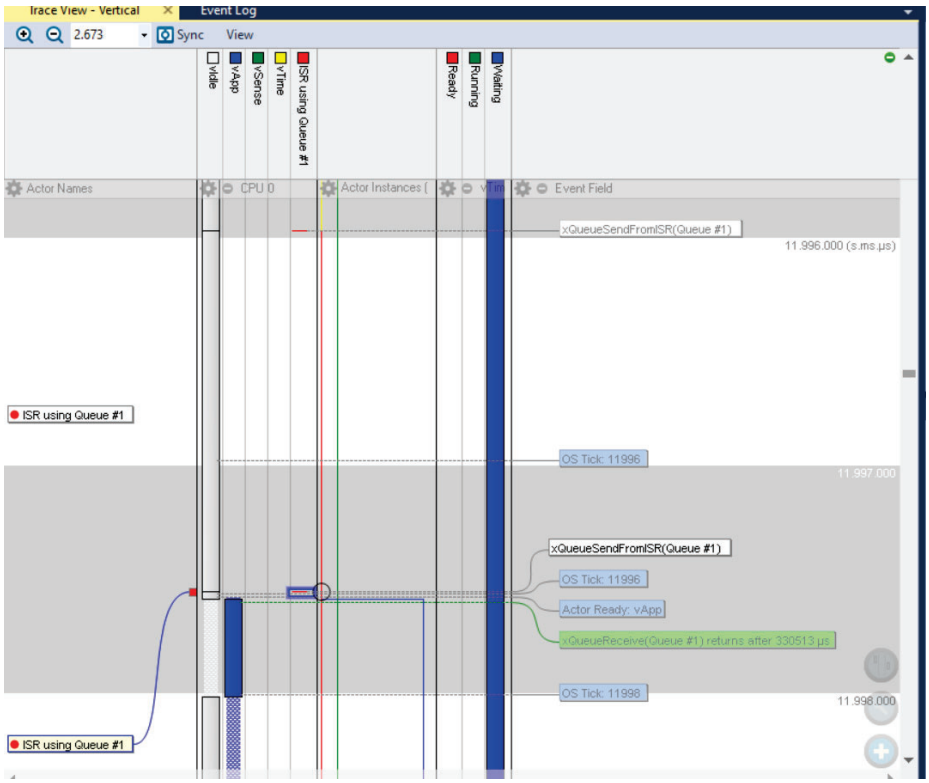


Figura 50: Activación de la tarea vApp mediante la interrupción externa recibida

De esta manera, es que confirmamos que nuestro sistema, previamente diagramado, funcionó acorde a su planificación. Tanto las tareas, su ejecución y consumos, como el uso de colas e interrupciones externas fueron validadas mediante cada uno de los test realizados.

5.1.3. Testeo del Modo de Actualización

La validación del Modo de Actualización la llevamos a cabo sin la necesidad de un software externo especializado para analizarla.

Dada la complejidad de este modo, recibimos el valioso respaldo de los expertos de Texas, quienes colaboraron en el desarrollo de ciertas etapas esenciales.

En primera instancia, para facilitar la transferencia de la imagen binaria a la POBC y corroborar que nuestro código funcionara correctamente, utilizamos una imagen binaria de ejemplo proporcionado por Texas, configurada para ser grabada en la posición de memoria 0x20020 de nuestro kit de desarrollo. En la fase de pruebas, empleamos un Arduino Uno como puente para transmitir la imagen binaria a través de un módulo CAN conectado a los puertos CAN de la POBC, y un módulo lector de memoria SPI para acceder a la imagen binaria del proyecto recibido por Texas.

El prototipo de prueba, detallado en la imagen 51, muestra la conexión del Arduino con los módulos CAN y el lector de SD, y su conexión a la POBC mediante CAN.

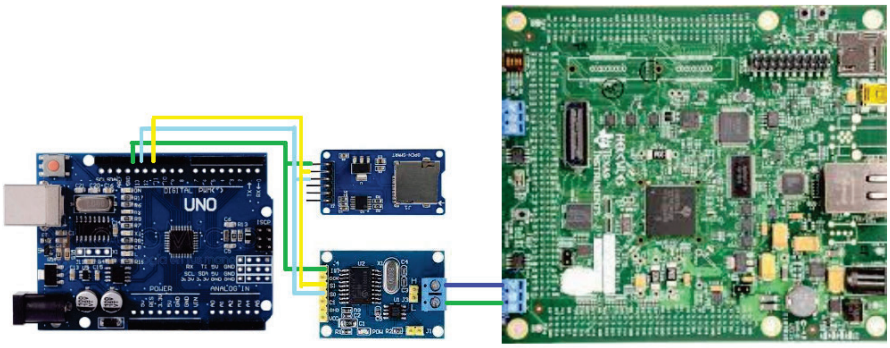


Figura 51: Prototipo armado para el testeo del Modo de Actualización

Con el prototipo montado, aseguramos una comunicación adecuada entre las placas, y grabamos la imagen binaria proporcionada por Texas en la memoria auxiliar. Siguiendo nuestro diagrama de actualización, explicado en la sección 4.1.3.4, programamos el Arduino para enviar los mensajes correspondientes desde la tarjeta de memoria a la POBC, facilitando así el proceso de prueba de la Actualización

Dentro del Modo de Actualización, el Arduino envió a la POBC la posición y la longitud del archivo binario a grabar. Luego de esa verificación, procedimos a enviar el archivo binario mediante mensajes CAN, que estaba guardado en la memoria SD. Para eso, el Arduino leyó los datos de la imagen binaria grabada en la SD y los envió mediante mensajes CAN hacia la POBC. Por su parte, la POBC recibió cada uno de los mensajes enviados por CAN con datos de la imagen binaria, los grabó en las posiciones de memoria correspondientes y respondió a cada mensaje con un comando ACK, confirmando la correcta recepción de los datos. Tras la transferencia completa

del archivo binario, reseteamos la POBC mediante un comando CAN antes de la ejecución de la nueva aplicación.

Para verificar que la imagen binario se grabó en la posición correcta de la POBC, utilizamos el 'Memory Browser' del IDE de Texas, el cual nos brindó información precisa de los datos grabados en nuestro kit en las distintas posiciones de memoria. Como puede observarse en las imágenes 52 y 53, vemos que la posición 0x20020 se encontraba vacía antes de iniciar el proceso, y con los datos luego de grabar la imagen binaria.

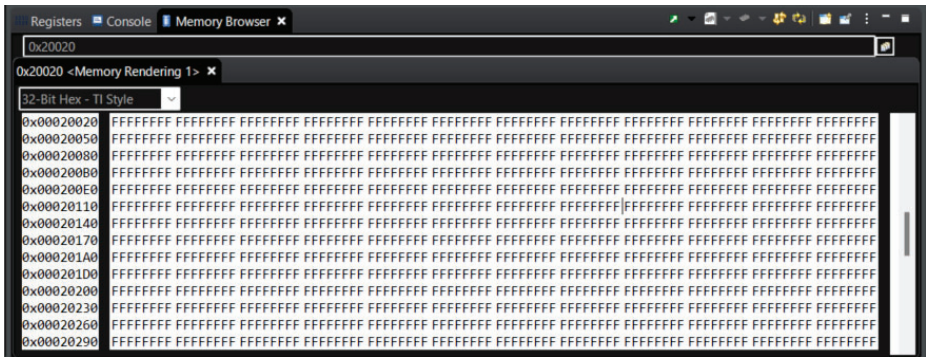


Figura 52: Información en la posición 0x20020 antes de grabar

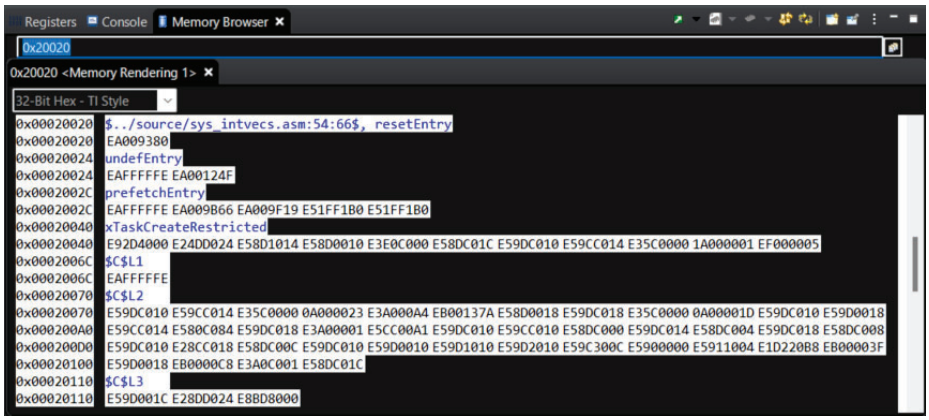


Figura 53: Información en la posición 0x20020 después de grabar

Con la grabación de la imagen binario concluida, procedimos a su ejecución. En el Modo Inicialización de la POBC, enviamos el comando 'Aplication' para ejecutar directamente la imagen grabada en la posición 0x20020. Confirmamos la ejecución correcta observando el encendido y apagado de tres LEDs programados en el código principal.

Posteriormente, replicamos el proceso de Actualización con un archivo binario propio del Modo Nominal, grabándolo en la misma posición 0x20020 de la POBC. Para

eso, generamos nuestro archivo binario a partir del Modo Nominal. Los expertos de FreeRTOS nos proporcionaron el soporte necesario, indicándonos las modificaciones de la tabla de vectores y de los archivos necesarios para poder utilizar una imagen binaria con FreeRTOS.

Corroboramos que el archivo binario estuviera grabado en el 'Memory Browser', y procedimos a ejecutarlo. La ejecución fue un éxito. Corroboramos que todas las tareas se hubieran ejecutado correctamente, y que respondieran a los comandos enviados por CAN y por Ethernet. En la imagen número 54 podemos ver la ejecución del Modo Nominal luego de ser actualizado por el Modo de Actualización.

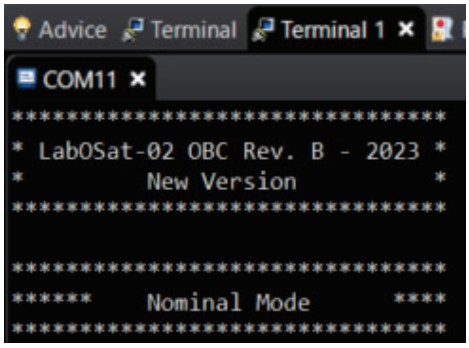


Figura 54: Consola de comandos al iniciar la nueva versión del Modo Nominal

5.1.4. Testeo del Modo Backup

Para comprobar el funcionamiento de este modo, hemos utilizado el mismo prototipo mostrado en la imagen número **??**, y hemos comprobado que la POBC responda a cada uno de los comandos enviados, y que cumpla con todas las funciones disponibles. Ya que este software está programado en Bare Metal, no utilizamos un programa especial para su análisis. Lo que hicimos fue corroborar que la POBC cumpla con todas las funciones codificadas, utilizando la consola de comandos y los LEDs integrados para su testeo. Corroboramos también, la comunicación con la Carga Útil en ambos sentidos. Muchas de las funciones son idénticas a las desarrolladas en el Modo Nominal, como por ejemplo el control de la Carga Útil, la recolección de telemetría el guardado de la información. Todas estas funciones fueron chequeadas, y se observó que su comportamiento era el esperado, cumpliendo con los mismos objetivos que en el Modo Nominal. En la imagen número 55, pudimos observar el correcto inicio de este modo de funcionamiento.

Con estos procesos de validación, concluimos exitosamente las pruebas sobre nuestro firmware de la POBC, poniendo a prueba la robustez del proyecto y corroborando que funcionase de acuerdo a lo planificado según los diagramas explicados en la sección anterior. A continuación procederemos a explicar los testeos realizados en los protocolos de comunicación utilizados.

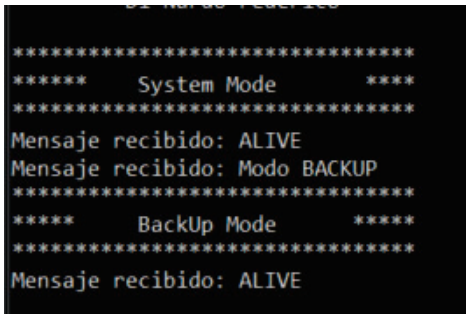


Figura 55: Consola de comandos al iniciar el Modo Backup

5.2. Protocolos de comunicación

En esta sección vamos a explicar cómo es que testeamos los protocolos de comunicación utilizados en nuestro proyecto. Cada uno de ellos fue testeado con distintas técnicas, por lo que explicaremos cada protocolo en una sección dedicada.

5.2.1. CAN

El protocolo de comunicación CAN fue el primero que comenzamos a testear, ya que fue elemental para poder progresar con nuestro desarrollo. Este protocolo se utiliza especialmente para comunicar la POBC con nuestro PSC, y poder realizar todas las pruebas en cada uno de los distintos modos de funcionamiento.

Como mencionamos en secciones anteriores, nuestra POBC cuenta con dos puertos CAN (CAN 1 y CAN 2). En una primera instancia, para comprobar que ambos puertos CAN estuvieran funcionando, lo que hicimos fue utilizar los dos puertos CAN que tiene nuestro Kit de desarrollo, y conectarlos entre sí, como podemos observar en la imagen número 56. Es decir, hicimos un bucle entre los dos puertos CAN de nuestra plaqueta, y procedimos a programar el funcionamiento de ambos puertos en un mismo proyecto. Comenzamos testeando la transmisión de datos por el puerto CAN 1, y la recepción de comandos por sondeo en el puerto CAN 2. Una vez que verificamos que los mensajes llegaban de un puerto a otro correctamente (tanto el ID como los datos de información enviada en el mensaje), procedimos a realizar las mismas pruebas pero, en este caso, activando la interrupción en el puerto CAN 2. Estas pruebas fueron exitosamente realizadas.

Una vez validada la transmisión de datos por el puerto CAN 1 y la recepción de comandos por el puerto CAN 2, procedimos a testear el camino inverso, es decir, la transmisión de datos por CAN 2 y la recepción de comandos por CAN 1. Realizamos las mismas pruebas de recepción de comandos tanto por sondeo como por interrupción.

En ambos casos, realizamos los testeos con distintas velocidades, y corroboramos que tanto el ID como los datos llegaron correctamente de un puerto al otro. De

esta manera, verificamos que tanto la transmisión y la recepción de mensajes CAN funcionaba correctamente.

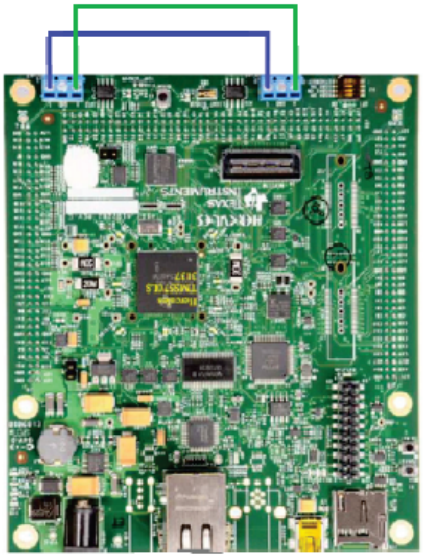


Figura 56: Puente entre ambos puertos CAN de la POBC

Una vez finalizado el testeo de ambos puertos CAN entre los mismos, procedimos a corroborar la comunicación contra otra plaqueta distinta. Para eso, programamos un Arduino UNO, al cual le otorgamos una interface CAN mediante un módulo CAN MCP2515, y conectamos el módulo a uno de los puertos CAN de nuestro Kit de la POBC, como podemos observar en la figura número 57. Programamos el Arduino para que pudiera comunicarse con la POBC por medio de los mensajes CAN. Una vez conectado, comenzamos con los testeos. Primero probamos conectando el Arduino al puerto CAN 1 de nuestra POBC, y testeamos la comunicación bidireccional. Una vez que observamos que en ambos lados recibíamos los mensajes correctamente, procedimos a probar el puerto CAN 2 de la POBC.

En ambos casos, realizamos las pruebas con distintos comandos, y distintas velocidades (1 Mbps, 500 Kbps, 250 Kbps y 125 Kbps).

Para asegurar la robustez de la señal enviada a través del bus de comunicación CAN, implementamos un enfoque de verificación adicional mediante la conexión de un osciloscopio al sistema. Esta herramienta de medición nos permitió realizar un análisis detallado de las características de la señal CAN, evaluando tanto la transmisión como la recepción de mensajes.

Para llevar a cabo esta tarea, empleamos ambos canales del osciloscopio, conectándolos respectivamente a los pines CAN_H y CAN_L del bus de comunicación. Esta configuración nos permitió medir la diferencia de potencial entre las señales CAN_H y CAN_L, obteniendo así una visión completa de la calidad de la señal CAN.

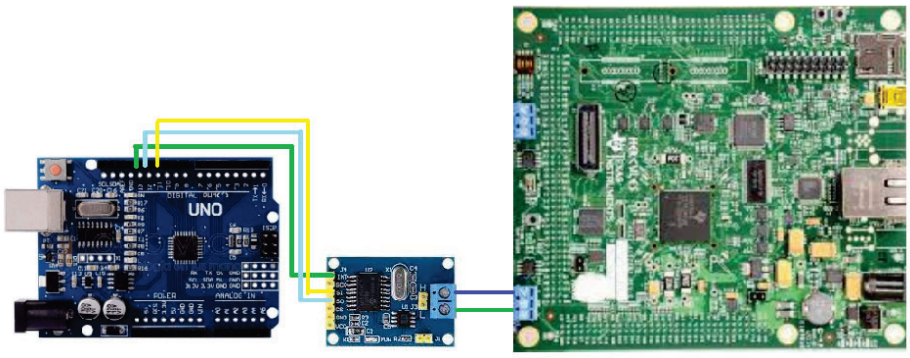


Figura 57: Conexión del Arduino y la POBC por el protocolo CAN

En la imagen número 58 podemos observar la señal de nuestra POBC obtenida por el osciloscopio. Podemos ver que la señal es muy estable, y que no había variaciones de tensión de la misma.

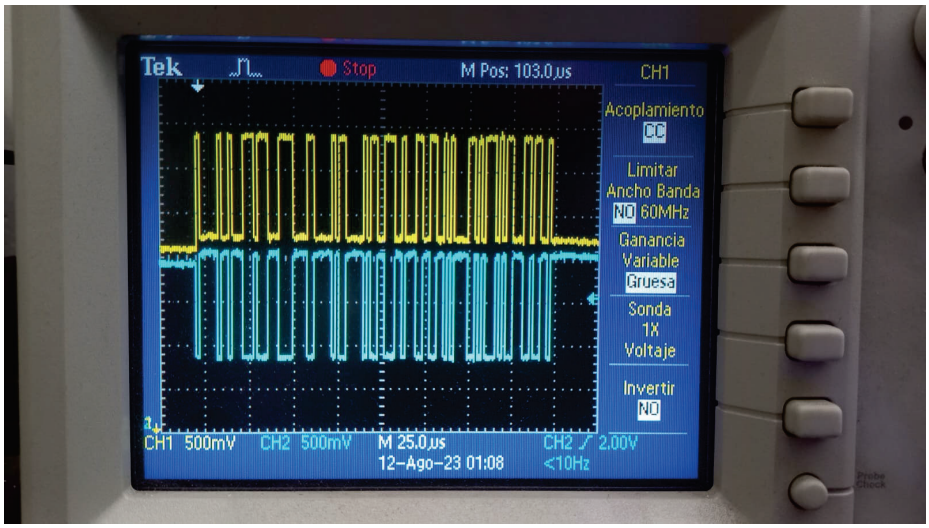


Figura 58: Medición de la señal CAN obtenida por el osciloscopio

Para finalizar con las pruebas sobre el protocolo CAN, decidimos conectar nuestro kit de desarrollo con el PSC, como podemos observar en la figura número 59, la cual representa la configuración física de esta conexión. Programamos el PSC con sus dos módulos CAN, y conectamos cada módulo CAN a un puerto CAN de la POBC. Programamos la STM32 de manera que pudiera comunicarse de manera bidireccional con la POBC.

Una vez conectada, procedimos a comunicar ambas plaquetas por ambos puertos CAN. Testeamos la transmisión y recepción de los comandos CAN, en ambos sentidos

y por ambos puertos. Probamos todos los ID programados, con distintos datos y verificamos que llegaron bien.

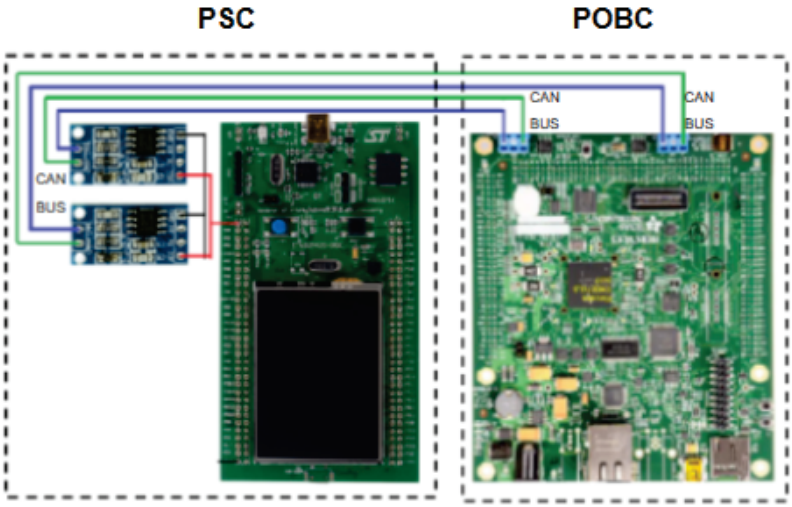


Figura 59: Conexión física entre la POBC y el PSC

En resumen, las pruebas de comunicación CAN culminaron con éxito, confirmando la eficacia y confiabilidad de la transmisión y recepción de mensajes CAN en el entorno de nuestro proyecto.

Procederemos ahora con las pruebas de protocolo Ethernet.

5.2.2. Ethernet

El protocolo de comunicación Ethernet se utiliza para poder controlar la Carga Útil, mediante la POBC. Nuestro kit de desarrollo cuenta con una interface Ethernet, mediante el puerto RJ45. Para su testeo, debimos programar el protocolo utilizando el stack de bibliotecas lwIP, el cual contiene los archivos necesarios para poder utilizar los distintos protocolos de transmisión dentro del protocolo TCP/IP.

Comenzamos su implementación sobre el sistema FreeRTOS. Su programación no fue algo intuitivo, por lo cual tuvimos que abordar varias consultas sobre nuestros foros de consultas de Texas y de FreeRTOS. Programamos las tareas para poder enviar los paquetes TCP desde la POBC. Para eso, habilitamos el puerto Ethernet desde la plaqueta, le otorgamos una dirección IP y un puerto por el cual debía comunicarse. Decidimos otorgar en ambos casos direcciones IPs fijas. Para la POBC utilizamos la dirección 192.168.0.25 con el puerto 7001, y para la Carga Útil utilizamos la dirección 192.168.0.20, con el puerto 7070.

Para poder probarlo, decidimos simular la Carga Útil con un software instalado en una computadora. Utilizamos el esquema de la figura número 60 para poder avanzar con los tests, en donde podemos ver que la POBC y la Carga Útil estaban comunicadas

por medio de una conexión Ethernet. El PSC ya se encontraba montado desde las pruebas realizadas de CAN, y fue necesario para el envío de comandos hacia la Carga Útil.

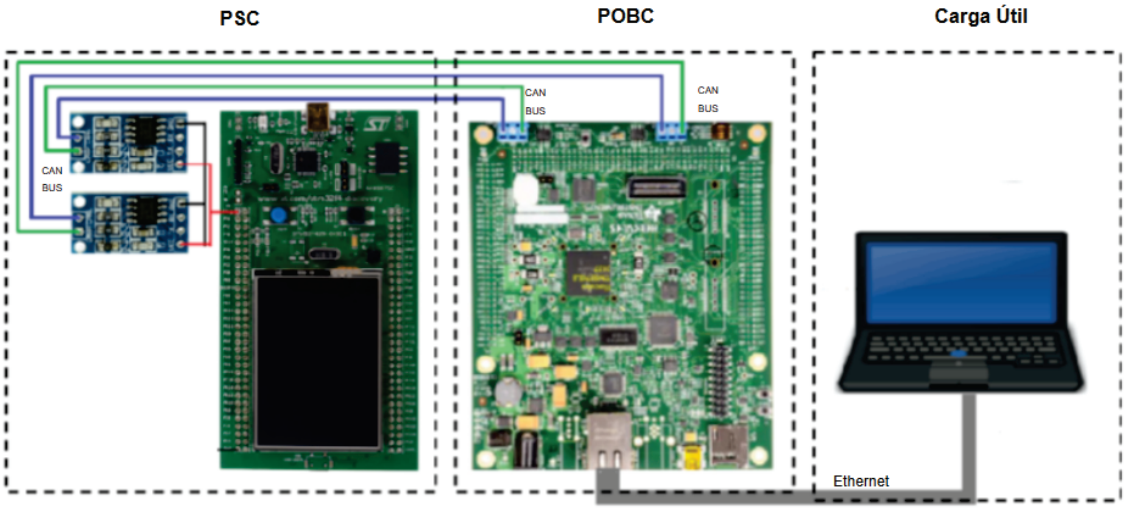


Figura 60: Prototipo montado para probar la comunicación Ethernet

Para simular la Carga Útil montamos un Servidor TCP en nuestra computadora. Esto lo hicimos mediante un software llamado Socket TCP, que se encarga de abrir clientes/servidores con distintas IPs y en distintos puertos disponibles. La configuración de nuestro servidor TCP puede observarse en la figura 61.

Para poder testear la comunicación TCP, y asegurarnos de que el protocolo funcionase correctamente, utilizamos mensajes de debug en la consola de programación de la POBC, y un sniffer para poder observar con más detalles los mensajes intercambiados. Para eso, instalamos el Wireshark en nuestra computadora, en donde teníamos montado el servidor TCP, y lo configuramos de manera que estuviera escuchando las conversaciones por nuestro puerto Ethernet, y nos mostrase todos los paquetes entrantes y salientes del puerto Ethernet.

En el Wireshark esperábamos ver una serie de mensajes, mediante los cuales nos pudiéramos asegurar de que la comunicación funcione correctamente. Esos pasos son:

1. Solicitud de dirección MAC: La POBC inicia el proceso con la necesidad de conocer la dirección MAC del servidor al que desea conectarse. Para lograrlo, emite un paquete ARP, una solicitud que busca obtener la dirección MAC correspondiente a la IP del Servidor TCP.

2. Sincronización TCP: El siguiente paso crucial es la sincronización de ambos dispositivos según el protocolo TCP. Este procedimiento consta de tres pasos distintos, cada uno representado por un paquete específico:

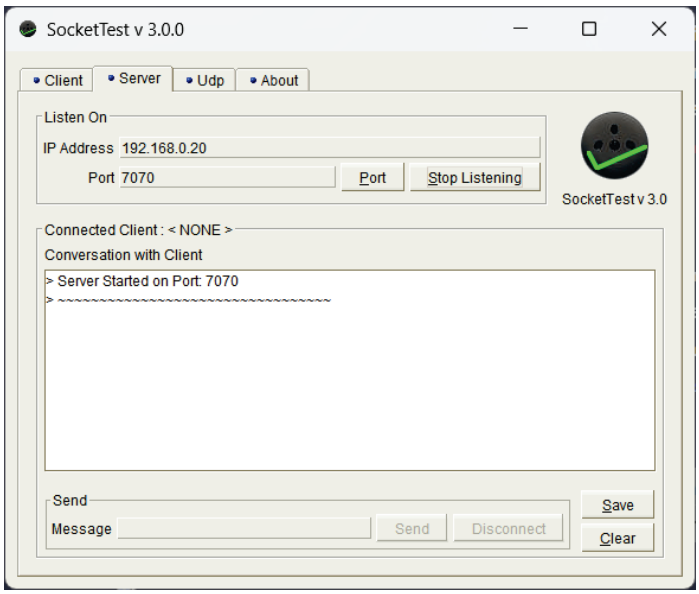


Figura 61: Configuración del Socket TCP que simula la Carga Útil

- La POBC envía un paquete SYN al servidor TCP, indicando su intención de establecer una conexión.
- El Servidor TCP responde con un paquete SYN ACK, confirmando la solicitud de conexión y esperando la confirmación final.
- La POBC envía el último paquete ACK, confirmando la sincronización y estableciendo la conexión de manera completa.

3. Intercambio de mensajes TCP: Una vez que ambas plaquetas hayan terminado con la negociación ARP y la sincronización, ya están lista para poder cambiar información mediante mensajes TCP. Por cada mensaje de transmisión TCP, se espera ver en el Wireshark un mensaje ACK de confirmación de recepción.

Una vez que estaba la conexión realizada, y con nuestro sniffer corriendo y testeando la comunicación Ethernet, procedimos a iniciar la conversación por TCP. La implementación de esta comunicación no fue algo sencillo, ya que no teníamos experiencia previa en la programación del protocolo TCP/IP, por lo cual debimos realizar ciertas modificaciones en nuestro código para su correcto funcionamiento.

Comenzamos las pruebas inicializando el protocolo TCP en nuestra POBC, por medio del comando PDA_ON, el cual fue enviado a través de CAN por nuestro PSC. Verificamos que la POBC inicializara el protocolo TCP por medio de un mensaje mostrado por consola, el cual podemos observar en la figura número 62.

Una vez que inició el protocolo IP, como podemos observar en la figura 63, la POBC envió un anuncio ARP con su dirección IP y su MAC, y en la figura número

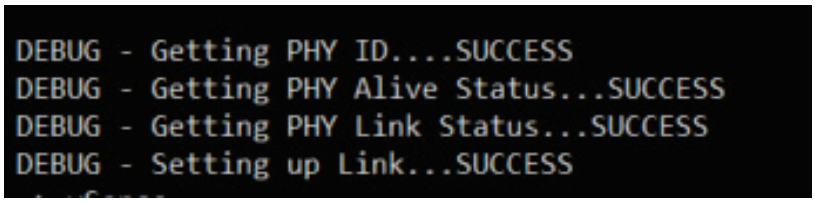


Figura 62: Mensaje de inicialización del protocolo TCP en la POBC

64 la POBC realizó el pedido de dirección MAC de la dirección IP que tenía como destino, es decir, la dirección IP de la Carga Útil. Este pedido de dirección MAC lo hizo una vez que habilitamos el puerto Ethernet.

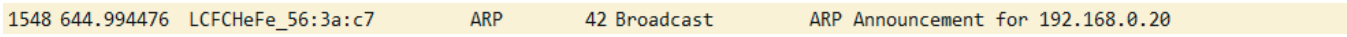


Figura 63: Anuncio ARP de la POBC

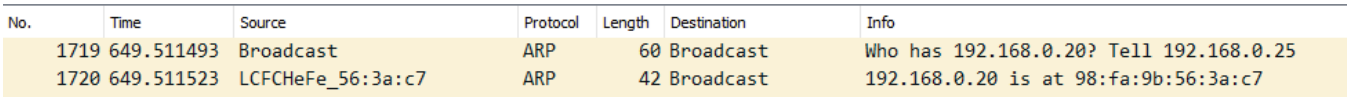


Figura 64: Mensaje de inicialización del protocolo TCP en la POBC

Una vez que el protocolo Ethernet estaba iniciado, procedimos a enviarle el comando TCP_INIT a la POBC para que inicie el protocolo TCP y comience con el sincronismo. En la imagen 65 podemos observar la información que nos proporcionó el Wireshark, en donde vimos la sincronización de tres pasos entre la POBC y la Carga Útil simulada. También, observamos en la imagen número 66 que el Servidor TCP ya reconoció a la POBC con la IP 192.168.0.25, y está listo para poder iniciar la conversación.

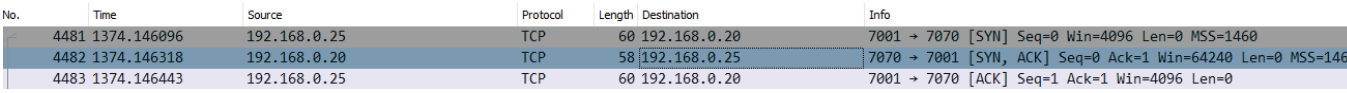


Figura 65: Sincronización de tres pasos entre la POBC y la Carga Útil

Una vez ya establecida la sincronización TCP, procedimos a testear la comunicación. Para eso, enviamos un mensaje TCP mediante el comando PAY1_TCP, con la información 'Hola_POBC' como dato, desde la POBC hacia la Carga Útil. Como puede observarse en la imagen 67, verificamos en el Wireshark que el protocolo estaba

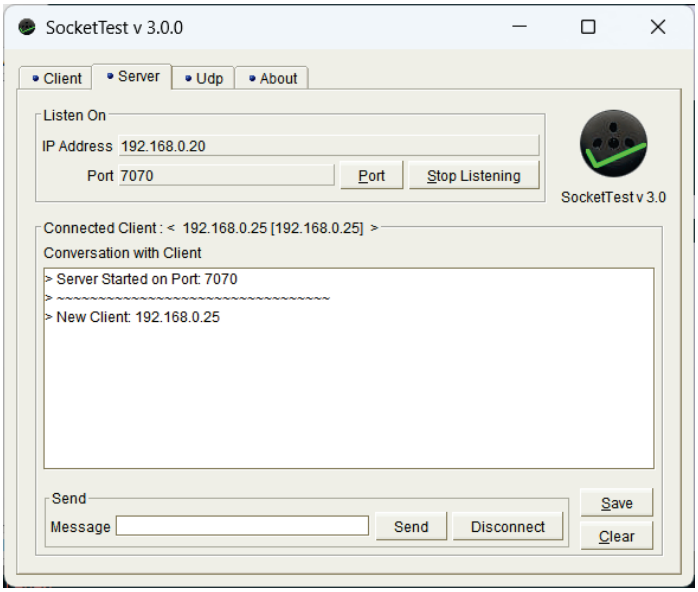


Figura 66: Sincronización del Servidor TCP en la Carga Útil

No.	Time	Source	Protocol	Length	Destination	Info
16	7.971534	192.168.0.25	TCP	73	192.168.0.20	7001 → 7070 [PSH, ACK] Seq=1 Ack=1 Win=4096 Len=19
17	8.011577	192.168.0.20	TCP	54	192.168.0.25	7070 → 7001 [ACK] Seq=1 Ack=20 Win=64183 Len=0

Figura 67: Captura del mensaje TCP realizada por el Wireshark

funcionando correctamente, y también corroboramos su arribo en el Servidor TPC, el cual podemos observar en la imagen 68.

Luego de verificar la transmisión desde la POBC y recepción de mensajes en la Carga Útil, procedimos a testear el camino inverso. Para eso, programamos la POBC para que reciba mensajes por el puerto Ethernet por medio de una interrupción.

Transmitimos un mensaje desde la Carga Útil, con los datos '48 4F 4C 5F 4F 42 43', como pudimos observar en la imagen 69. Corroboramos el envío del mensaje en el Wireshark, y la confirmación de recepción del mismo en la POBC, como se observa según la imagen número 70.

El mensaje con la información fue recibido por la POBC, la cual lo transmitió por CAN hacia el PSC. Para corroborar la correcto recepción, usamos la consola de comandos del PSC. Pudimos ver que la POBC que activó el puerto Ethernet, realizó la sincronización, y recibió el comando transmitido por la Carga Útil con los datos correctos. Esto lo podemos observar en la imagen número 71.

Una vez completado todos los pasos de la comunicación por Ethernet, concluimos en que el protocolo Ethernet funcionaba correctamente.

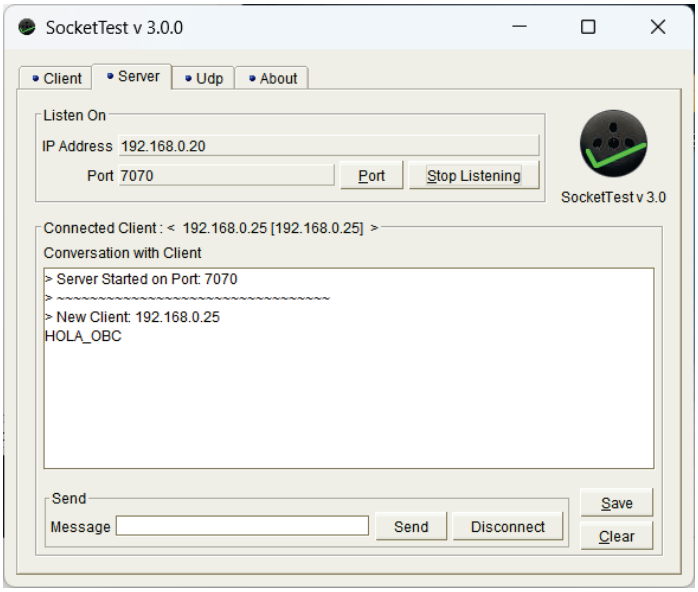


Figura 68: Recepción del mensaje TCP en la Carga Útil

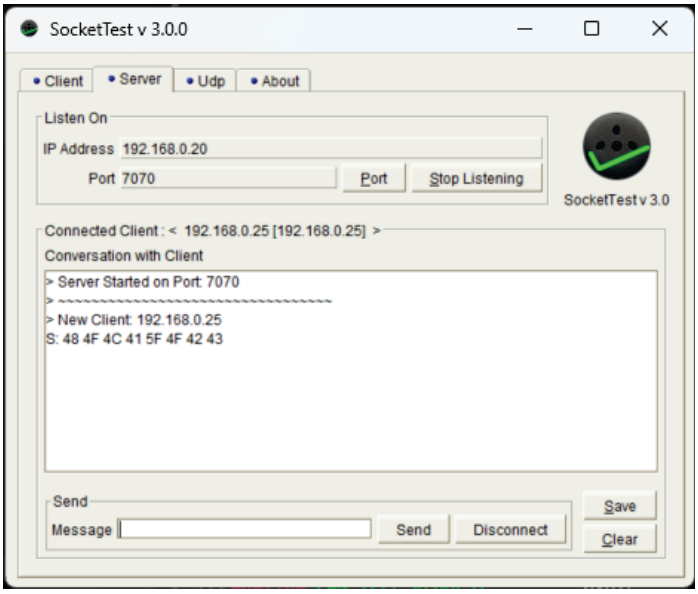


Figura 69: Transmisión de un mensaje TCP desde la Carga Útil

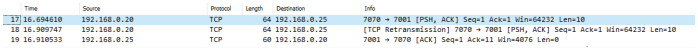


Figura 70: Captura del mensaje TCP realizada por el Wireshark

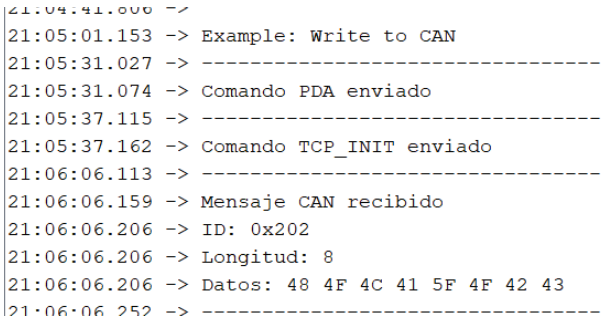


Figura 71: Recepción del comando Ethernet enviado desde la Carga Útil

5.2.3. LoRa

La comunicación LoRa fue la última en integrarse a nuestro proyecto, y su análisis lo realizamos una vez que ya habíamos corroborado el firmware de la POBC, y la comunicación por CAN y por Ethernet.

Para poder realizar el testeo de la comunicación, utilizamos cuatro módulos LoRa en total: dos para el PSC, y otros dos para recibir y enviar los comandos desde la Estación Terrena. Para eso, conectamos dos módulos LoRa a nuestro PSC, como podemos observar en la figura número 72, completando así nuestro prototipo de PSC. En el otro extremo, decidimos armar un prototipo de Estación Terrena, el cual comandamos a través de una computadora. Este prototipo de Estación Terrena fue conformado por dos Arduinos, en donde cada uno comandó un módulo LoRa, y fueron programados y controlados por una computadora. El prototipo de Estación Terrena fue programado para recibir y enviar mensajes por LoRa utilizando la frecuencia 433 MHz, la cual es una frecuencia de uso libre no comercial. Este prototipo lo podemos observar en la figura número 73. Si bien Argentina adoptó la versión Australiana de 915 MHz para la comunicación LoRaWAN de IOT, decidimos utilizar la frecuencia 433 MHz para la comunicación LoRa, aprovechando sus características como protocolo de comunicación, sin necesidad de su uso específico para IoT.

Comenzamos testeando el enlace descendente, es decir, la transmisión desde el PSC hacia la Estación Terrena. Para eso, programamos nuestro PSC para que transmitiera mensajes por LoRa. Configuramos el kit de desarrollo para que pudiera controlar los módulos LoRa, y para que enviara mensajes hacia la Estación Terrena. Por otra parte, programamos ambos Arduinos de la Estación Terrena para que recibieran los mensajes, y nos los mostraran en la consola de comandos. Como podemos ver en la figura 74, se muestran los mensajes que recibimos por LoRa junto al nivel de potencia de la señal.

Seguimos nuestro testeo en segundo lugar, transmitiendo desde el PSC los datos que la POBC le transmitía por CAN. Para eso, programamos el PSC para que cada paquete que recibiera por CAN, lo encapsulara y lo transmitiera en un mensaje LoRa. Utilizamos 12 bytes útiles por cada mensaje transmitido, de los cuales 8 bytes estaban

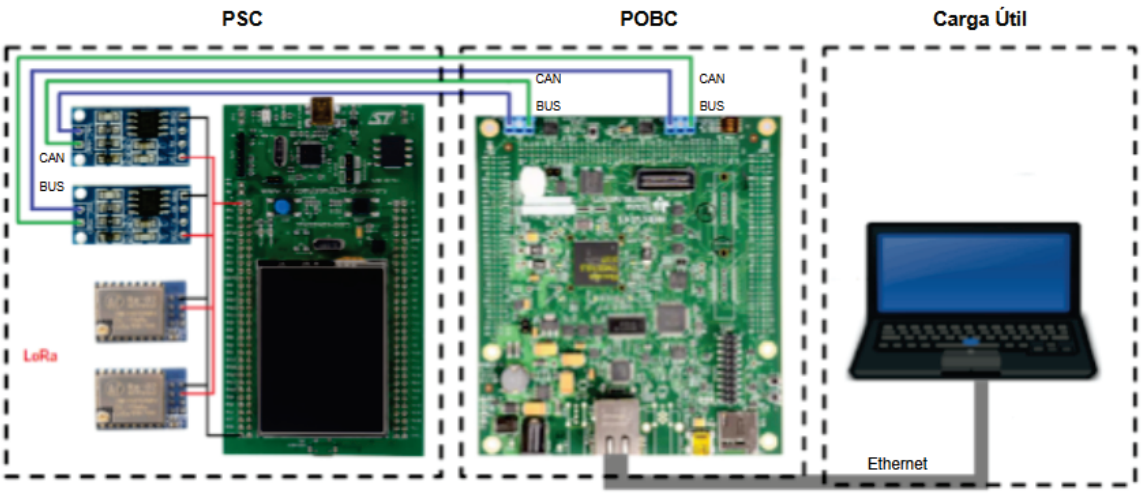


Figura 72: Prototipo de nanosatélite

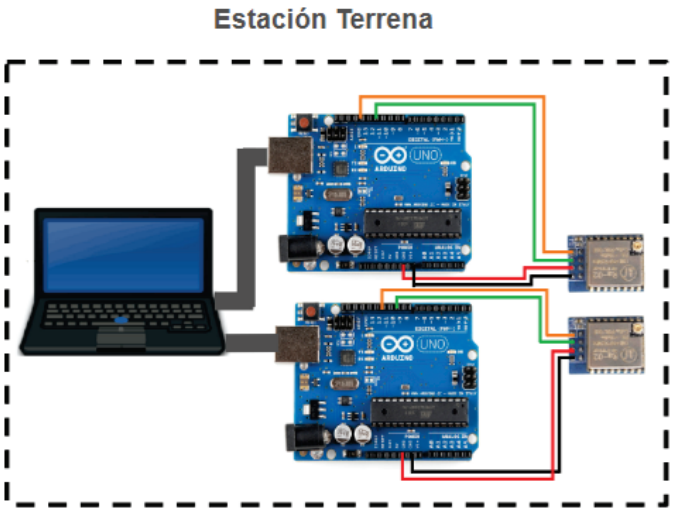


Figura 73: Prototipo de la Estación Terrena

destinados para enviar los datos del mensaje CAN a enviar, y los otros 4 bytes los utilizados para indicar el ID del mensaje, y la fuente del mismo, es decir, indicando si era propio de la POBC, de la Carga Útil o del PSC.

Como podemos ver en la figura 75, observamos en la consola de la Estación Terrena los datos recibidos por LoRa, en donde vimos la información propia de la POBC.

Ya habiendo corroborado la recepción de los mensajes provenientes de la POBC en nuestra Estación Terrena, decidimos proceder a analizar el presupuesto de enlace realizado. Para eso, decidimos graficar los datos obtenidos sobre el nivel de potencia

```
17:58:56.878 -> Mensaje LoRa recibido con RSSI -59
17:58:57.207 -> Mensaje LoRa recibido con RSSI -59
17:58:57.578 -> Mensaje LoRa recibido con RSSI -59
17:58:58.278 -> Mensaje LoRa recibido con RSSI -60
17:58:58.607 -> Mensaje LoRa recibido con RSSI -60
17:58:59.164 -> Mensaje LoRa recibido con RSSI -60
17:58:59.492 -> Mensaje LoRa recibido con RSSI -59
```

Figura 74: Mensajes LoRa recibidos en la Estación Terrena

```
17:47:27.691 -> ----------------------------------------------------------------
17:47:27.758 -> Received packet 'AA7F78E9E1E1E238345C51' with RSSI -56' and SNR 12.75dB
17:47:27.836 -> The message is from the OBC.
17:47:27.874 -> The ID is: 7F7
17:47:27.874 -> The data lenght is: 8
17:47:27.874 -> 23.3,22.5,22.5,22.6,5.6,5.2,0.92,0.81
17:47:27.907 -> The RSSI is: -56
17:47:28.058 -> ----------------------------------------------------------------
17:47:28.091 -> Received packet 'AA7F78E5E8E9E137335C53' with RSSI -56' and SNR 12.50dB
17:47:28.157 -> The message is from the OBC.
17:47:28.191 -> The ID is: 7F7
17:47:28.191 -> The data lenght is: 8
17:47:28.241 -> 22.9,23.2,23.3,22.5,5.5,5.1,0.92,0.83
17:47:28.274 -> The RSSI is: -56
17:47:28.357 -> ----------------------------------------------------------------
17:47:28.441 -> Received packet 'AA620800000429' with RSSI -57' and SNR 12.75dB
17:47:28.491 -> The message is from the OBC.
17:47:28.491 -> The ID is: 620
17:47:28.540 -> The data lenght is: 8
17:47:28.574 -> The data are: 00000429
17:47:28.574 -> The RSSI is: -57
17:47:28.890 -> ----------------------------------------------------------------
17:47:28.957 -> Received packet 'AA7F78E3E5E5E338345C51' with RSSI -56' and SNR 12.50dB
17:47:29.024 -> The message is from the OBC.
17:47:29.057 -> The ID is: 7F7
17:47:29.057 -> The data lenght is: 8
17:47:29.091 -> 22.7,22.9,22.9,22.7,5.6,5.2,0.92,0.81
17:47:29.140 -> The RSSI is: -56
17:47:29.223 -> ----------------------------------------------------------------
```

Figura 75: Mensajes LoRa recibidos en la Estación Terrena con información de la POBC

y la relación señal a ruido, y compararlos con el presupuesto. Implementamos una interface gráfica en nuestra Estación Terrena, mediante la cual observamos los niveles mencionados, y los datos de telemetrías recibidos por parte de la POBC. Esta interface nos facilitó el testo de nuestro enlace. La programamos mediante Visual Estudio y la podemos observar en la figura número 76. Vemos en el recuadro denominado 'LoRa' los gráficos correspondientes al enlace de comunicación, en el recuadro 'Telemetry' los gráficos de las telemetrías provenientes de la POBC, como por ejemplo las temperaturas, tensiones y corrientes, el valor del OBT, y la fuente del mensaje recibido.

Utilizamos estos gráficos para observar los niveles recibidos en la Estación Terrena de la potencia (RSSI, por sus siglas en inglés) y de la Relación Señal a Ruido (SNR). Comenzamos testeando el enlace con la configuración de fábrica de los módulos LoRa, es decir, con una frecuencia de transmisión de 433 MHz, un ancho de Banda de 125KHz, un Spreading Factor de 7, y una potencia de transmisión de 17 dBm. Podemos observar en la figura número 77, los niveles de potencia y de relación señal a ruido

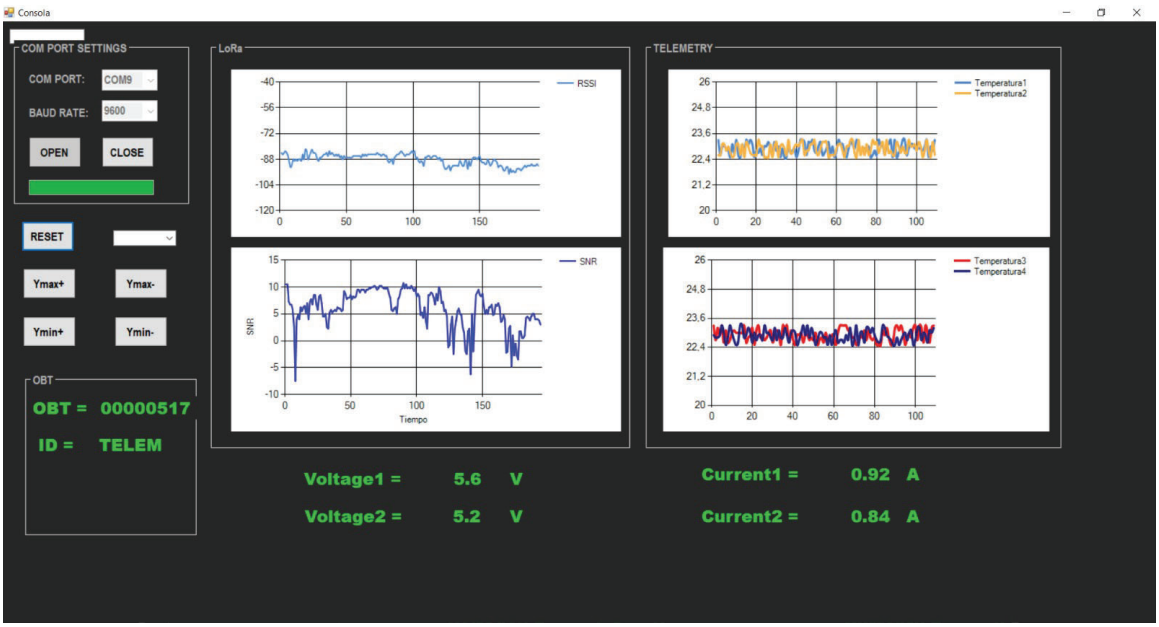


Figura 76: Mensajes LoRa recibidos en la Estación Terrena con información de la POBC

recibidos. Esta prueba la realizamos a una distancia de 10 metros. Luego procedimos a aumentar la potencia de transmisión a 20 dBm, como podemos observar en la figura número 78, en donde vemos un incremento del nivel de potencia recibido. El valor de potencia recibido concuerda con los cálculos realizados en nuestro presupuesto de enlace, según la figura número 35 de la sección 4.2.4.

Una vez que confirmamos la correcta recepción de los datos transmitidos, procedimos a mejorar nuestro enlace de comunicación. Para eso, testeamos el enlace con distintos Spreading Factors.

Probamos el enlace con un spreading factor de 8. Podemos observar en la imagen número 79 que el nivel de potencia recibida fue entre -28 y -29 dBm, y obtuvimos una SNR de 11 dB aproximadamente, unos 2 dB por encima del test anterior.

Realizamos las mismas pruebas, con un Spreading Factor con un valor de 9 y 10, como vemos en las figuras 80 y 81 respectivamente. En ambos casos, el nivel de potencia recibida fue entre -26 y -27 dBm, manteniéndose la transmisión más estable con Spreading Factor de 9. En cuanto a la relación señal a ruido, obtuvimos un valor medio de 9,4 y 9,8 dB.

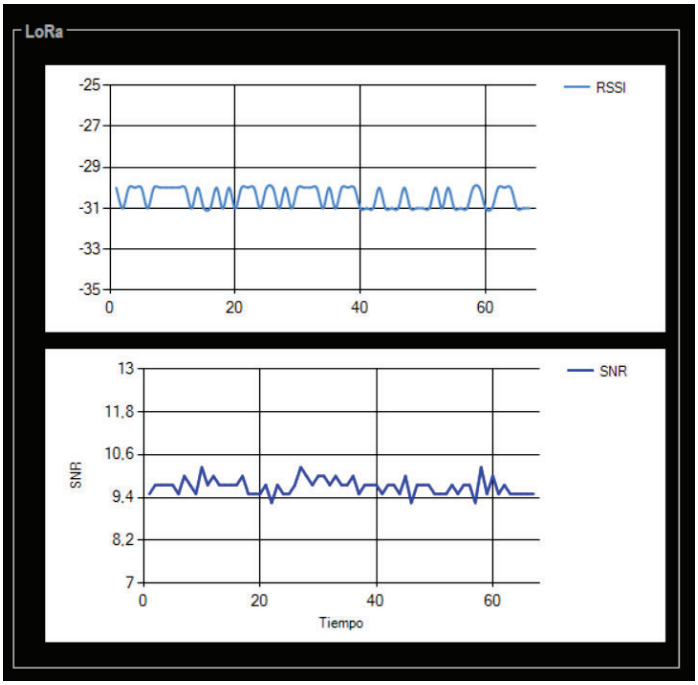


Figura 77: Distancia: 10m - SF: 7 - PTx: 17dBm

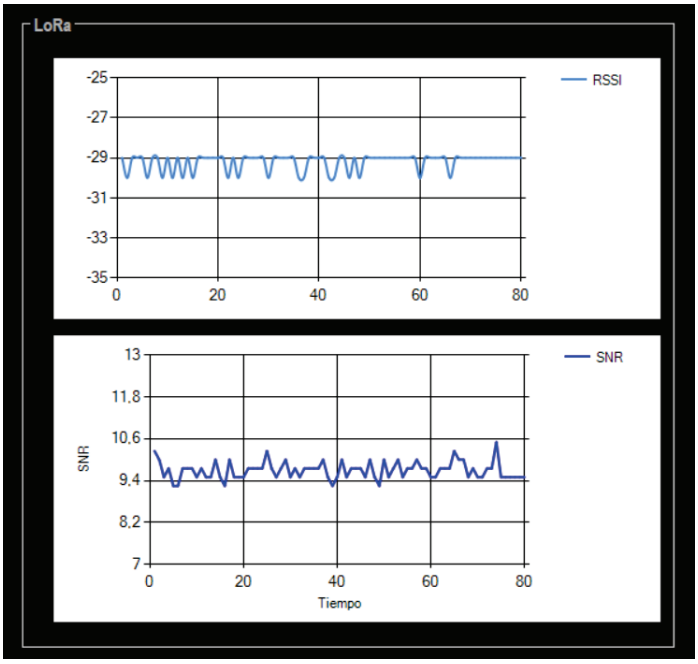


Figura 78: Distancia: 10m - SF: 7 - PTx: 20dBm

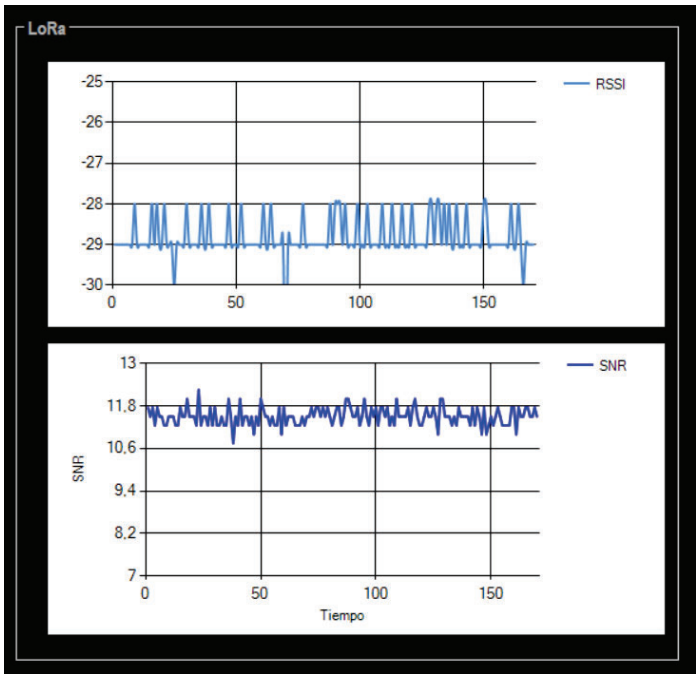


Figura 79: Distancia: 10m - SF: 8 - PTx: 20dBm

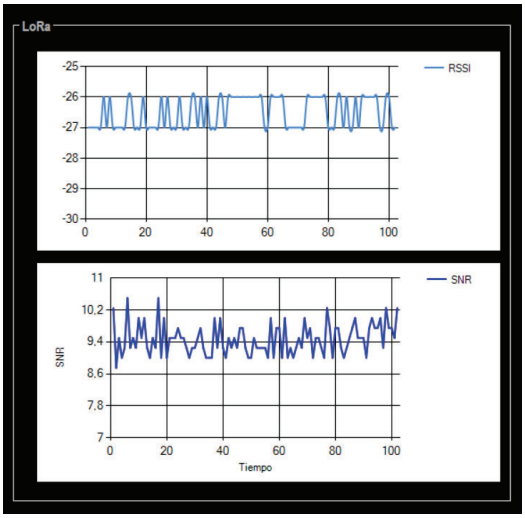


Figura 80: Distancia: 10m - SF: 9 - PTx: 20dBm

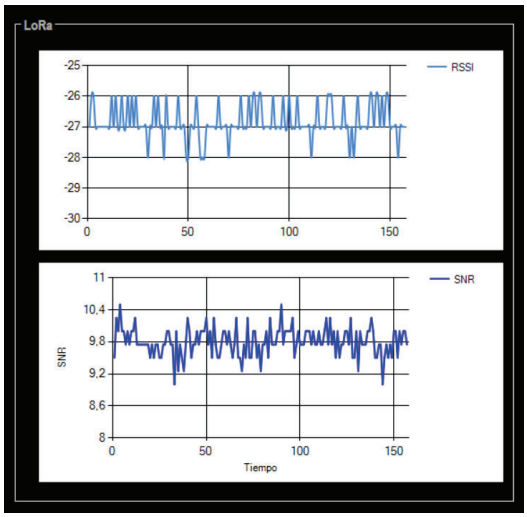


Figura 81: Distancia: 10m - SF: 10 - PTx: 20dBm

Luego de realizar estas pruebas, decidimos pasar a una segunda instancia, en donde la distancia entre el Subsistema de Comunicaciones y la Estación Terrena sea mayor. Utilizamos la configuración LoRa con una frecuencia de 433 MHz, un Spreading Factor de 9, y un ancho de banda de 125 KHz. Decidimos usar esa configuración, ya que observamos que la potencia recibida, la relación señal a ruido y la velocidad de transmisión eran aceptables para nuestro propósito.

Realizamos la prueba a una distancia de 100 metros aproximadamente. Como podemos observar en la figura 82, obtuvimos un nivel de potencia en el receptor de -40 dBm, y una relación señal a ruido de 10 dB aproximadamente. Estos valores concuerdan con nuestro cálculo de enlace, en donde calculamos una potencia de señal recibida de -44dBm, por lo cual, podemos concluir que los cálculos de presupuesto de enlace fueron realizados correctamente.

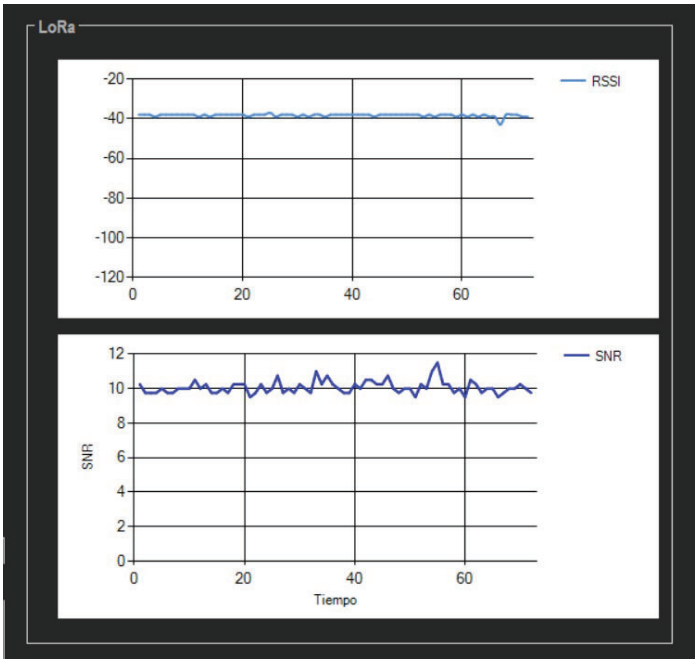


Figura 82: Distancia: 100m - SF: 10 - PTx: 20dBm

También, testeamos el enlace ascendente, es decir, desde la Estación Terrena hacia el nanosatélite. Para eso, programamos nuestro Arduino de la Estación Terrena, y probamos la transmisión de distintos comandos. Primero, verificamos el arribo de los mensajes en nuestro PSC. Para eso, utilizamos uno de los LEDs que trae integrados, para activarlo cada vez que recibiera un mensaje por LoRa. Una vez confirmada la recepción, procedimos a enviar ese mensaje LoRa recibido hacia la POBC. Utilizando la consola de comandos de la estación terrena, verificamos la llegada de los telecomandos transmitidos, mediante los mensajes ACK de confirmación de respuesta por la POBC, como pudimos confirmar según la imagen número 83.

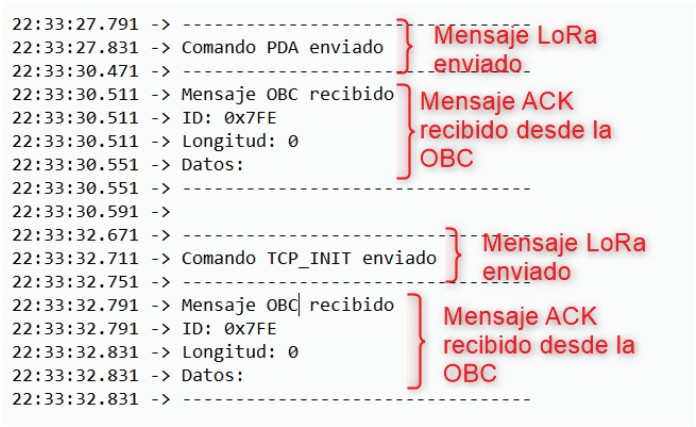


Figura 83: Consola de comandos de la estación terrena

De esta manera, concluimos exitosamente nuestras pruebas sobre la comunicación LoRa, verificando el correcto traspaso de los datos desde la POBC hacia la estación terrena, y verificando el envío de telecomandos desde la estación terrena hacia la POBC.

En esta sección exploramos los distintos testeos realizados, asegurando la funcionalidad, robustez y confiabilidad. Dividida en dos secciones principales, testeamos el firmware de la POBC y los protocolos de comunicación utilizados. Todos los testeos realizados fueron llevados a cabo exitosamente, obteniendo buenos resultados y una gran experiencia al realizar cada una de las pruebas pertinentes.

6. Conclusiones

Al comenzar nuestro proyecto, nos propusimos llevar a cabo el análisis, diseño e implementación de un firmware para una computadora de a bordo de Payload, junto al presupuesto de enlace para un Subsistema de Comunicación de Payload. Esto surgió ante la necesidad del grupo de investigación LabOSat, quienes tenían la necesidad de contar con un firmware para una computadora de a bordo de Payload. Entre alguno de sus requisitos, figuraba que el firmware sea robusto, actualizable, que tenga un sistema operativo, que pudiera controlar una Carga Útil por Ethernet y comunicarse con una placa de control por CAN. En cuanto al cálculo de presupuesto de enlace para un PSC, surgió ante la idea de poder transmitir los datos recolectados por la POBC hacia una Estación Terrena, y de estudiar la tecnología LoRa en las comunicaciones satelitales.

Una vez definidos los objetivos y requisitos, evaluamos diferentes tecnologías, las cuales eran relevantes para poder realizar nuestro proyecto. Entre ellas, estudiamos las arquitecturas de los nanosatélites, cómo se componen y cuáles son los objetivos de cada uno de sus subsistemas. Estudiamos también las computadoras de a bordo, y los distintos sistemas operativos que utilizan, haciendo énfasis en FreeRTOS, adentrándonos en cómo era su implementación y su configuración. Luego nos enfocamos en el manejo de los datos dentro de los nanosatélites y los distintos protocolos que utilizan para comunicarse entre sus subsistemas. Y por último, centramos nuestros estudios en los subsistemas de comunicación que se utilizan para transmitir y recibir los datos, haciendo foco en la tecnología LoRa, y su utilidad en las comunicaciones satelitales.

Luego de estudiar las tecnologías a utilizar, comenzamos con nuestro proyecto. Lo dividimos en dos secciones principales, en donde en la primera sección nos dedicamos al desarrollo del firmware de la POBC, y en la segunda mitad nos enfocamos en el PSC.

Para el desarrollo del firmware de la POBC, comenzamos diseñando la arquitectura principal, de a cuerdo a los requisitos solicitados por el grupo de estudio LabOSat, y utilizando los aprendizajes y estudios obtenidos en las secciones anteriores. Definimos así los modos de funcionamiento que debía tener el firmware, para poder otorgarle robustez. Cada uno de esos modos de funcionamiento debía cumplir con una función especial, y los desarrollamos y planificamos para cumplir con los requerimientos. Luego, implementamos el sistema operativo de tiempo real para su funcionamiento en el Modo Nominal, cumpliendo así con la necesidad que el firmware contara con un sistema operativo. Después de varios análisis, nos inclinamos por el sistema FreeRTOS, el cual era compatible con nuestro kit de desarrollo. Una vez definidos los modos y el sistema operativo a utilizar, procedimos al diseño del firmware y en cómo debía funcionar. Para eso, definimos el funcionamiento del firmware en diagramas en bloques, planificando cada una de las distintas etapas en los modos definidos. Luego, procedimos al desarrollo y programación, realizando distintas pruebas sobre nuestro kit de desarrollo. Hicimos consultas en las comunidades tanto de FreeRTOS como de Texas, que nos fueron de gran apoyo durante toda la etapa de desarrollo.

En cuanto al PSC, nuestro punto de partida fue el diseño funcional, en donde comenzamos con la estructura del sistema. En este proceso, definimos la placa de control a utilizar, y los módulos LoRa que fueron destinados para la comunicación. También, definimos los módulos a utilizar para poder llevar a cabo la comunicación por CAN con la computadora de a bordo de Payload. De esta manera, confeccionamos nuestro prototipo de PSC, y su arquitectura.

Con el diseño funcional completado, nos sumergimos en la fase de desarrollo e implementación del PSC. Dada la simplicidad del diseño, el cual no requería de un sistema operativo ni de distintos modos de funcionamiento, el proceso se simplificó notablemente. Realizamos la integración y su desarrollo según lo previsto, simplificándolo, para realizar una comunicación fluida sin interrupciones. Para culminar, realizamos los presupuestos de enlaces, en donde consideramos el peor escenario posible, es decir, la máxima distancia a la cual podría orbitar un nanosatélite.

Una vez culminado nuestro desarrollo del proyecto, procedimos a realizar los distintos análisis para corroborar su funcionamiento. Al ser un proyecto amplio, que cuenta con distintas etapas y secciones, decidimos dividir las pruebas realizadas en dos secciones principales: la primer sección fue destinada a los testeos sobre el firmware de la computadora de a bordo de Payload, y la segunda sección procedimos a los testeos de los protocolos de comunicación que utilizamos. Para el análisis del firmware de la computadora de a bordo de Payload, realizamos las observaciones de a cuerdo a los distintos modos que habíamos desarrollado. Para el Modo de Inicialización, chequeamos su funcionamiento verificando que cumpla con los distintos saltos hacia los demás modos configurados. En cuando al Modo Nominal, su análisis no fue algo trivial, y debimos utilizar un software especializado en realizar análisis en sistemas operativos. Lo configuramos para poder utilizarlo con nuestro kit de desarrollo, y procedimos a tomar la información brindada por el software sobre este modo desarrollado. De esta manera, pudimos observar y analizar cada una de las tareas que estaban corriendo en nuestro FreeRTOS, y verificar su correcta ejecución. Este paso fue importante para nosotros, ya que aprendimos mucho sobre cómo es que son testeados los sistemas operativos.

En cuando al Modo de Actualización, procedimos a implementar un dispositivo conectado a la computadora de a bordo de Payload, para corroborar que nuestro modo funcione correctamente, grabe y ejecute el archivo binario que se le transmitiera, en la posición de memoria correcta. Ese dispositivo estaba conformado por un Arduino, y módulo CAN y un lector de tarjeta SD. La corroboración de este modo fue de las más complejas, ya que debimos realizar varias modificaciones para que grabe correctamente, y tuvimos que aprender sobre cómo generar un archivo binario, y cómo debe estar modificado para que se pueda grabar y ejecutar en las posiciones de memoria correctas. Para eso, contamos con el soporte de nuestros foros de consulta, quienes nos brindaron gran ayuda para que este modo pudiera funcionar correctamente.

En cuanto a los testeos de los protocolos de comunicación, decidimos analizar cada uno de los protocolos en una sección específica. Comenzamos nuestro análisis

por el protocolo de comunicación CAN, ya que era indispensable para poder continuar con el desarrollo. Para su verificación, realizamos distintas pruebas, comenzando con la comprobación de la funcionalidad de los dos puertos CAN del kit de desarrollo, conectándolos en un bucle, y testeando la comunicación bidireccional, en distintas velocidades, con distintos IDs y mensajes.

Una vez que corroboramos el correcto funcionamiento de ambos puertos CAN, procedimos a comprobar la comunicaciones CAN de nuestro kit de desarrollo contra otra plaqueta. Para eso, armamos un prototipo con un Arduino y un módulo CAN, con el cual verificamos su comunicación. Luego, verificamos la comunicación con nuestro PSC, realizando distintas pruebas, y corroborando su correcto funcionamiento.

La verificación de Ethernet fue algo más compleja, ya que no contábamos con experiencia en codificación de este protocolo, y no había información sobre su implementación con nuestro kit de desarrollo. Luego de exhaustivas investigaciones, decidimos llevar su testeo implementando un stack de bibliotecas Ethernet, el cual fue previamente configurado para que sea compatible con nuestro kit de desarrollo. Para simular la Carga Útil, nos decidimos por utilizar un Servidor TCP en nuestra computadora, en conjunto con un sniffer, poder observar la comunicación por TCP. Realizamos distintas pruebas, corroborando que cada uno de los pasos en la transmisión TCP se cumpliera con éxito.

Por otra parte, para las pruebas de LoRa, procedimos a verificara el correcto traspaso de los datos desde la POBC, pasando por nuestro PSC y llegando hasta una Estación Terrena implementada por módulos LoRa y dos Arduinos. Verificamos la transmisión a distintas frecuencias y con diversas configuraciones de modulación, tomamos los resultados de las potencias y relación de señal a ruido obtenidos, y los comparamos con los cálculos de presupuesto de enlace que habíamos realizado. Comprobamos el correcto traspaso de los datos, obteniendo en nuestra Estación Terrena los valore de potencia de la señal recibida y de la relación señal a ruido. Para eso, implementamos una interface gráfica en donde pudimos observar los datos de la comunicación LoRa.

Completado el diseño y el testeo del firmware de la computadora de a bordo de Payload y del Subsistema de Comunicación de Payload, podemos concluir con que el objetivo de nuestro trabajo se ha cumplido. Este proyecto aportó una serie de beneficios y aspectos positivos tanto a nivel técnico como académico. La implementación y validación para la computadora de a bordo de Payload representó una gran contribución para el grupo de estudio LabOSat, ya que se encontraban en la necesidad del mismo. Se cumplió con los estándares solicitados, los cuales requerían que la POBC cuente con un sistema operativo, que maneje los protocolos de comunicación CAN y Ethernet, y que el sistema operativo sea actualizable. Durante el desarrollo, obtuvimos valiosos conocimientos fundamentales y prácticos sobre el diseño e integración del firmware de las computadoras de a bordo, el cual abarcó desde el análisis inicial hasta la validación del firmware, lo que nos permitió comprender a fondo los requisitos y desafíos asociados. La utilización de los distintos protocolos de comunicación

utilizados amplió nuestros conocimientos sobre ellos, y sus usos y beneficios en el ámbito espacial. Además, abordamos los análisis correspondientes para la utilización de la tecnología LoRa en los nanosatélites.

En cuanto a futuras perspectivas, este proyecto sentó bases para diversas oportunidades. Habiendo realizado la primera versión funcional del firmware de la computadora de a bordo de Payload, nos encontramos muy conformes, ya que se cumplió con los objetivos iniciales. No obstante, siempre hay espacio para futuras mejoras e investigaciones. Por ejemplo, en cuanto a la eficiencia del sistema operativo, queda pendiente realizar un análisis más minucioso en cuanto al consumo de cada una de las tareas, utilizando otros programas de análisis, y disminuyendo los consumos de cada una de las tareas. En cuando a la actualización del firmware, se puede continuar trabajando para hacer un sistema aún más robusto, realizando una tabla de vectores dinámica para poder grabar una imagen binaria en otras posiciones de la memoria, y de esa manera poder tener una redundancia de los datos en memoria. El proyecto es escalable, pudiendo así controlar más Cargas Útiles, haciendo que el Payload cuente con mayores beneficios. Para ello, se debería modificar la tarea vApp, expandir la capacidad y procesamiento de la tarea memoria. En cuanto al Subsistema de Comunicación de Payload, se podría mejorar realizando un prototipo más robusto, utilizando módulos LoRa de mayor potencia, con antenas de transmisión acordes para realizar las pruebas de comunicación de manera más eficientes. Con eso, se podrían llevar a cabo más pruebas para poder analizar la potencia recibida, el alcance logrado, y la corroboración de eficiencia del mismo, que resultarían más exactos para una comunicación satelital.

En conclusión, el diseño y prueba del firmware, junto con el sistema de comunicación, cumplen con el alcance y requisitos esperados, dejando una sólida base para futuras investigaciones y desarrollos.

7. Agradecimientos

Quisiera expresar mi sincero agradecimiento a todas las personas que contribuyeron de diversas maneras a la realización de este proyecto.

En primer lugar, agradezco a mis tutores, Leandro Gagliardi y Carlos Canal, por su experta orientación, paciencia y valiosos comentarios que han enriquecido enormemente este proyecto. Sin ellos, este trabajo no hubiera sido posible realizar.

También quiero reconocer el apoyo brindado por mis profesores y compañeros de clase, cuyas sugerencias y discusiones fueron fundamentales para el desarrollo de este trabajo.

Agradezco especialmente al grupo de estudio LabOSat, donde tuve la oportunidad de colaborar y aprender de profesionales dedicados y apasionados, y a quienes considero un valioso grupo que abre sus puertas a todo estudiando que quiera realizar una investigación.

No puedo dejar de mencionar el apoyo incondicional de mi familia y amigos,

quienes me brindaron aliento y comprensión a lo largo de mi etapa académica.

Finalmente, agradezco enormemente a la Universidad de San Martín, en donde tuve la oportunidad de formarme, capacitarme, y obtener valiosos conocimientos, y gracias a eso poder dedicarme hoy en día a lo que más me gusta, que son las comunicaciones satelitales.

Este logro no hubiera sido posible sin la contribución de cada uno de ustedes. ¡Gracias!

8. Bibliografía

1. Onboard Computers, Onboard Software and Satellite Operations, Jens Eickhoff, 2011.

2. Design and Implementation of Generic Flight Software for a CubeSat, André Emile Heunis, 2014

3. Software Design of an Onboard Computer for a Nanosatellite, Magne Alver Normann, 2016

4. Open Source RTOS Implementation for On-Board Computer (POBC) in STUDSAT-2, Bheema Rajulu, 2016.

5. Firmware Updating Systems for Nanosatellies, Indrek Sünter, Andris Slavinskis, 2016.

6. Design and Analysis of RTOS and Interrupt Based Data Handling System for Nanosatellites, Akshit Akhoury.

7. Development of On Board Computer for a Nanosatellite, Saurabh Raje, 2019.

8. Development of an onboard computer (POBC) for a CubeSat, Tony Lumbw, 2013.

9. Assessing LoRa for Satellite-to-Earth Communications Considering the Impact of Ionospheric Scintillation, IEEE, 2020.

10. A New LoRa-like Transceiver Suited for LEO Satellite Communications, Mohamed Amine, Ben Temim, 2020

11. Store and Forward CubeSat using LoRa Technology and Private LoRaWAN-Server

12. Low-Earth-Orbit satellite communications using LoRa-like signals, Mohamed Amine Ben Temim, 2022.