UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Sistema de detección y corrección de fallas de una computadora de a bordo de un nanosatélite CubeSat

TESIS

Que para obtener el título de

Ingeniero Eléctrico Electrónico

P R E S E N T A

Maximiliano Fragoso Escalante

DIRECTOR DE TESIS

Dr. Saúl de la Rosa Nieves

Ciudad Universitaria, Cd. Mx., 2024

Agradecimientos

Me gustaría agradecer al Dr. Saúl de la Rosa Nieves por su gran apoyo y disposición durante la elaboración de este trabajo de tesis, así como darme la oportunidad de ser parte de este proyecto y ayudarme a crecer profesionalmente.

Por otra parte, me gustaría agradecer a Aldair, Pablo, Luis y Gerardo, por todo el apoyo y disponibilidad bridada durante la elaboración del proyecto.

Me gustaría agradecer a mis padres, Ana y Martin, por su gran apoyo durante toda mi vida, ya que sin ellos nada de esto sería posible.

Me gustaría agradecer a Guillermo, Miguel, Lizbeth y todos los grandes amigos que me acompañaron durante mi carrera universitaria e hicieron de la facultad un lugar donde no solo aprendí, sino que también me reí y disfrute mucho.

Finalmente, pero no menos importante, me gustaría agradecer a Minu, la Sra. Bety, mis hermanos, mis abuelos y todas las personas que me brindaron su apoyo incondicional para poder desarrollarme profesionalmente.

Max

Contenido

Índice de figuras.

Acrónimos

ADC	Analog Digital Converter
ARCADE	Atmosoheric Limb Interferometer for Temperature Exploration
BIS	Built-In-Self-Test
CDS	CubeSat Design Specification
COTS	Commercial Off-The-Shelf
CPU	Control Processing Unit
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
eMMC	embedded MultiMediaCard
ESA	European Space Agency
FeRAM	Ferroelectric Random Access Memory
FPGA	Field Programmable Gate Array
FRAM	Ferroelectric Random Access Memory
GAUSS	Group of Astrodynamics for the Use of Space
GPIO	General Purpose Input Output
I2C	Inter Integrated Circuts (Comunicación serial)
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
INSPIRE	International Satellite Program in Research and Education
IP	Intelectual Property
JTAG	Join Test Action Group
LED	Light Emitting Diode
MCU	Microcontroller Unit
MPSoC	Multiprocessor System on a Chip

NASA	National Aeronautics and Space Administration
OBC	On Board Computer
OS	Operating System
PCM	Phase Change Memory
PFI	Power Fail Indicator
PIC	Programmable Interrupt Controller
PL	Programable Logic
POR	Power-On-Reset
PS	Processing System
PWM	Pulse Width Modulation
QSPI	Quad Serial Peripheral Interface
RISC	Reduced Instruction Set Computing
RTC	Real Time Clock
RTL	Register Transfer Logic
SCMI	Sistema de Comando y Manejo de Información
SEE	Single Event Effects
SEFI	Single Event Functional Interrupt
SEL	Single Event Latchup
SEM	Soft Error Mitigation
SEU	Single Event Upset
SoM	System on Module
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TAP	Test Access Port (JTAG)
JTAG	Join Test Action Group
TI	Texas Instruments

UART	Universal Asycronous Receiver Transmitter
WD	WatchDog
WDT	WatchDog Timer

1 Planteamiento del proyecto

1.1 Introducción

Los satélites de la Tierra son elementos que describen una trayectoria cerrada entorno al planeta, denominada órbita. Dependiendo de su naturaleza los satélites tienen dos clasificaciones: naturales como es el caso de la luna, y artificiales en el caso de los objetos que la humanidad pone en órbita. Normalmente cuando se usa la palabra satélite se refiere a los que el ser humano lanza al espacio. Hay miles de satélites que han sido lanzados al espacio y que orbitan la tierra. Tienen aplicaciones muy importantes dentro de las cuales destacan: estudio del clima en la tierra, estudio de otros planetas y comunicaciones. Sin los satélites no se tendrían predicciones exactas del clima, sistemas de comunicaciones avanzados y el conocimiento del universo sería mucho menor del que se tiene actualmente.

Actualmente ha surgido una filosofía de crear satélites menos costosos y con un tiempo de desarrollo menor, permitiendo que el sector aeroespacial no sea solo para grandes agencias o empresas, acercándolo a instituciones educativas e institutos, esto se logra gracias a la miniaturización de los componentes electrónicos, lo cual ahorra costos. Gracias a esto nacen los nanosatélites lo cuales son satélites con una masa de 1 kg a 10 kg, dentro de los cuales se encuentran distintas clasificaciones: CubeSats, PocketQubes, TubeSats, SunCubes, ThinSats, entre otros.

Los satélites se encuentran en ambientes de radiación donde la electrónica convencional tiene una probabilidad alta de fallos. Los efectos del ambiente espacial en los sistemas espaciales resultan de gran importancia en su diseño. Los meteoritos, cometas y otros fenómenos espaciales han demostrado la presencia de un ambiente radioactivo espacial. Este ambiente puede limitar la operación de los satélites y en circunstancias extremas llevarlo a su pérdida total.

Debido a lo anterior es necesario considerar los efectos de la radiación espacial en la electrónica convencional y realizar el diseño de los satélites con materiales y

componentes electrónicos resistentes a la radiación, o en algunos casos aplicar sistemas de monitoreo, detección y corrección de fallas que permitan recuperar al sistema espacial después de una falla inducida por eventos provocados por la radiación espacial.

El presente trabajo propone el diseño de un sistema de monitoreo, detección y corrección de fallas de una computadora de a bordo de un satélite bajo el estándar CubeSat.

El capítulo 1 de este trabajo describe el *Planteamiento del proyecto,* en donde se determina y define el problema a solucionar, se plantea la hipótesis, los objetivos, justificación y alcance de este trabajo.

El capítulo 2 presenta al *Estado del arte,* donde se hace una revisión exhaustiva de problemas similares al que compete esta tesis y las soluciones que han sido propuestas por empresas privadas, institutos e investigadores. Este capítulo da el fundamento para la solución que se propondrá ante el problema definido.

El capítulo 3 corresponde al *Marco teórico* donde se presentan y explican los antecedentes necesarios que permiten entender el problema, los conceptos y técnicas que serán empleadas para su solución.

El capítulo 4 explica el *Diseño y construcción* donde se aborda ampliamente el proceso de diseño y construcción de la solución propuesta.

El capítulo 5 muestra los *Resultados y conclusiones* donde se analiza que tan efectiva ha sido la solución propuesta y se reconocen las áreas de mejora.

1.2 Presentación del problema.

La electrónica convencional tiene una alta probabilidad de sufrir fallos en ambientes de radiación como el que se encuentra en el medio ambiente espacial, estos fallos pueden llevar a un mal funcionamiento del sistema al que pertenecen, y en el peor de los casos a la pérdida total del satélite. Los eventos provocados por la radiación espacial generan cambios indeseados en la memoria y registros de los

microprocesadores y microcontroladores, alterando así la ejecución del programa cargado en estos dispositivos lo cual genera comportamientos no deseados en las tareas de la misión. Debido a lo anterior es necesario diseñar sistemas de monitoreo, detección y corrección de fallas que ayuden a aumentar la fiabilidad de la misión.

Con el fin de contribuir al desarrollo tecnológico para el incremento de la confiabilidad de misiones satelitales, se propone en el presente trabajo. Se pretende aumentar la fiabilidad de una misión de satélite artificial, implementando técnicas de tolerancia a fallas en la computadora de a bordo del satélite, por medio de redundancia en hardware y software.

Este trabajo de tesis es parte de un proyecto el cual consiste en el diseño de un Sistema de Comando y Manejo de Información de un nanosatélite bajo el estándar CubeSat. De manera general este SCMI emplea dos microcontroladores: un microcontrolador maestro y un microcontrolador supervisor. El microcontrolador maestro es el encargado de controlar la carga útil de la misión: una cámara multiespectral. Por otra parte, se aumenta la fiabilidad de la misión implementando un segundo microcontrolador supervisor el cual tiene como función aplicar técnicas de tolerancia a fallas en el microcontrolador maestro, para así detectar y corregir errores en el funcionamiento de este.

Esta tesis se enfocará en el diseño del sistema de supervisión, desde la selección del hardware hasta la implementación del software.

1.3 Hipótesis.

La implementación de esquemas de supervisión, detección y corrección de fallas mediante una arquitectura supervisora puede aumentar en gran medida el desempeño ante el medio ambiente de radiación espacial al que están expuestas las computadoras de a bordo de nano satélites bajo el estándar CubeSat, particularmente las que se implementan con componentes COTS. La arquitectura supervisora en la computadora de a bordo en conjunto con la implementación del software utilizando técnicas de tolerancia a fallas, permitirá aumentar la

confiabilidad y durabilidad de las misiones espaciales que son realizadas por CubeSats construidos con componentes COTS.

1.4 Objetivos.

Diseñar un sistema de monitoreo, detección y corrección de fallas de una computadora a bordo de un nanosatélite bajo el estándar CubeSat implementando técnicas de tolerancia a fallas a partir de redundancia en hardware y software.

1.4.1 Objetivos particulares.

- Implementar un watchdog inteligente que permita recuperar la computadora de a bordo después de una pérdida total de la misma.
- Implementar un monitoreo periódico que permita corregir fallas parciales de la computadora a bordo. Realizando una reprogramación total de la computadora en caso de ser necesario.
- Implementar un sistema de reconfiguración en vuelo.

1.5 Alcance.

El alcance de este proyecto contempla el diseño de un prototipo de un sistema de monitoreo, detección y corrección de fallas implementando componentes COTS. Aumentando la robustez general del sistema implementando técnicas de tolerancia a fallas por medio de hardware y software.

1.6 Justificación

Las técnicas de tolerancia a fallas por medio de redundancia de hardware y software implementando componentes COTS permiten aumentar la fiabilidad de los sistemas espaciales sin tener que recurrir a componentes de grado aeroespacial y resistentes a la radiación, los cuales son muy costosos en comparación a los componentes COTS, por lo que quedan fuera del presupuesto para proyectos universitarios.

1.7 Metodología.

El éxito de un proyecto se fundamenta gran parte en seguir una buena metodología bien definida en cada una de sus etapas, considerando sus condiciones de inicio y finalización.

El desarrollo de este proyecto en particular seguirá la metodología propuesta por Ulrich en su libro *Diseño y desarrollo de productos, vea* [1].

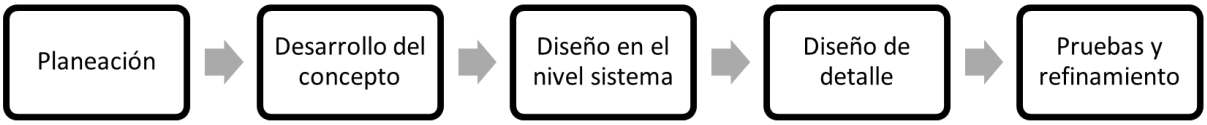

Figura 1. Metodología de diseño.

En la Figura 1 se observa de manera general la metodología de diseño a implementar. Se puede ver que consta de varias etapas, a continuación, se explicará cada una de las etapas, así como sus requerimientos de inicio y fin.

1.7.1 Planeación.

La planeación es elemental en cualquier proyecto ya que fundamenta al mismo y da la pauta para la definición de las siguientes etapas.

La planeación o fase 0 consiste en la revisión del estado del arte para tener una evaluación preliminar de las soluciones propuestas a problemas en la industria similares al que compete esta tesis.

La planeación permite conocer y delimitar al problema, así como dar un primer acercamiento a la solución de este. Al ser la fase inicial no tiene requerimientos de inicio, pero sí requerimientos de fin. A continuación, se presentan los resultados de la planeación:

- Delimitación del problema.
- Declaración de la misión y alcance del proyecto.
- Determinación de los objetivos y especificaciones del producto.

- Determinación de las metas de desempeño.
- Determinación de las limitaciones del proyecto.

1.7.2 Desarrollo de concepto.

En esta etapa se realiza la descripción de la forma, función y características de un producto. El desarrollo de concepto es el punto de partida para el diseño a nivel de sistema. Los entregables de esta etapa son:

- Un conjunto de especificaciones.
- Un análisis de otras propuestas para la solución de problemas similares por medio del estado del arte.
- Una justificación económica del sistema.

1.7.3 Diseño a nivel sistema.

Esta etapa consiste en la definición de la arquitectura general del producto asi como su descomposición en subsistemas y componentes. Se determina como operará el sistema en general y los subsistemas necesarios para lograr estos funcionamientos.

En esta fase se determinan:

- Un diseño a bloques del producto.
- Una especificación funcional de cada uno de los subsistemas del producto. Con esto también se determinan los requerimientos de cada uno de los módulos.

1.7.4 Diseño de detalle.

En esta etapa se definen los componentes a utilizar en cada uno de los subsistemas. Esta definición comprende: la geometría, circuitos, materiales y tolerancias de todas las partes únicas del producto.

Los entregables de esta fase son:

- La documentación de control del producto.

- Las especificaciones de cada uno de los componentes del sistema.
- Los planes de proceso para la fabricación y ensamble de producto.

1.7.5 Pruebas y refinamiento.

Una vez que se tiene un prototipo funcional se realizan pruebas para ver que el funcionamiento sea el adecuado. En caso contrario se determinan los problemas y se reajusta el diseño para solucionarlo.

2 Estado del arte.

El estado del arte es una investigación de un área de conocimiento muy específica con el fin de saber cómo se han llevado a cabo las más avanzadas soluciones ante el problema al que se enfrenta este trabajo de tesis. En esta sección se revisarán distintas soluciones que se han propuesto por parte de la industria o por parte de departamentos de investigación que han implementado técnicas para la corrección de errores en computadoras a bordo de satélites bajo el estándar CubeSat.

Una vez que se hayan revisado las distintas soluciones que se encuentran en el estado del arte se diseñará una solución a la medida de este proyecto, teniendo siempre en cuenta las necesidades de este: requerimientos del medio ambiente espacial, tiempo de diseño, posibilidad de manufactura y financiación posible del costo del proyecto. Se pondrán los conocimientos teóricos en práctica y se contemplarán todos los aspectos que esto conlleva.

2.1 Estado del arte. Sistemas de supervisión y monitoreo de computadoras a bordo.

2.1.1 ISIS On-board Computer

Figura 2. ISIS On Board Computer (IOBC) [2]

De acuerdo con [2] la computadora *ISIS On Board computer (IOBC,* Figura 2*)* ha sido probada en vuelo, tiene una unidad de procesamiento de alto desempeño basada en un microprocesador ARM9 con una velocidad de 400 MHz, haciéndola una de las computadoras de a bordo para nanosatélites más capaces, en un precio de mercado dentro del rango. Su *placa hija* conectable permite tener flexibilidad adicional y personalización ya que se puede tener un gran rango de interfaces extra para cargas útiles, sensores o actuadores en un factor de forma compacto. El diagrama a bloques de la IOBC se muestra en la Figura 3.

Interfaces:

- I2C. Modo maestro o esclavo.
- SPI. Modo maestro, hasta 8 esclavos.
- 2x UART (RS232 + RS232, RS485 / RS422)
- Entradas y salidas de propósito general (GPIO)
- ADC: 8 canales de 10 bits de resolución
- PMW: 6 canales
- JTAG para programación y depuración.
- LEDs y UART dedicados para depuración.
- Puerto USB
- Interfaz para sensor de imagen.

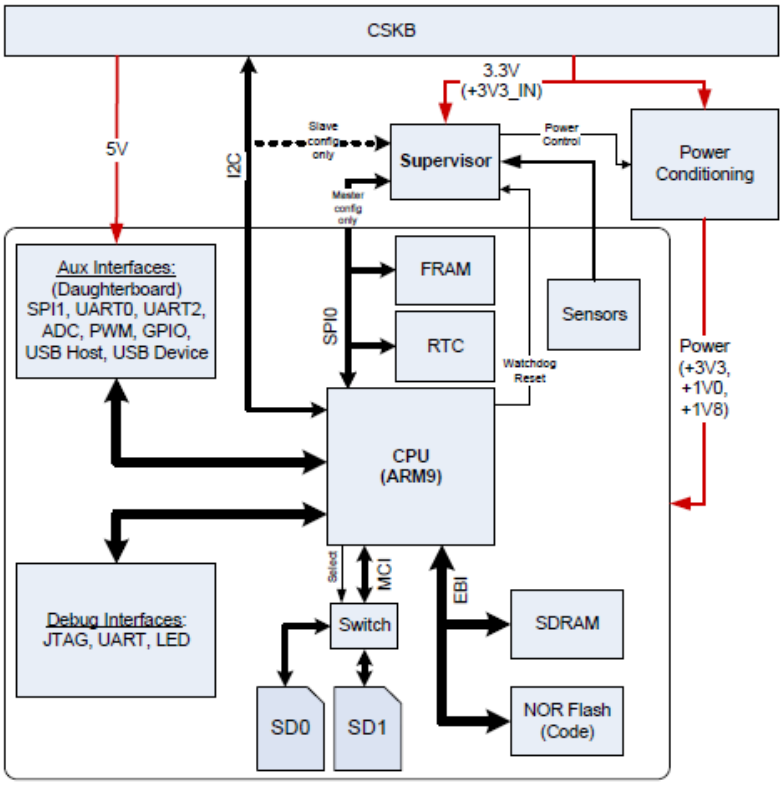

Figura 3. ISIS Onboard Computer - Diagrama de bloques. [2]

2.1.1.1 Supervisor

De acuerdo con [3] se puede ver que el supervisor es un microcontrolador PIC que controla la distribución de energía a través de la tarjeta. Puede apagar o encender individualmente el CPU con los demás componentes de la placa, pero controla individualmente el RTC con el fin de mantener el tiempo del satélite cuando el resto de la computadora de a bordo (OBC, por sus siglas en inglés) se reinicia. El circuito de control de energía incluye una protección para sobre corriente.

El supervisor también funciona como un *watchdog* externo, sin embargo, no es un watchdog convencional, ya que, a diferencia de uno convencional, verifica que la señal de alivio llegué en el instante esperado, no antes ni después, por lo que señal de alivio siempre se tiene que enviar con la misma frecuencia.

El supervisor también tiene diversos sensores para medición de temperatura, voltaje y corriente.

2.1.1.2 Almacenamiento de datos críticos.

La OBC tiene una FRAM de 256kB conectada al CPU a través del bus SPI. La FRAM es una memoria que es más robusta que las memorias FLASH y EEPROM. A diferencia de las celdas de memoria de una SRAM, las celdas de memoria de una FRAM no son susceptibles a fallas debidas a la radiación como lo son los *Single Event Upsets (SEU)*. Por lo tanto, una memoria FRAM se utiliza para almacenar los datos críticos de la misión.

2.1.2 ABACUS OBC

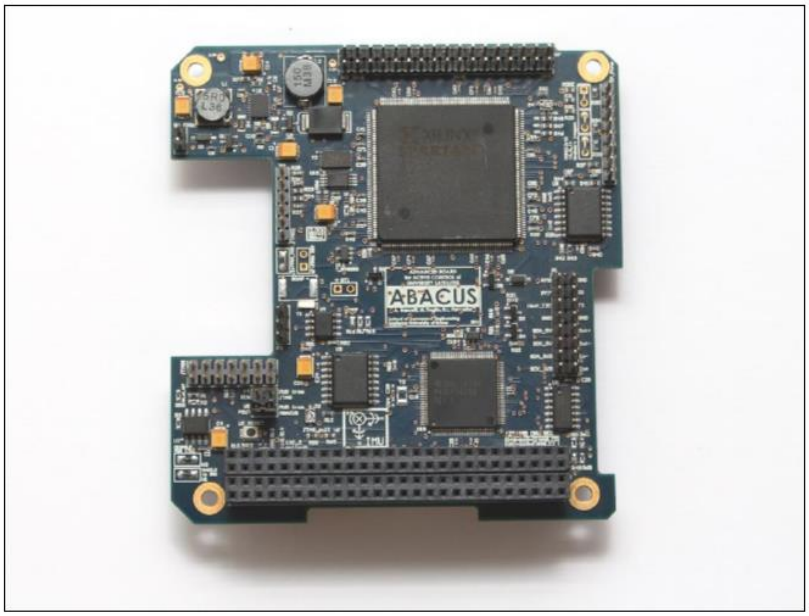

Figura 4. ABACUS OBC. [4]

De acuerdo con [4] se puede ver que la ABACUS 2017 (ver Figura 4) es una unidad OBC con una plataforma de hardware de propósito general, diseñada para una gran variedad de misiones satelitales. Está diseñada para ser flexible y escalable en términos de poder de procesamiento, con el objetivo de mantener un bajo consumo energético. Está compuesta por dos núcleos diferentes, un MCU y un FPGA que trabajan de manera cooperativa.

El MCU de la ABACUS es un MSP430F5438A-EP manufacturado por TI (Texas Instruments). TI provee ejemplos de software con el fin de ayudar a los

desarrolladores a aprender a usar el MCU. Por otra parte, GAUSS provee otro conjunto de librerías que ayudan que la interfaz del usuario con los elementos de la tarjeta sea sencilla, requiriendo muy poco conocimiento del hardware implementado.

Características de ABACUS.

En [5] se puede ver la presencia del microcontrolador MSP430 y el FPGA Spartan-3E, organizado en dos núcleos independientes pero colaborativos, provee al sistema con redundancia en hardware y tolerancia a fallas comunes. Los dos núcleos ofrecen muchas configuraciones para ser implementados (Maestro-esclavo, multimaestro) y el FPGA ofrece las ventajas de la programación a nivel de transferencia de registros (*RTL,* por sus siglas en inglés), para implementar tareas específicas (por ejemplo, control de altitud) o sistemas genéricos utilizando módulos IP de terceros.

El diseño del sistema permite la posibilidad de reconfigurar en vuelo al FPGA y al MCU.

Las principales características de la OBC son:

- Dos núcleos (MCU MSP430 y FPGA Spartan-3E) conectados directamente con un bus de 24 líneas.
- EL MSP430 serie EP es un MCU de 16 bits bajo arquitectura RISC corriendo a una velocidad de hasta 25 MHz. EL MCU de TI soporta aplicaciones espaciales.

El diagrama a bloques de la ABACUS OBC se presenta en la Figura 5

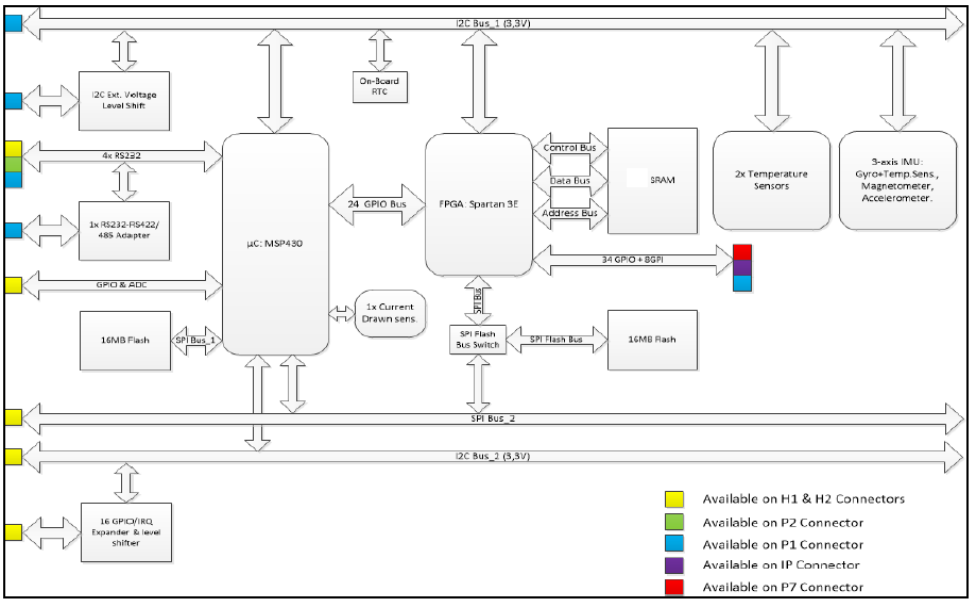

Figura 5. ABACUS OBC - Diagrama de bloques. Imagen obtenida de [5]

2.1.3 Sistema de cómputo tolerante a fallas de un nanosatélite CubeSat desarrollado con un bajo presupuesto.

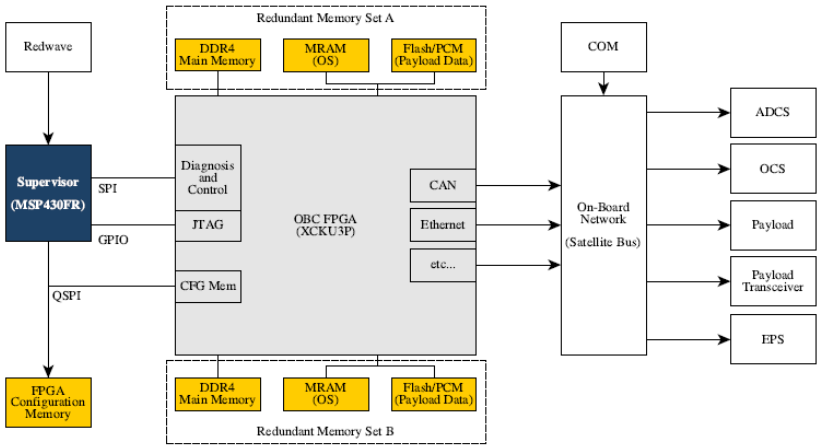

Figura 6. Fuchs OBC - diagrama de bloques. Imagen obtenida de [6]

Este trabajo [6] presenta el diseño de una arquitectura de una OBC compatible con el estándar CubeSat que ofrece una tolerancia a fallas robusta para misiones espaciales a largo plazo (ver Figura 6). La tolerancia a fallas en esta OBC se logra

sin recurrir al uso de componentes de calificación espacial o militar con mayor resistencia a la radiación, sino implementando el software de una manera inteligente.

Dentro del FPGA se ha implementado una arquitectura MPSoC que ofrece un fuerte aislamiento entre los procesadores individuales, y permite recuperarse de fallas permanentes. El FPGA funciona como la plataforma de procesamiento principal para la OBC, y es capaz de correr un sistema operativo OS de propósito general como Linux. Para el desarrollo del concepto se utilizaron los modelos de FPGA *Kintex* y *Virtex Ultrascale+* desarrollados por Xilinx, sin embargo, para el diseño final solo se ocuparán los FPGA *Virtex Ultrascale+* debido a su bajo consumo energético en comparación a los FPGA Kintex Ultrascale+.

La memoria de configuración del FPGA se conecta a él por medio de SPI. El FPGA actúa por defecto como un maestro SPI, y automáticamente carga su código desde ahí. En la implementación de la prueba de concepto, se utiliza una NOR-flash. Sin embargo, este tipo de memorias es inherentemente sensible a la radiación por lo tanto es mejor usar una PCM (Phase-Change Memory) la cual es inmune a la radiación. Por lo tanto, para el prototipo se utilizará un PCM IC en lugar de una NOR-flash serial.

Para implementar la tolerancia a fallas de la OBC, se aísla la ejecución del código dentro de la OBC tanto como sea posible sin limitar el diseño del software. Para realizar esto se ha diseñado un MPSoC como plataforma para la funcionalidad del software. La distribución lógica se presenta en la Figura 7.

Figura 7. Fuchs OBC - Distribución lógica en el MPSoC. Imagen obtenida de [6]

Se ha colocado cada núcleo de procesamiento dentro de un *compartimiento* dedicado. La aplicación y el entorno en el que se ejecutan son aislado fuertemente a través de la topología del MPSoC. A continuación, se explicarán cada uno de los *compartimientos mostrados en la* Figura 7*.*

- Compartimiento 1, 2, 3 y 4. El MPSoC descrito en este trabajo tiene 4 procesadores Xilinx MicroBlaze, y por lo tanto 4 compartimientos los cuales se presentan en café, verde, azul y morado.
- Los *compartimientos* tienen acceso a dos controladores de bancos de memoria a través del bus de alta velocidad embebido en el FPGA. Los dos controladores de los bancos de memoria se presentan en rojo y amarillo.
- El segmento lógico en rosa contiene un módulo IP encargado del mantenimiento del FPGA, así como un controlador de configuración él cual tiene acceso al *Internal Configuration Access Port (ICAP)*.

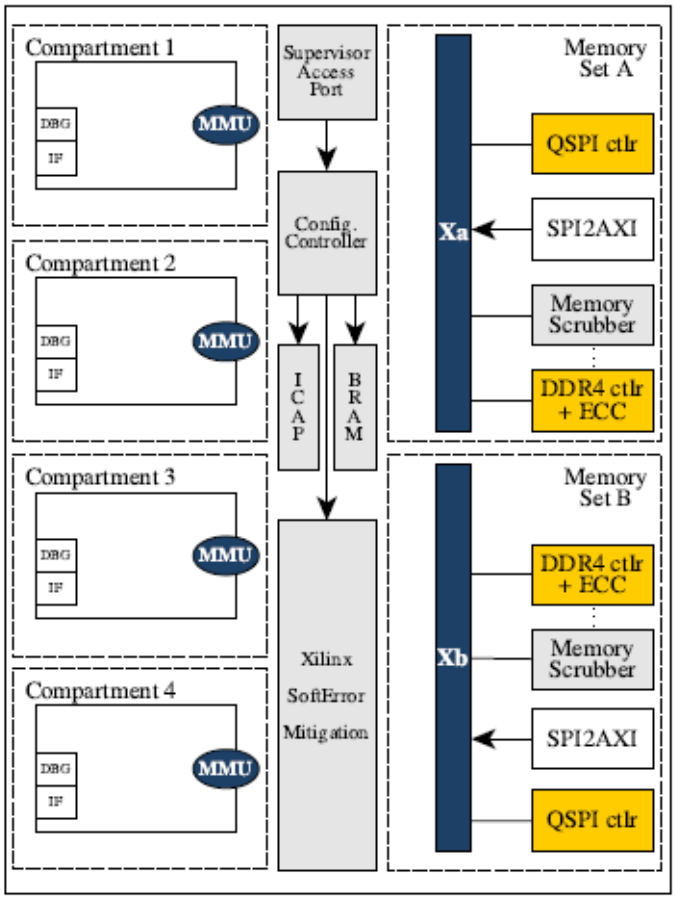

Figura 8. Fuchs OBC - Diseño a nivel de bloques en el MPSoC. Las particiones de reconfiguración se indican con las líneas punteadas. Imagen obtenida de [6]

Como se muestra en la Figura 8, varios componentes del MPSoC relacionados con el mantenimiento del FPGA se colocan en lógica estática:

- El controlador de configuración constituye solo una pequeña parte de la lógica mostrada en rosa.
- La interfaz de depuración del supervisor
- Así como un módulo IP que permite la detección y corrección de errores dentro de la configuración en ejecución del FPGA (Xilinx Soft Error Mitigation IP – SEM).

Usualmente, la reconfiguración total del FPGA interrumpiría la operación del MPSoC, y dependiendo de la configuración de memoria usada, puede tomar un tiempo muy grande. Gracias a usar la reconfiguración parcial, se puede dividir al

MPSoC en diferentes particiones, las cuales pueden ser reconfiguradas individualmente. El uso de un controlador de reconfiguración embebido mejora drásticamente la velocidad.

2.1.3.1 La interfaz supervisor-FPGA.

El supervisor puede acceder al FPGA a través de interfaz JTAG sin embargo los TAP (Test Access Port) pueden ser muy complejos por lo tanto solo se usará para reconfigurar el FPGA en caso de que el controlador de configuración embebido falle.

El supervisor puede disparar una interrupción o deshabilitar un *compartimiento* permanentemente, puede inducir un reset en los *compartimientos,* los bancos de memoria, el controlador de configuración, y el FPGA como tal. Esto se logra a través de interconexión por medio de pines GPIO. El supervisor puede realizar diagnósticos a bajo nivel en cada localidad de memoria, sin tener que depender del procesador de cada compartimiento.

El acceso de alta velocidad se realiza a través de SPI, debido a que la comunidad de CubeSat esta familiarizada con esta interfaz. Se utiliza un puente SPI en cada compartimiento local, y adicionalmente en cada controlador de memoria.

2.1.3.2 Supervisor

Como la mayoría de las OBC de los CubeSat se incluye un microcontrolador adicional que actúa como un *watchdog,* y realiza tareas de depuración y diagnóstico. Sin embargo, debido a que se está utilizando un FPGA como plataforma principal de procesamiento. Solo controla el FPGA y MPSoC implementado. Por lo tanto, actúa como como un subsistema de respaldo (*redwave/hard-command-unit*), y puede resolver fallas dentro del MPSoC, sus IP y periféricos en caso de fallo. Para ilustrar este rol se refiere a él como “supervisor”

El supervisor por sí mismo no está conectado a otros subsistemas del satélite, y no puede controlar otras partes del satélite más allá de la OBC como tal. Durante la operación regular no forma parte del procesamiento de datos realizado por la OBC

solamente recibe información correcta por parte del MPSoC. Sin embargo, en caso de un diagnóstico de fallo el supervisor puede ser usado para reprogramar el FPGA OBC para así poder acceder al resto del satélite con propósitos de depuración. El supervisor requiere poco poder de procesamiento, por lo tanto, se utiliza un microcontrolador MSP430FR5969 el cual es robusto y de bajo desempeño. La familia de microcontroladores MSP430FR se construye con memorias FeRAM las cuales son tolerantes a la radiación. Esta familia de microcontroladores se ha vuelto popular en productos de bajo desempeño de CubeSat COTS debido a su buen funcionamiento bajo radiación en el espacio. También existe un sustituto de grado espacial, MSP430FR5969-SP.

Como se muestra en la Figura 9, el supervisor se encuentra conectado al FPGA a través de GPIO y un bus SPI. La interfaz SPI permite acceso para diagnóstico de bajo nivel a diferentes partes del MPSoC. Así como facilitar el acceso al diagnóstico de los componentes ligados al FPGA. El supervisor a través de GPIO controla la interfaz JTAG del FPGA y puede reiniciarlo, así como a las distintas partes del MPSoC.

El bus SPI proveniente del supervisor llega a cada uno de los compartimientos del MPSoC y a cada uno de los bancos de memoria.

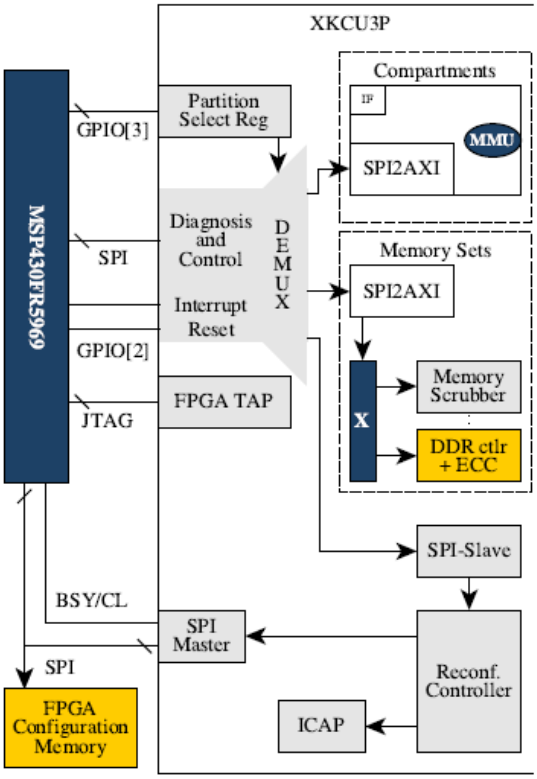

Figura 9. Fuchs OBC - Diseño de la interfaz de diagnóstico y control de la OBC. Imagen obtenida de [6]

El supervisor también se comunica con el controlador de configuración interno al FPGA que está equipado con una interfaz SPI convencional (Maestro-esclavo). El controlador de configuración colabora activamente con el supervisor. El controlador de configuración se comunica con SEM (módulo IP, Soft Error Mitigation) y puede ser desactivado por el supervisor en caso de fallo. Durante la operación normal SEM notificará al supervisor acerca de fallos en el FPGA. El supervisor puede realizar la reconfiguración a través del ICAP. Por lo tanto, el desarrollador de satélites puede depositar distintos diseños para cada partición en la memoria de configuración del FPGA, con los cuales el controlador de configuración puede intentar usarlos para resolver una falla. Finalmente, el controlador notificará al supervisor del resultado del intento de reparación.

En caso de fallo, el supervisor puede sustituir totalmente la funcionalidad del controlador de configuración a través de JTAG, y puede recuperarlo a través la reconfiguración completa del FPGA.

Como se muestra en la Figura 9, el supervisor puede utilizar su interfaz SPI para acceder a cada uno de los distintos componentes del MPSoC en una manera controlada y eficiente. Puede deshabilitar componentes individuales en caso de fallo usando la circuitería existente para la reconfiguración parcial o total del MPSoC como se puede apreciar en la Figura 10. Sin embargo, usar al mismo tiempo las líneas de SPI e interrupciones para cada compartimiento, bancos de memoria y el controlador de configuración requeriría una gran cantidad de GPIO. Por lo tanto, en practica el supervisor solo se comunicará con un componente del MPSoC a la vez, y nunca con varios al mismo tiempo. Se implementa un *DE-MUX* para la interfaz, gracias a esto se reduce la necesidad de puertos I/O solamente a los de la interfaz SPI y 5 líneas GPIO.

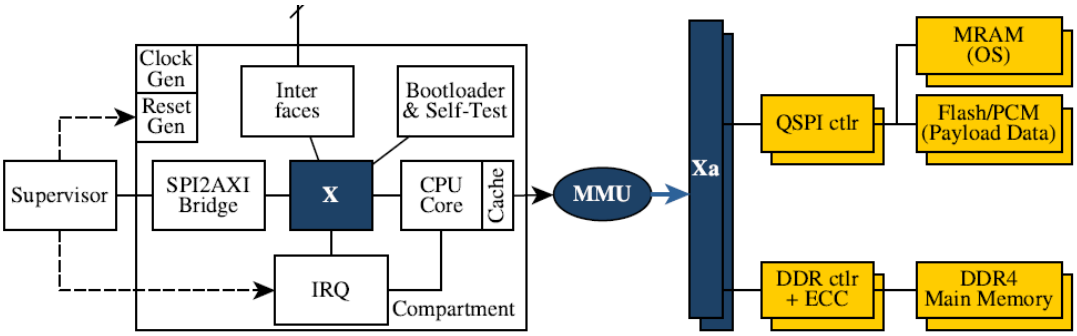

Figura 10. Fuchs OBC - La memoria y topología lógica de un compartimiento. Imagen obtenida de [6]

2.1.4 Reconfiguración en vuelo de arquitecturas basadas en SoM para instrumentos científicos en nano satélites

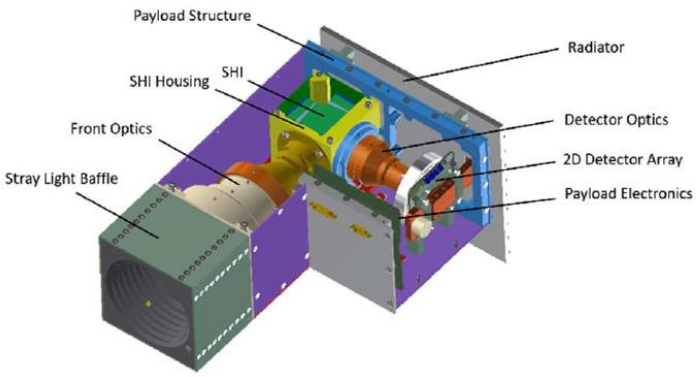

Figura 11. AtmoLITE - Componentes internos. Imagen obtenida de [7]

La Sonda Atmosférica de Interferometría para la Exploración de Temperatura [7] *(AtmoLITE,* por sus siglas en inglés) es una sonda extremadamente miniaturizada, para la medición de la temperatura de la atmosfera en el día y la noche. Fue desarrollada como una carga útil para la misión de Exploración de Acoplamiento y Dinámica Atmosférica *(*ARCADE, por sus siglas en inglés*)* programada para el cuarto trimestre del 2024 dentro del programa Internacional de Satélites para la Educación e Investigación (INSPIRE, por sus siglas en inglés). Los perfiles de temperatura atmosférica se obtienen por emisiones ópticas. El AtmoLITE es un instrumento de medición remota que contiene óptica, detector, electrónica, estructura y un radiador. La Figura 11 da un panorama general del diseño y los componentes del *AtmoLITE.*

El hardware está basado en el SoM *Xilinx Zynq-7000* con una cantidad mínima de componentes externos. Se implementa un circuito supervisor que permite inicializar el *System On Module (SoM)* desde dispositivos de memoria alternos y redundantes, también permite iniciar la reconfiguración del SoM debida a cierto evento. Esta implementación permite actualizaciones de firmware durante el vuelo, en caso de que haya errores debidos a las condiciones de la medición.

La reconfiguración de la OBC se puede realizar en escenarios de tierra y vuelo, para el primer caso el SoM puede ser configurado en fases posteriores al diseño del sistema espacial, para el otro caso, puede ser reconfigurado después del lanzamiento. Adicionalmente en caso de que se necesite el SoM se puede reconfigurar con una aplicación diferente a la de fábrica.

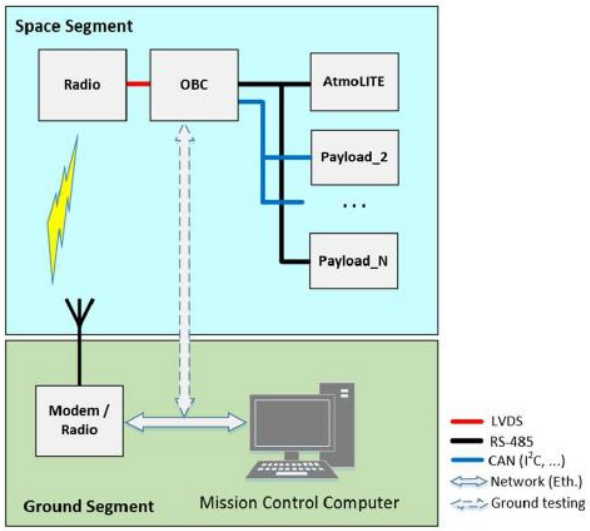

Figura 12. AtmoLITE - Diagrama a bloques de la comunicación para la reconfiguración en vuelo. Imagen obtenida de [7]

La estrategia de reconfiguración en vuelo y en tierra se muestra en la Figura 12. La OBC se encarga de la comunicación con la estación ya sea en tierra o en vuelo. En el caso de la reconfiguración en vuelo se puede conectar con la estación de control por medio de una banda de radiofrecuencia. En el caso de la reconfiguración en tierra, la OBC se puede conectar directamente (por medio de cables).

2.1.4.1 Estrategia de reconfiguración.

La estrategia de mitigación de errores está enfocada en la reconfiguración en tiempo real para corregir errores debidos a eventos SEU, y un apagado seguro ante eventos SEL. Se realiza un diagnóstico disparado por eventos para la reconfiguración completa por medio del reinicio de la OBC. La parte de diagnóstico consiste en un módulo de autoverificación embebido (BIST, por sus siglas en inglés) que monitorea el software y hardware implementado.

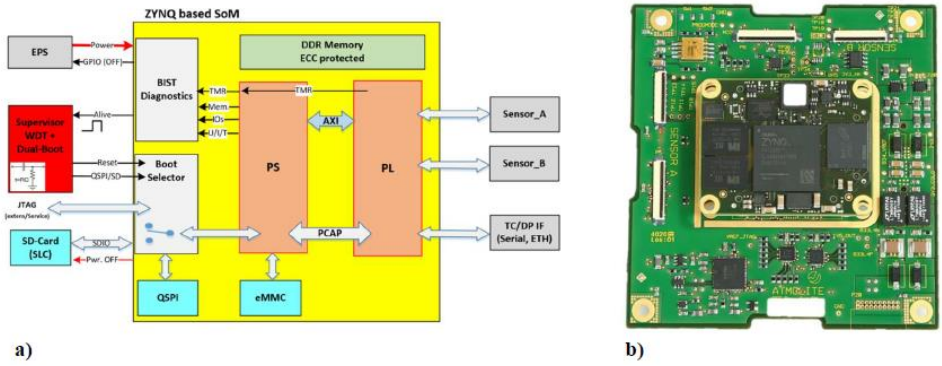

Figura 13. Diagrama a bloques del AtmoLITE. Imagen obtenida de [7]

En la Figura 13 se puede observar el diagrama a bloques del AtmoLITE, del lado izquierdo se pueden observar las interfaces del supervisor con el SoM, se puede ver que esta interfaz está formada por:

- Una señal de alivio.
- Una señal de reset
- Señales QSPI/SD que determinarán de que memoria se inicializará el proceso de arranque.

2.1.4.2 Rol del supervisor

El sistema supervisor verifica la operación y reconfiguración del controlador del sistema. Para este propósito, es disparado por el BIST continuamente. En caso de fallo del software del Sistema de Procesamiento (PS, por sus siglas en inglés) o de un fallo de hardware adentro de la lógica programable (PL, por sus siglas en inglés), el supervisor reinicia el sistema. Se utiliza el microcontrolador MAX706, el cual está disponible comercialmente con varios fabricantes y es resistente a la radiación.

El supervisor tiene dos bloques funcionales internos los cuales se muestran en la Figura 14. Los dos bloques son los siguientes:

- POR y WDT.
- PFI.

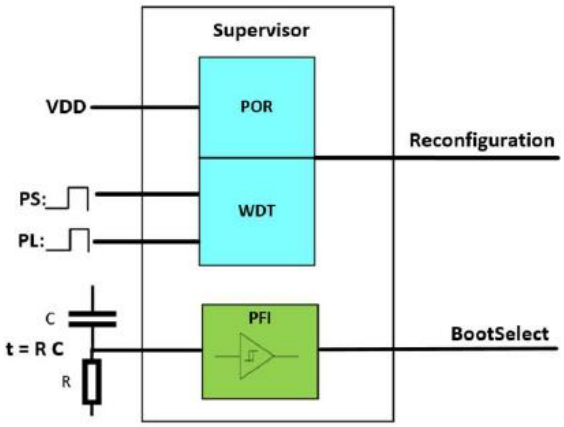

Figura 14. AtmoLITE - Bloques funcionales del circuito supervisor. Imagen obtenida de [7]

El primer bloque está compuesto por un monitor del voltaje de alimentación y un watchdog el cual genera un reset después de 1.6 s en caso de que falte el pulso de alivio generado por el BIST el cual lo suprimirá en caso de una falla.

Por lo tanto, el watchdog WDT comienza el proceso de reconfiguración cuando se realiza un diagnóstico por el BIST y se detecta una falla en los siguientes elementos:

- PL: Cuando se detecta una falla triplicada en 3 bloques.
- PS: Cuando un bit ha cambiado en la memoria en los paquetes de datos que resultan incoherentes con el CRC. Cuando se presentan banderas de error en las interfaces de periféricos como I2C, SPI y DMA. Cuando se detectan anomalías en el vector de configuración.

El segundo bloque principalmente se usa como un Indicador Temprano de Fallas en la Alimentación (*PFI,* por sus siglas en inglés). Se utiliza al PFI para generar una señal que seleccione entre arrancar al FPGA desde la memoria principal o secundaria. El PFI utiliza en un comparador de voltaje, el cual genera como salida una señal digital basada en una señal analógica de entrada la cual es entregada por un circuito RC, en caso de que la memoria principal de arranque haya sido corrompida el SoM no generará una señal de alivio y, por lo tanto, la señal *boot select* se encontrará en estado bajo debido al circuito RC cargado y, por lo tanto, se seleccionará la memoria redundante como dispositivo de arranque.

2.1.4.3 Monitoreo y corrección de fallas en el respaldo del código redundante cargado en las memorias de arranque.

La imagen de arranque se almacena en tres memorias de distinta tecnología para tener redundancia triple del código de arranque. Las tecnologías de la memoria se presentan a continuación:

- QSPI (memoria primaria).
- SD-Card (memoria redundante)
- eMMC memoria de transferencia y “Golden image”

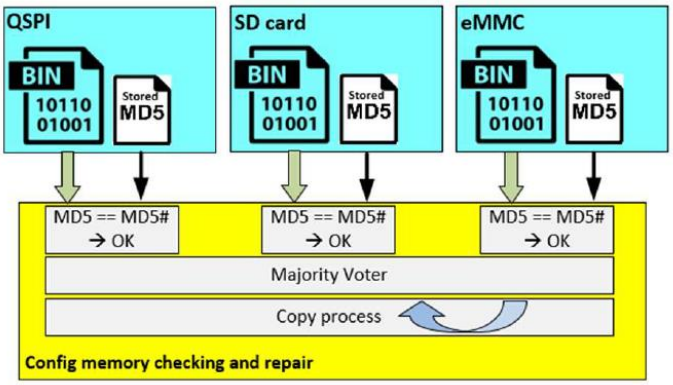

Figura 15. AtmoLITE - Memorias redundantes para el almacenamiento del Firmware y reparación. Imagen obtenida de [7]

Cada memoria contiene la imagen de arranque y una etiqueta única MD5 en un archivo separado. El MD5 es una función hash criptográfica y será usada como suma de verificación para verificar la integridad de los datos. La memoria eMMC es usada para cargar y transferir la imagen de arranque a las otras memorias, en una reconfiguración en vuelo los bloques de datos recibidos se almacenan en la eMMC hasta que la actualización de firmware es completada y revisada. En la Figura 15, la eMMC siempre tiene una imagen de comparación del firmware para las dos memorias de arranque.

Para asegurar que todas las memorias tienen el mismo firmware, se realiza una validación del contenido de las memorias.

Cuando se enciende el sistema, el software llama a un procedimiento llamado *verificación de sistema*. En primera instancia la *verificación de sistema* compara al firmware almacenado en la memoria contra su respectivo MD5, así se corrobora que los dispositivos de memoria operan de manera adecuada, y los datos son consistentes. Para el segundo paso, las etiquetas MD5 de las copias del firmware se comparan entre sí.

En caso de que exista una discrepancia entre las tres memorias de configuración, se inicia un proceso de reparación. El cual se muestra en la Figura 16.

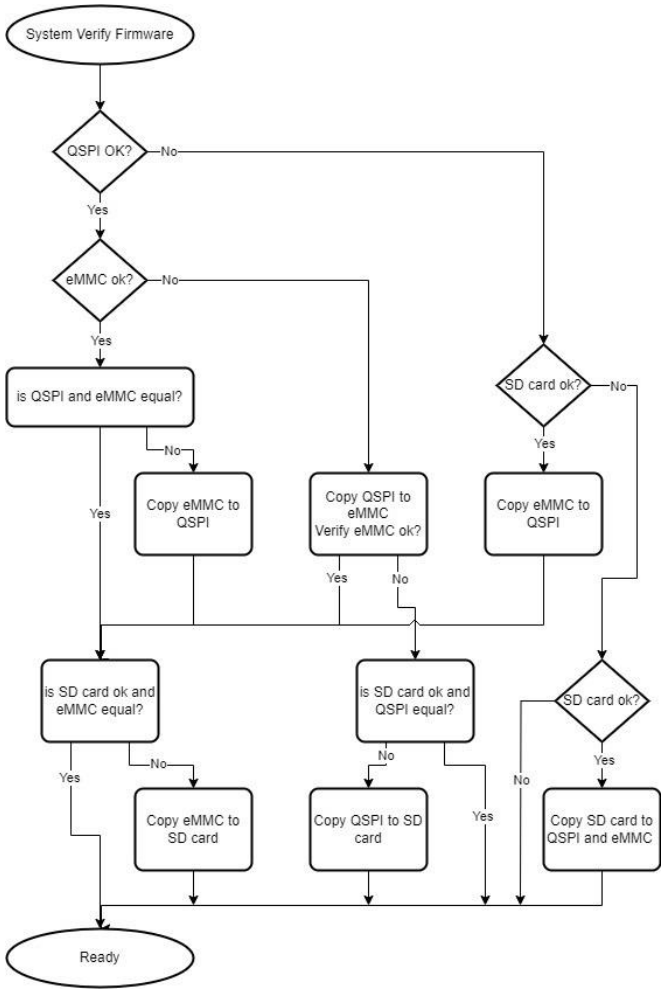

Figura 16. AtmoLITE - Proceso de verificación y corrección del firmware almacenado en las memorias redundantes. Imagen obtenida de [7]

2.2 Conclusiones.

En esta sección se han revisado las más avanzadas soluciones publicadas que podrían resolver el problema que compete a esta tesis. En 2.1.1, 2.1.2 y 2.1.3 se ha visto que la tolerancia a fallas se lleva a cabo por medio de un esquema de supervisión donde en todos los casos se utiliza un microcontrolador como arquitectura supervisora. Por otra parte, es interesante notar que en 2.1.2 y 2.1.3 se utiliza un microcontrolador de la serie MSP430FR fabricado por Texas Instruments, este microcontrolador podría resultar una buena opción para este proyecto de tesis.

En 2.1.4 se ha observado como se ha aumentado la fiabilidad de las memorias redundantes utilizando un algoritmo de verificación del contenido por medio de etiquetas criptográficas HASH. Las etiquetas HASH permiten tener una huella digital de una longitud determinada a partir de un vector de datos de entrada prácticamente de cualquier tamaño lo cual resulta útil para verificar si los contenidos de memoria no han sido alterados. Las propiedades de las etiquetas HASH resultan una gran herramienta a considerar en este trabajo de tesis, sin embargo, hay algunos otros elementos como el CRC, los cuales pueden cumplir la misma función.

3 Marco teórico.

3.1 Satélites

De acuerdo con [8] los satélites son cualquier elemento que orbite un planeta o una estrella, pueden ser lunas, asteroides o máquinas lanzadas por el ser humano. Por ejemplo, el planeta tierra es un satélite porque orbita al sol, y la Luna es un satélite porque orbita la Tierra. Los satélites se clasifican en dos grandes tipos:

- Naturales. (vea Figura 17)
- Artificiales.

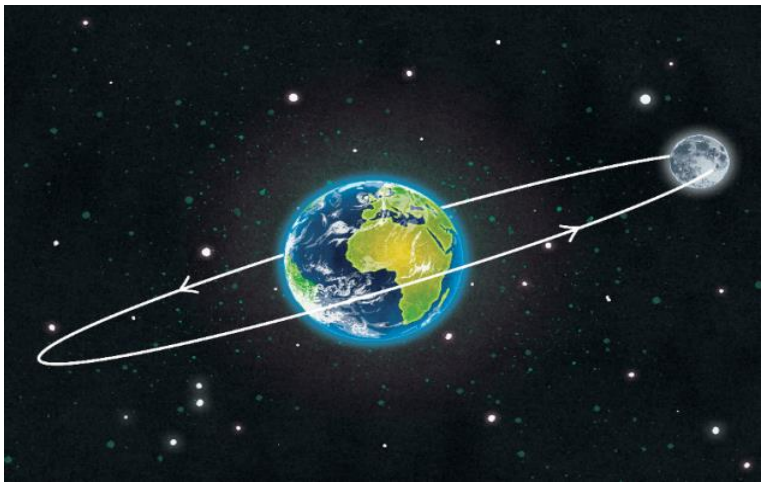

Figura 17. satélites naturales. (Luna orbitando al planeta tierra). Imagen obtenida de [8]

Los satélites naturales son cualquier cuerpo celeste que orbite un planeta o una estrella y no han sido creados por el ser humano.

Por otra parte, los satélites artificiales son máquinas lanzadas por el ser humano los cuales orbitan principalmente el planeta Tierra, y tienen distintas funciones desde exploración del espacio, estudio del clima o del cambio climático y telecomunicaciones, solo por mencionar algunas.

El primer satélite artificial en la historia de la humanidad fue Sputnik 1 y fue lanzado por la ex unión soviética el 4 de octubre de 1957. Desde ese día y hasta la actualidad se han lanzado miles de satélites que orbitan la tierra, de distintos tamaños y con distintas aplicaciones. Algunos se utilizan para el estudio de otros planetas,

agujeros negros, galaxias y en general del universo, algunos otros se utilizan para telecomunicaciones, canales de televisión, llamadas telefónicas, sistemas GPS, entre otros. En la Figura 18 se puede observar al satélite artificial Jason-2 el cual ha sido lanzado por la NASA para el estudio de los océanos.

Figura 18. El satélite artificial Jason-2. Imagen obtenida de [8]

Gracias a la vista de la Tierra que se posee desde los satélites es posible visualizar grandes áreas. Esto permite que se obtengan más datos de lo que se obtendrían con instrumentos instalados en la tierra. Los telescopios instalados en satélites tienen una mejor vista del espacio exterior que los telescopios instalados en Tierra. Los satélites también han representado una gran mejora en el mundo de las telecomunicaciones ya que antes de que estos existieran las señales telefónicas no podían ser transmitidas a grandes distancias.

Hay muchos tipos de satélites y dependiendo de las características de la misión será necesario definir los componentes de este, sin embargo, los siguientes componentes son comunes en la mayoría de los satélites:

- Subsistema de telecomunicaciones: Están compuestos por antenas, transmisores y receptores. Estos sistemas son utilizados para enviar y recibir señales hacia y desde la Tierra.
- Subsistema de potencia: El sistema de potencia de un satélite es el encargado de proveer el voltaje a cada uno de los componentes eléctricos y electrónicos que forman parte del satélite. La energía que el satélite requiere es obtenida a través de uno o más paneles solares.
- Subsistema de manejo y comando de información: Este sistema es el encargado de monitorear cada aspecto del satélite.

- Subsistema de control de orientación: Por medio de sensores monitorea la posición del sensor para ver que se encuentra en la posición correcta y cuenta con actuadores que permiten su estabilización y su orientación.
- Estructura mecánica: La estructura mecánica consiste en el contenedor de todos los componentes del satélite, los materiales deben ser capaces de soportar el medio ambiente espacial.

3.1.1 Órbitas

Se le denomina orbita a la trayectoria que sigue un satélite alrededor de un cuerpo, el cual puede ser un sol, planeta, luna, entre otros. Las órbitas existen gracias a la fuerza gravitacional que ejerce un cuerpo a otro en el espacio. Dependiendo de las características de la misión del satélite puede ser lanzado a una de las órbitas alrededor de la tierra presentadas en la Tabla 1.

Tabla 1. Clasificación de las orbitas de la Tierra. [9]

Orbita	Aplicaciones	Características
LEO (Low Earth Orbit)	• Observación de la tierra. • Monitoreo del clima. • Tecnología. • Astronomía. • Comunicaciones	Altitud de 300 a 1500 Km
MEO (Medium Earth Orbit).	• Comunicaciones. • Navegación.	Se encuentra entre LEO y GEO.
GEO, (Geoestationary Orbit)	• Comunicaciones.	35 786 Km por encima del nivel del mar.

A diferencia de GEO, las trayectorias en LEO no tienen que estar alrededor del ecuador de la Tierra, pueden tener cierta inclinación lo que permite que haya más rutas posibles en LEO, gracias a esto LEO es una órbita muy usada para lanzamientos de satélites. La proximidad de LEO a la tierra la hace muy útil para varias tareas, es la órbita más utilizada para aplicaciones de imagenología ya que su proximidad a la Tierra permite tomar fotografías de gran resolución.

3.1.2 Clasificación de satélites según su masa.

De acuerdo con [9] los satélites se clasifican según su masa de la siguiente manera:

- Satélites grandes: >1000 kg.
- Satélites medianos: de 500 a 1000 kg.
- Satélites pequeños: <500 kg.
    - Minisatélites: de 100 a 500 kg.
    - Microsatélites: de 10 a 100 kg.
    - Nanosatélites: de 1 a 10 kg.
    - Picosatélites: de 100 g a 1 kg.
    - Femtosatélites: de 10 g a 100 g.
    - Attosatélites: de 1 g a 10 g.
    - Zeptosatélites: de 0.1 g a 1 g.

3.1.3 CubeSats

De acuerdo con [9] un nanosatélite es cualquier satélite con una masa de 1 a 10 kg. Algunas de las clasificaciones de los nanosatélites son: CubeSats, PocketQubes, TubeSats, SunCubes, ThinSats, entre otros.

Los CubeSats son un tipo de nanosatélite definidos por la Especificación de Diseño de CubeSats Las dimensiones de los CubeSats son las siguientes:

- 1U CubeSat: 10 cm x 10 cm x 11.35 cm.
- 2U CubeSat: 10 cm x 10 cm x 22.70 cm.

- 6U CubeSat: 20 cm x 10 cm x 34.05 cm.
- 12U CubeSat: 20 cm x 20 cm x 34.05 cm.

Los CubeSats han revolucionado la industria aeroespacial, haciendo que el acceso a esta industria sea más fácil y sobre todo más barato. Inicialmente fueron desarrollados con motivos educativos, sin embargo, recientemente han sido utilizados para estudios científicos e incluso aplicaciones comerciales. Se pueden personalizar para cumplir con los requerimientos específicos de cada misión.

Los CubeSats pueden ser lanzados en los espacios vacíos de cohetes con satélites más grandes. Esto significa que se tienen muchas oportunidades de lanzamiento y los costos son relativamente bajos.

Estos satélites pequeños proveen de un acceso asequible para compañías aeroespaciales pequeñas, institutos de investigación y universidades. Su diseño modular permite acoplar el diseño según los requerimientos de la misión y además de esto el tiempo de desarrollo es muy rápido (de uno a dos años).

Quizá lo más emocionante en términos científicos es que los CubeSats permiten tener una gran versatilidad en la exploración espacial.

A manera de motivación se presentan algunas misiones espaciales de la Agencia Espacial Europea (ESA, por sus siglas en inglés) donde se utilizan CubeSats.

Misión Hera [10]**:** Su lanzamiento ha sido programado en 2023. Hera es la contribución por parte de la ESA para la colaboración una misión que tiene como objetivo estudiar el impacto de asteroides (AIDA, por sus siglas en inglés). La misión DART (misión espacial de la NASA destinada a probar un nuevo método de defensa

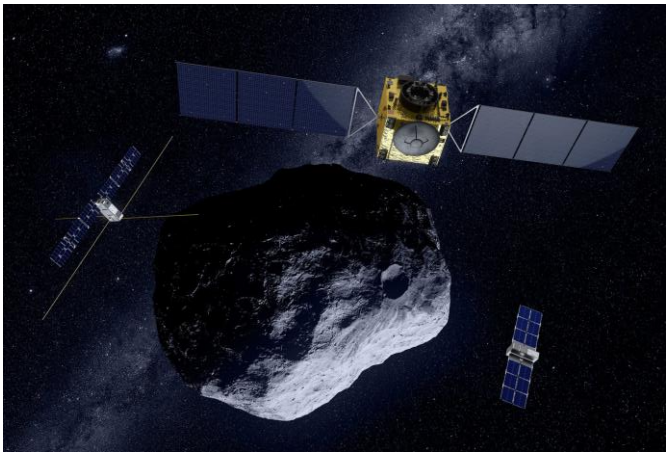

Figura 19. ESA - Misión Hera llevada por CubeSats. Imagen obtenida de [10]

planetaria contra objetos próximos a la Tierra) impactará a la luna de Didymos (Sistema binario sincrónico) y Hera estudiará el efecto del impacto de este tipo de asteroides. Hera tiene 3 objetivos:

- Probar la tecnología en el espacio interplanetario.
- Investigar técnicas de mitigación de objetos cercanos a la tierra y que presenten un posible riesgo. (desviar asteroides).
- Obtener datos para comprender la evolución del sistema solar.

Hera es una misión que consiste en dos CubeSats 6U, y es la primera oportunidad de la ESA de trabajar con CubeSats en el espacio exterior lejano.

SpectroCube [10]**:** El ESE pronto lanzará SpectroCube, una misión que viajará lejos de la Tierra para llevar a cabo experimentos de astrobiología y astroquímica. Los objetivos de SpectroCube son evaluar el impacto del espacio en la biología y la química de los componentes básicos de la vida.

3.1.4 Computadoras de a bordo de satélites (OBC)

Un nombre alternativo para las computadoras de a bordo (OBC, por sus siglas en inglés) es Sistema de Comando y Manejo de Información (SCMI) el cual describe de mejor manera su función. La OBC es el cerebro del satélite, tiene muchas funciones, pero destacan las siguientes:

- Control del satélite.
- Procesamiento asociado a la carga útil.
- Procesamiento de datos asociado con los sistemas de comunicación.
- Almacenamiento de datos.

Una OBC es fundamental para un satélite ya que sin ella este no puede operar. Una OBC está formada por varios módulos de hardware y software.

Los bloques de hardware de una OBC varían de misión a misión siempre atendiendo los requerimientos de esta, sin embargo, de acuerdo con [11] algunos bloques comunes son los siguientes:

- Unidad de procesamiento, en la mayoría de las misiones se utilizan microcontroladores o FPGA.
- Memorias volátiles y no volátiles.
- Relojes en tiempo real.
- Controladores de comunicación como SPI, UART y I2C.
- Módulos de reconfiguración.

El software de una OBC se puede dividir en dos grandes grupos:

- Software relacionado al control del satélite.
- Software relacionado al control de la carga útil.

3.2 El entorno de radiación espacial.

Los satélites se encuentran en ambientes de radiación espacial donde la electrónica convencional tiene una probabilidad alta de fallos, por lo tanto, es necesario desarrollar técnicas o tecnologías que permitan elevar la fiabilidad de los satélites y los sistemas espaciales en general.

Los efectos del ambiente espacial en los sistemas espaciales son de vital importancia en su diseño. Los cometas, meteoritos y otros fenómenos extraterrestres han demostrado la presencia de un “ambiente espacial”. Este ambiente puede limitar la operación del aparato espacial y en las circunstancias más extremas llevarlo a su pérdida

3.2.1 Cinturones de radiación de Van Allen

De acuerdo con [12], Estados Unidos lanzó en 1958 su primer satélite el *Explorer I.* Los datos capturados por el contador Geiger (instrumento para medir la radiación) dieron pie al surgimiento del estudio de la física espacial y por lo tanto comenzó una era nueva en la tecnología espacial.

Los datos de radiación obtenidos por el *Explorer I* fueron el primer acercamiento a los cinturones de radiación de la tierra, dos anillos concéntricos de partículas altamente cargadas rodeando el planeta. El cinturón interior está compuesto

principalmente por protones, mientras que el cinturón exterior está compuesto principalmente por electrones. Estos cinturones fueron llamados cinturones de radiación de Van Allen, después de que James Van Allen estudiara los datos recopilados por el *Explorer I.*

El anillo exterior está compuesto de millones de partículas energéticas altamente cargadas que se han originado debido al viento solar y han quedado atrapadas debido al campo magnético de la tierra. El anillo interior resulta de la interacción de los rayos cósmicos con la atmosfera terrestre. Los satélites que orbitan en esta área pueden ser afectados por la radiación.

En Figura 20 se puede observar la primera aproximación obtenida en 1958 de los cinturones de radiación de Van Allen, esta aproximación consiste en dos cinturones: el interior y el exterior. Más adelante la NASA obtuvo una aproximación mucho más acertada.

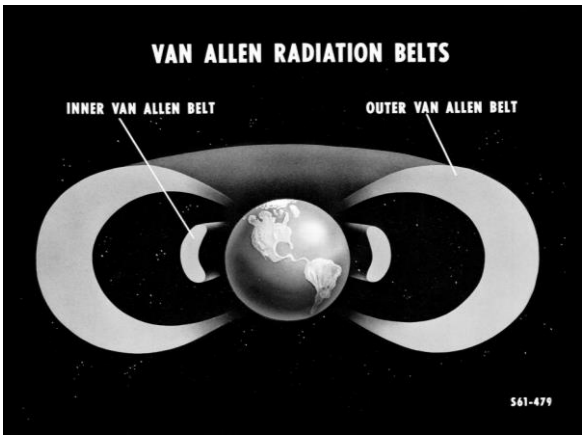

Figura 20. Primera aproximación a los cinturones de radiación de Van Allen obtenidos en 1958. Imagen obtenida de [12]

En 2012, la NASA lanzó las sondas gemelas Van Allen para estudiar el comportamiento de las partículas en la región dinámica. Equipadas con tecnología resistente a la radiación, los instrumentos implementados fueron más sofisticados que los del *Explorer I,* lo que permitió estudiar de una mejor manera las partículas, ondas y campos contenidos en los cinturones de radiación de Van Allen.

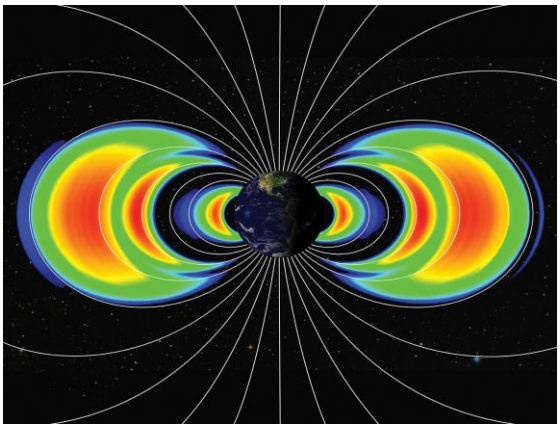

Figura 21. Aproximación de los cinturones de Van Allen obtenidas por la NASA con las sondas gemelas Van Allen en 2012. Imagen obtenida de [12]

Después de algunos días de su lanzamiento las sondas descubrieron un tercer cinturón temporal, el cual se encontraba entre los cinturones interior y

exterior. El tercer cinturón solamente duró un mes, pero después apareció de nuevo en la misión cuando la actividad solar fue mayor. Vea Figura 21.

3.2.2 Rayos cósmicos

De acuerdo con [13] La radiación o rayos cósmicos son partículas subatómicas altamente cargadas acompañadas de emisiones electromagnéticas que impactan con la superficie terrestre y las orbitas de la Tierra.

Lo rayos cómicos se presentan en dos tipos: solares y galácticos. (Vea Figura 22)

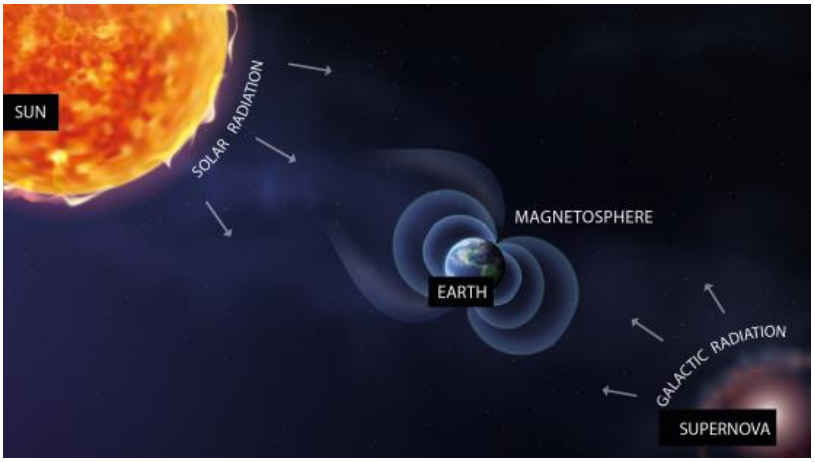

Figura 22. Radiación cósmica. Imagen obtenida de [13]

3.2.2.1 Partículas solares

Las erupciones solares están caracterizadas por una liberación muy grande de energía (usualmente en forma de rayos X) en pocos minutos. Las partículas solares que se deben a estas erupciones incrementan en gran medida el flujo de partículas energéticas y pueden durar desde unas cuantas horas hasta varios días. A pesar de que ocurren pocas veces al año los efectos que provocan en los sistemas espaciales tienen grandes consecuencias.

3.2.2.2 Rayos cósmicos galácticos.

De acuerdo con [14] los rayos cósmicos galácticos (GCR, por sus siglas en inglés) se constituyen principalmente de partículas cargadas. En la heliosfera, interactúan con el campo magnético solar. Las cantidades de GCR que alcanzan la atmosfera de la Tierra dependen de la fuerza del campo magnético solar y de la energía de las partículas del GCR. Una baja actividad solar implica menor blindaje magnético solar y por lo tanto más rayos cósmicos alcanzando la Tierra. Sin entrar en detalles técnicos, los GCR en la heliosfera también dependen de la polaridad del campo magnético solar.

Los rayos cósmicos galácticos provienen de los restos de supernovas, los cuales son explosiones masivas durante las etapas finales de estrellas masivas que se convierten en agujeros negros o son destruidas. La energía liberada en estas explosiones acelera partículas cargadas desde fuera del sistema solar, haciéndolas altamente penetrantes y difíciles de aislar. En esencia, las supernovas actúan como aceleradores de partículas naturales y gigantes. La tierra está continuamente expuesta a la radiación cósmica.

Los rayos cósmicos son un problema grave ya que pueden inducir fallas en dispositivos electrónicos convencionales, como las memorias RAM, microprocesadores y transistores de potencia.

3.2.3 Efectos de la radiación en la electrónica convencional.

El ambiente de radiación espacial tiene fuertes efectos en la electrónica convencional por lo que es necesario considerar esta variable en el diseño de aparatos espaciales. A continuación, se dará una pequeña introducción de los efectos de la radiación en la electrónica.

Como se explica en [15], los efectos de evento único (SEE, por sus siglas en inglés) son causados únicamente por una partícula energética que atraviesa una región sensible del dispositivo y los efectos pueden manifestarse de muchas formas. Los SEE pueden ser destructivos o no destructivos.

Los trastornos de un solo evento *(SEU,* por sus siglas en inglés*)* son errores no destructivos. Estos efectos ocurren principalmente en memorias y registros de lógica secuencial como lo pueden ser flip flops. Uno de los efectos más comunes debido a los SEU en los *bits flips* los cuales consisten en el cambio en el estado de un bit en memoria, el estado puede cambiar de un estado alto a bajo y viceversa. Estos efectos no generan un daño permanente y se pueden solucionar reescribiendo la memoria.

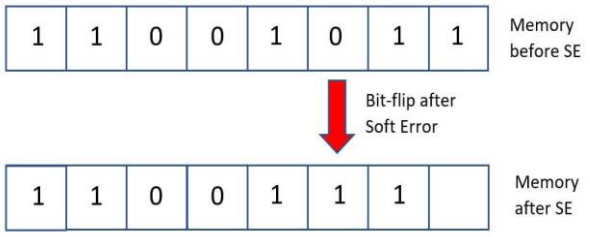

Figura 23. Bitflip Imagen obtenida de [15]

En la Figura 23 se muestra un bit flip, se puede observar que los datos en memoria después de un SEE se ven corrompidos. En este caso cambiando el estado de un bit de bajo a alto.

Muchos tipos de errores graves, y potencialmente destructivos pueden aparecer, uno de ellos son los Cortos Circuitos de Evento Único (SEL, por sus siglas en inglés) resultan en una corriente de operación muy elevada, por arriba de las especificaciones del dispositivo, y deben ser mitigadas por un reinicio de energía. Otros efectos graves incluyen sobrecalentamiento en los MOSFET de potencia, destrucción de la compuerta de los transistores, bits congelados y ruido en los circuitos integrados.

Una causa de los SEE son los rayos cósmicos (partículas solares y rayos cósmicos galácticos). Los rayos cósmicos afectan a la electrónica convencional induciendo iones, si un ion deposita la suficiente carga en una región sensible se pueden ocasionar eventos como *bit flips* o estados transitorios no deseados.

3.3 Parámetros importantes en el estudio de los efectos de la radiación en los dispositivos electrónicos.

3.3.1 Radiación.

De acuerdo con [16] la radiación es una forma de energía que se emite en forma de rayos, ondas electromagnéticas y partículas, en algunos casos la radiación es visible para el ser humano o puede ser sentida, la radiación también puede venir de distintas fuentes como lo son los rayos x y gamma.

Según su nivel de energía, la radiación se puede clasificar en dos tipos:

- Radiación no ionizante. Menos energética, por lo que no tiene la energía suficiente de remover electrones del material que cruza.
- Radiación Ionizante. Mas energética, por lo que tiene la energía suficiente para remover electrones del material que cruza.

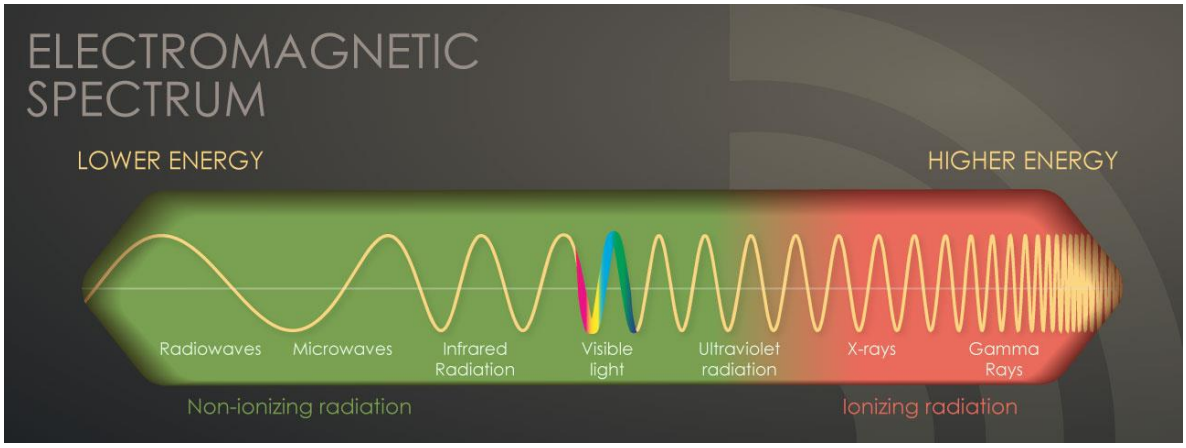

Figura 24.Radiación ionizante y radiación no ionizante. Imagen obtenida de [16]

En la Figura 24 se puede observar que la radiación no ionizante está comprendida por ondas de radio, microondas, radiación infrarroja, luz visible, y rayos ultravioletas, mientras que la radiación ionizante está compuesta por rayos X y gamma.

A diferencia de la radiación que se experimenta en la tierra, la radiación espacial es diferente ya que se compone de átomos que han perdido sus electrones debido a la gran aceleración que han sufrido en su viaje interestelar, alcanzando la velocidad de la luz.

La radiación espacial se puede dividir en tres tipos:

- Partículas atrapadas en el campo magnético de la tierra.
- Partículas disparadas durante las erupciones solares.
- Rayos cósmicos galácticos.

3.3.2 Dosis Total de Ionización

De acuerdo con [17], la dosis total de ionización (TID, por sus siglas en inglés) es una medida que permite saber la energía total absorbida por la materia, su unidad de media se presenta a continuación:

$$rad = 0.01 \frac{J}{Kg}$$

Donde

$rad \overset{\text{def}}{=} Dosis\ de\ radiación\ absorbida$

$J = Joules$

$Kg = Kilogramos$

3.3.3 Transferencia Lineal de Energía

La transferencia lineal de energía (LET, por sus siglas en inglés) se usa para medir el rastro de ionización descrito por el recorrido de un ion a través de un material. La LET depende del material y también de la energía y carga de la partícula energética. Para iones provenientes de la radiación espacial, la LET se incrementa para energías decrecientes [18]

La medida de la perdida de energía, se le conoce como poder de frenado. A menudo la LET se aproxima al poder de frenado.

En resumen, cuando un ion atraviesa un material, la LET mide la energía depositada en un material por unidad de longitud. Sus unidades son:

$$MeV\ \frac{cm^2}{mg}$$

Un parámetro importante en que considerar en los dispositivos para aplicaciones espaciales es el umbral de trasferencia de energía lineal.

$$LET_{TH}$$

Su definición es la LET mínima que ocasiona un SEE en los dispositivos electrónicos [19].

3.4 Técnicas de detección y corrección de fallas.

Debido a que el espacio exterior es un ambiente hostil para la electrónica, en los sistemas espaciales es necesario aplicar técnicas de corrección y detección de fallas que permitan diagnosticar y reparar el estado funcional de cada uno de los módulos de la OBC, ya que estos módulos pueden alterar su funcionamiento debido a la presencia de SEE

A continuación, se revisarán algunas de las técnicas de detección y corrección de fallas que podrán ser aplicadas a este proyecto.

3.4.1 Watchdog

Los sistemas embebidos necesitan ser autosuficientes, atender sus propias necesidades sin requerir un elemento externo o un operador. En algunos casos cuando se presentan fallas en el funcionamiento del sistema embebido no es posible esperar a que un elemento externo los reinicie, por ejemplo, un operador humano. Los humanos somos extremadamente lentos a comparación de los sistemas digitales, un segundo en un sistema digital es una eternidad y en algunas aplicaciones de los sistemas embebidos ni siquiera es posible que un ser humano interactúe de forma directa.

Un claro ejemplo de un sistema embebido que no tiene interacción con un ser humano es el que compete a esta tesis: la computadora de a bordo de un satélite.

Debido a las razones anteriores existen circuitos tolerantes a fallas conocidos como los perros guardianes (WD, por sus siglas en inglés), los cuales son módulos de hardware que pueden ser usados para detectar automáticamente anomalías de software y reiniciar al CPU. De manera general un WD es un contador que cuenta desde un valor inicial hasta cero. El software del sistema embebido selecciona el valor inicial del contador y periódicamente lo reinicia por medio de una señal de alivio. Si el contador alcanza el valor final de 0 quiere decir que el sistema embebido no ha generado la señal de alivio y por lo tanto se intuye que se ha presentado un

fallo por lo que el contador generará una señal que reiniciará al sistema embebido para así poder recuperar su funcionamiento habitual.

En la Figura 25 se puede observar el diagrama funcional de un watchdog, es de particular interés observar las señales entre el CPU y el WD. En un correcto funcionamiento el CPU reiniciará la cuenta del WD por medio de la señal *Reset CPU* de manera periódica evitando que este llegue a su valor final (cero). Si se presenta una falla en el CPU este dejará de emitir la señal de alivio, por lo tanto, el *watchdog* llegará a su valor final y activará la señal *Reset CPU* la cual reiniciará al CPU y por lo tanto recuperará el buen funcionamiento. [19]

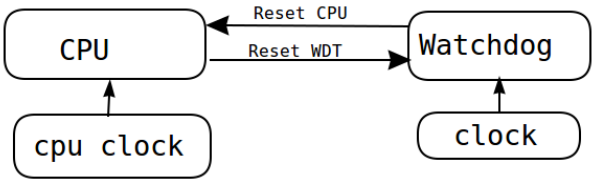

Figura 25. Diagrama funcional de un watchdog timer. Imagen obtenida de [19]

3.4.2 Redundancia de Hardware

De manera general la redundancia es una estrategia que permite que una función siga ejecutándose a pesar de que se presenten fallas en uno de sus elementos por lo que permite mejorar la confiabilidad de los sistemas espaciales. La implementación tradicional consiste en tener dos módulos funcionales que realizan la misma función.

Para determinar si un sistema embebido requiere redundancia es importante considerar si la función que se quiere hacer redundante resulta critica para la misión, ya que implementar redundancia de hardware implica aumentar el peso, costo y consumo energético, variables muy importantes en el sector aeroespacial.

Anteriormente la redundancia solía aplicarse en componentes físicos diferentes, sin embargo, con el avance de la tecnología ahora se puede implementar en el mismo componente físico.

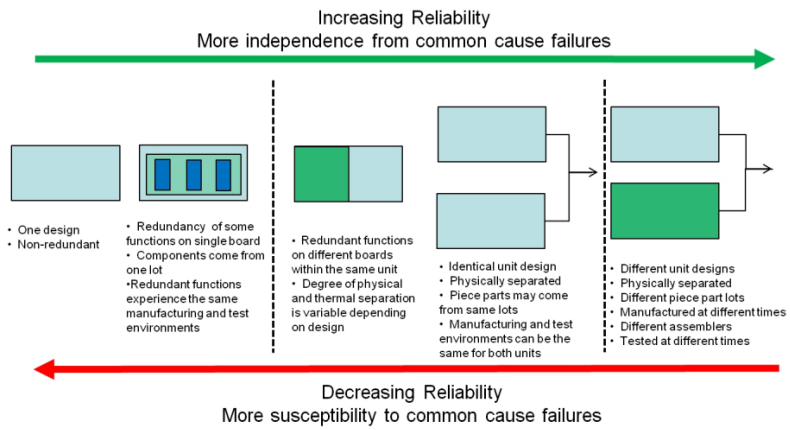

Figura 26. Variedad de esquemas para implementar redundancia en sistemas electrónicos. Imagen obtenida de [40]

De acuerdo con [20] se mostrarán algunos ejemplos de la redundancia física en sistemas electrónicos.

En la Figura 26 se pueden observar algunos esquemas de redundancia aplicados a sistemas electrónicos. De manera general los esquemas de redundancia de la derecha son más fiables sin embargo tienen ciertas desventajas por lo tanto es necesario escoger el esquema redundante que mejor cumpla la función.

En la Figura 26 se ilustran los siguientes esquemas redundantes:

- Un solo diseño sin redundancia. Un sistema electrónico puede no tener redundancia, una función de la misión es atendida por solo un módulo.
- Redundancia con módulos de hardware distintos, pero dentro de una misma placa. Un ejemplo de esto es implementar dos módulos UART desde un mismo microcontrolador, y cada uno de estos módulos está dedicado a atender una función común.
- Funciones redundantes en placas distintas dentro de la misma unidad. Un ejemplo de esto es implementar comunicación UART en dos microcontroladores idénticos, pero en diferentes placas con el fin de atender la misma función.
- Diseño de unidades idénticas y físicamente separadas. Las unidades se fabrican bajo la misma tecnología, sin embargo, aumenta la confiabilidad al tener aislamiento físico.
- Diseño de unidades diferentes y físicamente separadas. Cada una de las unidades se fabrica con tecnología diferente, el esquema de validación es

diferente, por lo tanto, es muy poco probable que se presente el mismo tipo de falla en ambas unidades.

3.5 Códigos de detección de errores.

3.5.1 CRC

El CRC es un código usado para detectar inconsistencias en grupos de datos, y particularmente se puede usar para detectar errores durante la transmisión mensaje, su principio de funcionamiento se basa en la división polinómica, el dato es tratado como un polinomio que es dividido por el polinomio CRC (el cual tiene múltiples definiciones como lo puede ser el CRC-16 utilizado en el protocolo USB o CRC-15 utilizado en el protocolo CAN, entre otros), al residuo de esta división se le conoce como suma de verificación CRC el cual se adjunta al mensaje original.

El receptor del mensaje se encarga de hacer otra división polinómica, donde el dividendo es el mensaje junto con la suma de verificación CRC recibida y el divisor es el polinomio CRC. Si después de efectuar la división el residuo es igual a 0, se asume que la transmisión de datos ha sido exitosa.

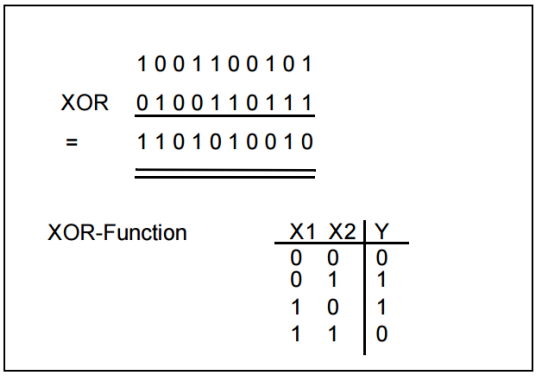

Figura 27. Aritmética Modulo-2. Imagen obtenida de [21]

De acuerdo con [21], la división utiliza aritmética de Modulo-2. La aritmética de Modulo-2 se realiza simplemente haciendo una función XOR entre dos números binarios como se puede apreciar en la Figura 27.

A continuación, se presentará un ejemplo para ilustrar los conceptos antes mencionados.

Suponga que se quiere transmitir el siguiente mensaje: 0x35, utilizando un polinomio de grado 2 como divisor ($P_1 = x^2 + 1$).

Primero se tiene que obtener el número en binario correspondiente al mensaje, agregando dos bits a la derecha ya que se utilizará un polinomio de grado dos como divisor

$$mensaje = 0x35 = 11010100b$$

después se obtiene el polinomio correspondiente

$$(1)x^7 + (1)x^6 + (0)x^5 + (1)x^4 + (0)x^3 + (0)x^2 + (0)x^1 + 1$$

$$P_2 = x^7 + x^6 + x^4 + x^2$$

Finalmente, la suma de verificación CRC será el resto de la siguiente división polinómica:

$$suma\ de\ verificación = P_2/P_1$$

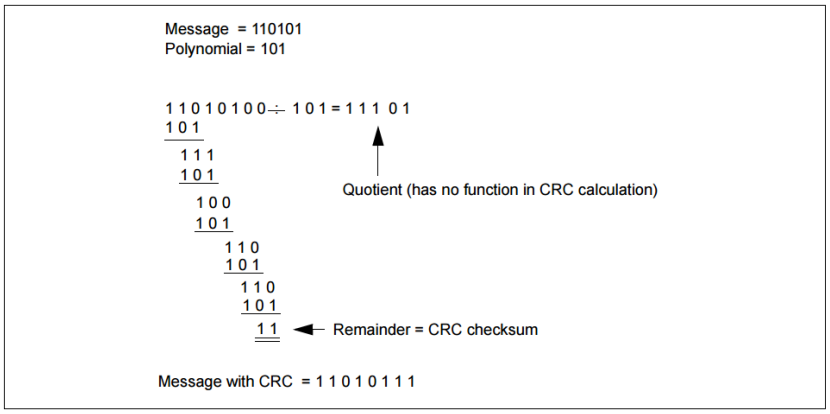

Figura 28. Obtención del checksum CRC. Imagen obtenida de [21]

En la Figura 28 se observa el proceso efectuado para obtener la suma de verificación CRC, es importante hacer notar que antes de efectuar la división polinómica se han agregado 2 bits a la derecha del mensaje, esto es debido a que el polinomio CRC es de grado 2, por lo que para un polinomio CRC de grado n se deberán agregar n bits al mensaje original. La suma de verificación obtenida con los datos del ejemplo es:

$$suma\ de\ verificación = x + 1 = 011b$$

En el receptor recibe el siguiente dato:

$$11010111b$$

El polinomio correspondiente es:

$$P_3 = x^7 + x^6 + x^4 + x^2 + x + 1$$

El receptor realiza la división polinómica:

$$P_3/P_1$$

Del lado del receptor se realiza la verificación como se muestra en la Figura 29, se puede asumir que el mensaje se ha transmitido correctamente si el residuo de la división polinomial es 0.

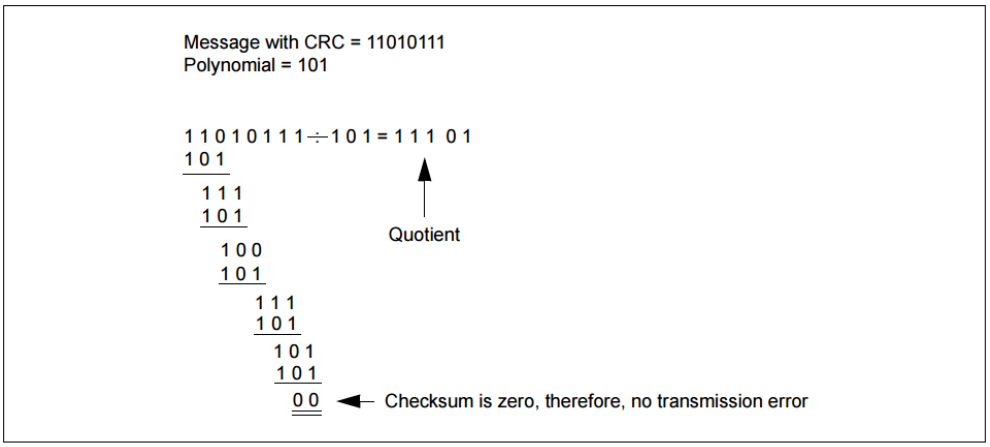

Figura 29. Verificación de los datos recibidos por medio de la suma de verificación CRC. Imagen obtenida de [21]

3.6 Unidades de almacenamiento en sistemas embebidos.

Los microcontroladores requieren de unidades de almacenamiento para su funcionamiento, desde almacenar datos permanentes hasta datos dinámicos que se escriben, borran y cambian únicamente durante la ejecución del programa.

Las unidades de almacenamiento se pueden clasificar según su uso en el sistema embebido en cuestión: almacenamiento primario y almacenamiento secundario. A

continuación, se explicará que función desempeñan los dispositivos de almacenamiento según su clasificación.

Almacenamiento primario: Consiste en dispositivos de almacenamiento en los que el CPU guarda datos temporales durante la ejecución del programa, se puede decir que la ejecución del programa se realiza totalmente sobre este tipo de memorias. Generalmente este tipo de memorias son volátiles, lo que significa que se pierden los datos almacenados al perder la alimentación eléctrica.

Almacenamiento secundario: Consiste en dispositivos de almacenamiento que no operan directamente con el CPU, sino que son usados para el almacenamiento masivo de datos, por ejemplo, imágenes. Generalmente este tipo de memorias son no volátiles lo que significa que los datos almacenados en la memoria permanecen aún sin energización.

Las memorias en los sistemas embebidos caen principalmente en dos categorías: volátiles y no volátiles, usualmente denominadas RAM y ROM respectivamente.

3.6.1 Memorias RAM.

De acuerdo con [22], las memorias de acceso aleatorio (RAM, por sus siglas en inglés) son tipo de memoria que de acuerdo con su arquitectura de hardware es posible acceder a cualquier localidad de memoria, en cualquier orden, para realizar operaciones de escritura y lectura. Todas las RAM permiten la capacidad de lectura y escritura. Debido a que las RAM pierden los datos almacenados cuando no están energizadas, se les conoce como memorias volátiles.

A continuación, se presentan los tipos de memorias RAM que un sistema embebido puede usar:

- SRAM: la memoria estática de acceso aleatorio (SRAM, por sus siglas en inglés) tiene la propiedad de mantener su contenido mientras esta se encuentre energizada, si se interrumpe la alimentación ya sea temporal o permanentemente, los datos se perderán para siempre. La velocidad de la SRAM es aproximadamente cuatro veces mayor que la DRAM sin embargo

es mucho más costosa. Usualmente se implementa una SRAM cuando la velocidad de acceso es crucial.

- DRAM: la memoria dinámica de acceso aleatorio (DRAM, por sus siglas en inglés) tiene la particularidad de que los datos tienen un tiempo de vida extremadamente bajo, por lo que los datos tienen que ser actualizados constantemente por medio de un controlador. Tienen un costo por byte menor que una SRAM, por lo que se vuelven atractivas cuando se requieren grandes cantidades de memoria, pero demandan mayor energía para su operación.

Algunos sistemas embebidos suelen incluir ambos tipos de memorias: un bloque pequeño de SRAM para datos críticos o que requieren una alta velocidad, y un bloque mucho más grande de DRAM para todo lo demás.

3.6.2 Memorias ROM.

De acuerdo con [22], las memorias ROM son un tipo de memorias que debido a su arquitectura de hardware permiten almacenar los datos de forma permanente a pesar de que no estén energizadas. Por lo general, las ROM no permiten operaciones de escritura, solamente de lectura. Es importante que aclarar que al igual que las RAM, las memorias ROM también son de acceso aleatorio.

A continuación, se presentan los tipos de memorias ROM que un sistema embebido puede usar:

- ROM enmascarable: Es un tipo de ROM que solo puede ser programada durante su fabricación, después los datos en memoria no pueden cambiar. La ROM enmascarable es extremadamente barata sin embargo tiene la desventaja de que los datos ya grabados en la memoria no pueden cambiar jamás.
- PROM: las memorias programables de solo lectura (PROM, por sus siglas en ingles), se pueden comprar sin programar sin embargo una vez que se programan, los datos almacenados no pueden ser cambiados. Los

programadores las utilizan para cargar firmware de un sistema embebido que jamás cambiará.

3.6.3 Memorias hibridas.

Algunas memorias no volátiles tienen características clave de las memorias volátiles. Se pueden programar y reprogramar, pero también guardan su contenido a pesar de la ausencia de energización.

- EEPROM: Las memorias programables de solo lectura borrables eléctricamente (EEPROM, por sus siglas en inglés) pueden ser reprogramadas a través de un proceso eléctrico el cual se realiza byte a byte. Las EEPROM se suelen utilizar para almacenar firmware que posteriormente será actualizado.
- FLASH: Son las más comunes en los sistemas embebidos ya que las operaciones de lectura y escritura se pueden ejecutar rápidamente, además son de bajo costo. Al igual que las EEPROM, son eléctricamente reprogramables, sin embargo, solo se puede reprogramar un sector a la vez.
- FRAM: La memoria de acceso aleatorio ferroeléctrica (FRAM, por sus siglas en inglés) basa su principio de funcionamiento en el efecto ferroeléctrico, el cual permite que los datos no se pierdan después de perder la alimentación. Es más rápida, además ofrece más ciclos de lectura/escritura y su consumo energético es menor.

3.6.4 Ventajas de la FRAM frente a otras tecnologías.

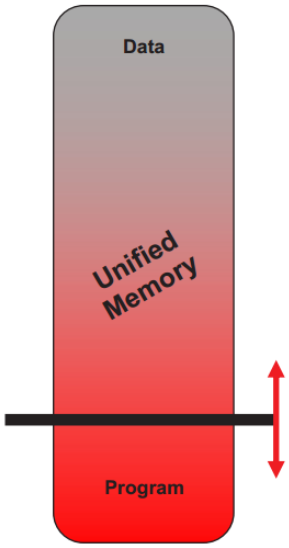

Figura 30. FRAM como memoria unificada. Imagen obtenida de [24]

De acuerdo con [23] las memorias FRAM se comportan similar a una DRAM. Permiten el realizar operaciones de escritura y lectura en cada byte. A diferencia de las EEPROM o las FLASH, las FRAM no requieren una secuencia especial para escribir datos. La FRAM es una memoria no volátil debido a que utiliza un material dieléctrico especial en el capacitor de almacenamiento, este material permite hacer uso del efecto ferroeléctrico.

La FRAM permite que surja el concepto de memoria unificada ya que permite tener particiones para el código y los datos como se puede ver en la Figura 30, esto permite usar la misma memoria tanto como para el almacenamiento del código y como par el de los datos, que tradicionalmente se guardan en memorias diferentes (RAM y ROM).

Por otra parte, las memorias FRAM tienen una gran durabilidad, lo que significa que soportan un número muy grande de operaciones de escritura en comparación con las memorias FLASH, estimado en más de 10^{14} ciclos de escritura.

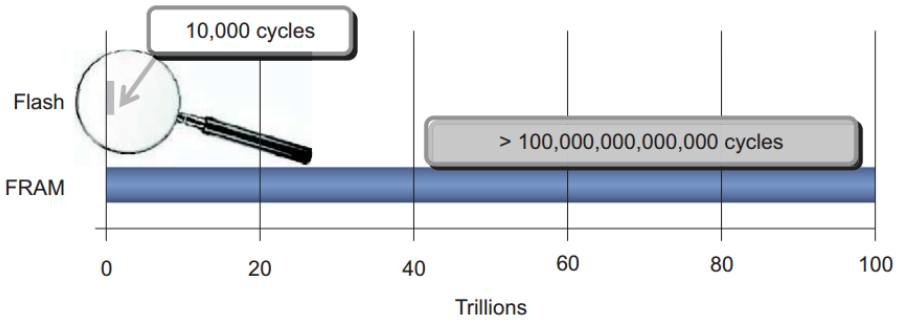

Figura 31. FRAM – Durabilidad. Imagen obtenida de [24]

En la Figura 31 se puede observar una comparación entre la durabilidad de una memoria FLASH y una FRAM, se puede apreciar que la durabilidad de la FRAM es mucho más grande que la de la FLASH.

Otra gran ventaja de la FRAM frente a otras memorias es la velocidad de escritura, por ejemplo, el proceso para escribir una palabra (ocho bits) en una memoria FLASH puede ir desde 37 µs hasta 85 µs sin considerar la operación de borrado en caso de requerir una, por otra parte, una FRAM solo requiere de aproximadamente 100 ns para escribir una palabra (ocho bits) y además no requiere operación de borrado ya que se puede sobrescribir.

3.6.5 Desempeño de las FRAM en ambientes radioactivos.

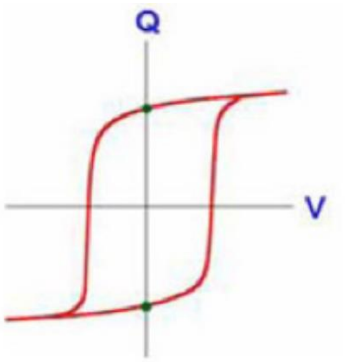

Figura 32. Curva Q(V) de los materiales ferroeléctricos. Imagen obtenida de [25]

Comúnmente se entiende mal el termino ferroeléctrico, ya que los cristales ferroeléctricos tienen acero o son ferromagnéticos o tienen propiedades similares. Sin embargo, el termino ferroeléctrico se refiere a la similitud de la gráfica de la carga en función del voltaje (vea Figura 32) en comparación con la curva de histéresis de los materiales ferroeléctricos. Los materiales ferroeléctricos no son afectados por los campos magnéticos.

De acuerdo con [24], las memorias volátiles SRAM y DRAM usan un capacitor o un candado digital (digital latch) para almacenar el estado. Estas celdas de memoria pueden ser alteradas fácilmente debido a partículas Alpha, rayos cósmicos, iones pesados, rayos X y gamma. Debido a que las celdas de memoria de la FRAM almacenan el estado por medio de una polarización PZT, es muy poco probable que el impacto de una partícula Alpha cambie el estado de una celda de memoria. La resistencia a la radiación de las memorias FRAM las hace altamente atractivas para aplicaciones espaciales.

3.7 Tecnologías resistentes a la radiación.

3.7.1 El microcontrolador MSP430FR5969.

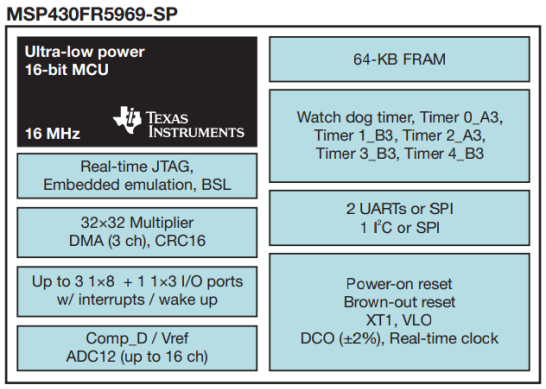

Figura 33. Módulos funcionales del microcontrolador MSP430FR5969-SP. Imagen obtenida de [26]

De acuerdo con [25] el microcontrolador MSP430FR5969-sp es ampliamente utilizado en el sector espacial ya está construido con tecnología resistente a la radiación y tiene herencia de vuelo. Por otra parte, es importante mencionar que este microcontrolador tiene una versión COTS, lo cual resulta interesante para la etapa de desarrollo del proyecto.

La arquitectura del microcontrolador de grado espacial se puede observar en la Figura 33. Algunas características claves son las siguientes:

- Consumo energético extremadamente bajo
- 64 KB de memoria FRAM no volátil.
- Periféricos integrados para mantenimiento y telemetría.
- Soporte para osciladores de 32 kHz o fuentes de reloj internas.
- Tamaño y peso reducidos.
- Resistente a la radiación
- TID = 75 krad (Si), 50 krad (Si) RHA
- SEL immune to LET = 72 MeV-cm^2/mg

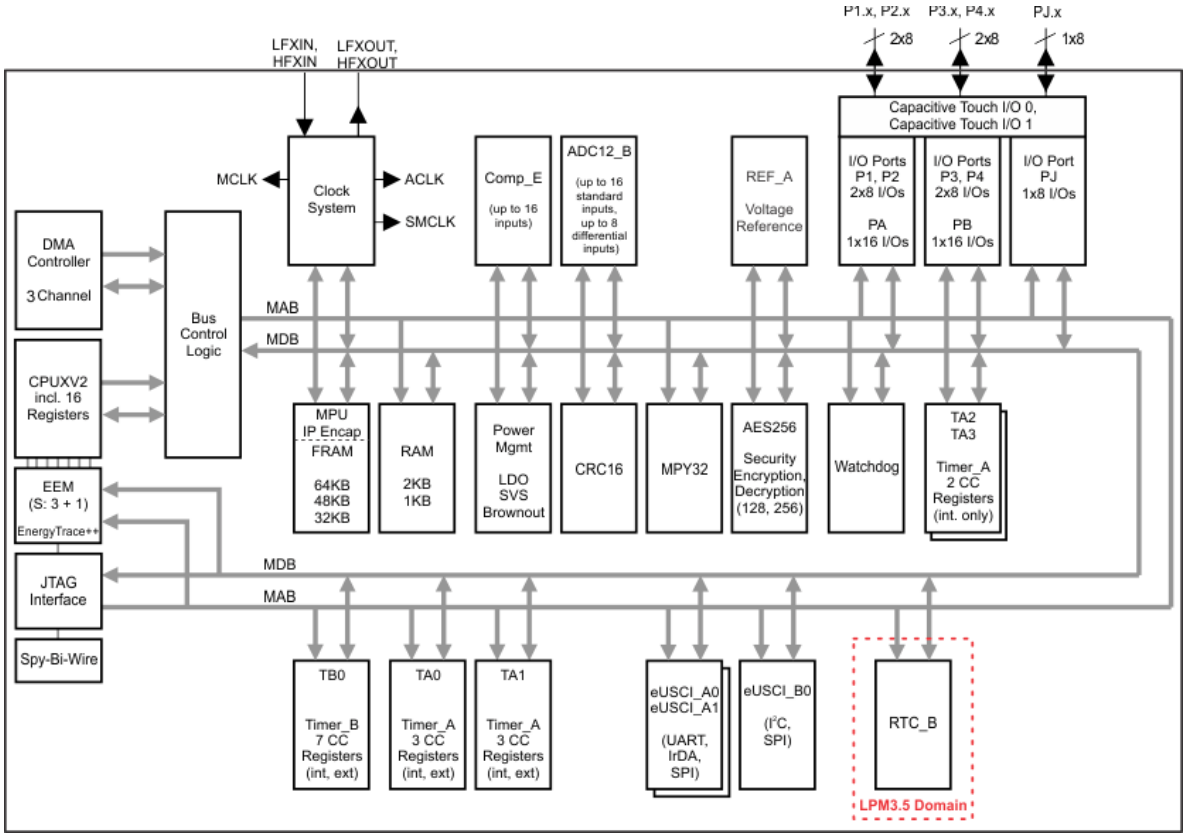

Figura 34. Arquitectura del microcontrolador MSP430FR5969

En la Figura 34 se puede observar el diagrama funcional del microcontrolador MSP430FR5969. A continuación, se describirán los aspectos más importantes de la arquitectura teniendo en cuenta el objetivo de este trabajo.

Se puede observar que los bloques funcionales se encuentran conectados por dos buses: el bus de datos en memoria (MDB, por sus siglas en ingles), y el bus de direcciones de memoria (MAB, por sus siglas en ingles).

3.7.1.1 CPU

El CPU implementado en esta serie de microcontroladores es de 16 bits de arquitectura RISC, que soporta velocidades de hasta 16 MHz.

3.7.1.2 Sistema de reloj

Los relojes de los microcontroladores solían ser simples (Un simple oscilador con una frecuencia fija) sin embargo las necesidades de la industria demandan un alto desempeño y al mismo tiempo un bajo consumo energético, por lo que los

microcontroladores modernos tienen sistemas de reloj mucho más complicados, a menudo con dos o más fuentes de reloj, un reloj rápido para alto desempeño y uno

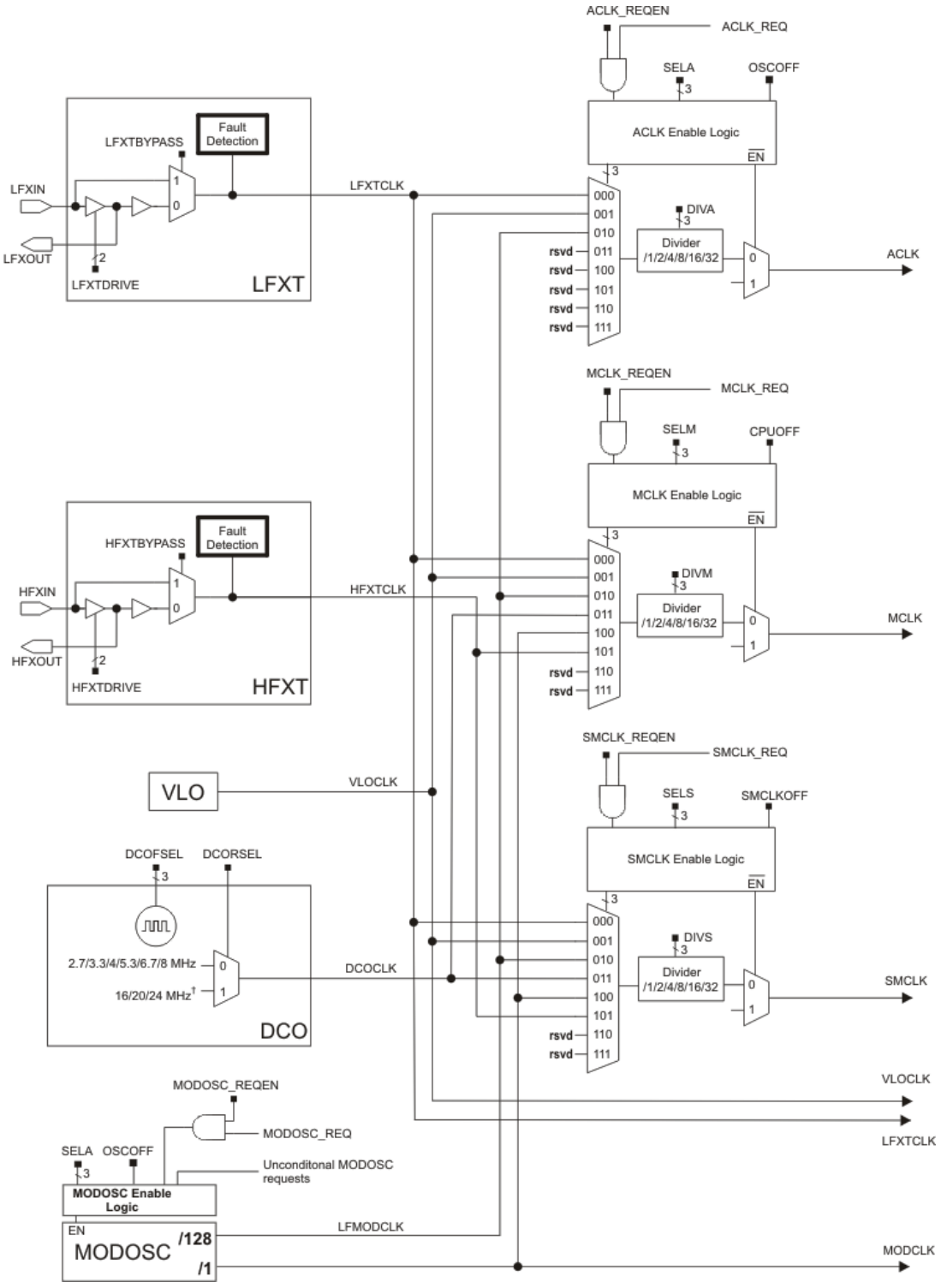

Figura 35. Diagrama a bloques del sistema de reloj del MSP430FR5969

lento para bajo desempeño, pero con gran eficiencia energética.

En la Figura 35 se puede observar el diagrama a bloques del sistema de reloj del MSP430FR5969. De manera general la arquitectura está compuesta de la siguiente manera:

- 4 posibles fuentes de reloj: LFXTCLK, HFXTCLK, DCO, LFMODCLK.
    - En el caso de LFXTCLK y HFCTCLK cada uno tiene su propio oscilador, el primero de ellos es de baja frecuencia y el segundo es de alta frecuencia, también se presenta un módulo de hardware que permite detectar errores en la señal oscilante.
    - El DCO es un oscilador digitalmente controlado, el cual puede ser configurado por software y generar distintas frecuencias de señal de reloj (desde 2.7 hasta 24 MHz).
    - El MODOSC es un oscilador interno que puede ser usado por todas las señales de reloj. También se usa como un reloj libre de fallas. Es el más seguro de todas las fuentes de reloj.
- 3 principales fuentes de reloj seleccionables por los periféricos: ACLK, MCLK y SMCLK. Cada señal de reloj puede tomar su fuente de reloj de todas las posibles, adicionalmente a esto se tiene un circuito lógico que puede dividir la frecuencia de la señal de reloj (2,4,8,16 o 32 veces). A diferencia de otros microcontroladores, los periféricos no tienen módulos dedicados a modificar la frecuencia de la señal de reloj que reciben.

3.7.1.3 FRAM

LA FRAM puede ser programada a través del puerto JTAG, Spy-Bi-Wire (SBW) y el BSL. Las características de la FRAM son las siguientes:

- Consumo energético extremadamente bajo. Operación de escritura en la memoria no volátil extremadamente rápida.
- Código de detección de errores. (ECC, por sus siglas en ingles)

3.7.1.4 Módulos eUSCI_A0 y eUSCI_A1

El módulo eUSCI_Ax soporta los siguientes protocolos de comunicación serial:

- UART, las características más importantes son:
  - Longitud de datos configurable: 7 o 8 bits. Bit de paridad par, o impar o sin bit de paridad.
  - Registros de corrimiento independientes para la transmisión y recepción.
  - Baudrate programable.
  - MSB primero o LSB primero.
  - Banderas de para detección y corrección de errores.
  - Banderas de interrupción para recepción, transmisión, comienzo de transmisión y transmisión de datos completa.
- SPI, las características más importantes son:
  - Longitud de datos configurable: 7 o 8 bits.
  - MSB primero o LSB primero.
  - Operación de 4 o 3 líneas (Con o sin CS).
  - Modo esclavo o maestro.
  - Registros de corrimiento independientes para la transmisión y recepción.
  - Registros de buffer independientes para transmisión y recepción.
  - Selección de la polaridad del reloj y el control de fase.
  - Frecuencia de reloj programable en modo maestro.

3.7.1.5 RTC

El módulo RTC provee modo calendario, programación de alarmas y calibración. El RTC también soporta operación en LPM3.5. A continuación se muestran las características más importantes:

- Reloj en tiempo real y modo calendario, provee: segundos, minutos, horas, día de la semana, día del mes, mes y año.
- Capacidad para interrupciones.

- Selección de formato BCD o binario.
- Alarmas programables.
- Lógica de calibración para desfasamientos de tiempo.
- Operación en LPM3.5.

3.7.1.6 Timer

El módulo temporizador es de 16 bits con varios registros de captura/comparación. A continuación, se presentan brevemente las características.

- Contador de 16 bits asíncrono con cuatro modos de operación.
- Fuente de reloj seleccionable y configurable.
- Hasta siete registros de captura/comparación.
- Interrupciones programables.

3.7.1.7 Modos de bajo consumo.

El microcontrolador MSP430FR5969 puede admitir modos de bajo consumo (LPM, por sus siglas en inglés) los cuales son modos de hibernación que reducen drásticamente el consumo energético y pueden volver al estado de operación activo o normal después de una interrupción causada por algún periférico. A continuación, se revisará más a detalle los modos de bajo consumo de la serie de microcontroladores MSP430.

Hay cinco LPM, pero dos de ellos se emplean raramente, por lo que se revisarán los más comunes:

- Modo activo: el CPU, todos los relojes, y los módulos inicializados están activos (El consumo de corriente es aproximadamente I = 300 µA). El MSP430 inicia por defecto en este modo, el cual debe ser implementado cuando se requiere el CPU. Una interrupción cambia automáticamente el dispositivo al estado activo.
- LPM0: el CPU, y el MCLK se deshabilitan, SMCLK y ACLK se mantienen activos, (el consumo de corriente es aproximadamente I = 85 µA). Este modo

se utiliza cuando no se requiere el CPU, pero algunos módulos requieren fuentes de reloj de SMCLK y ACLK.

- LMP3: el CPU, el MCLK, el SMCLK y el DCO se deshabilitan, únicamente el ACLK permanece activo (El consumo de corriente aproximado es de I = 1 µA). Este modo se utiliza cuando el microcontrolador necesita despertarse a sí mismo cada cierto tiempo debido a una interrupción causada por un evento programado en el RTC interno.
- LMP4: el CPU y todas las señales de reloj se encuentran deshabilitadas (El consumo aproximado de corriente es de I = 0.1 µA). El dispositivo puede ser despertando únicamente por una señal externa.

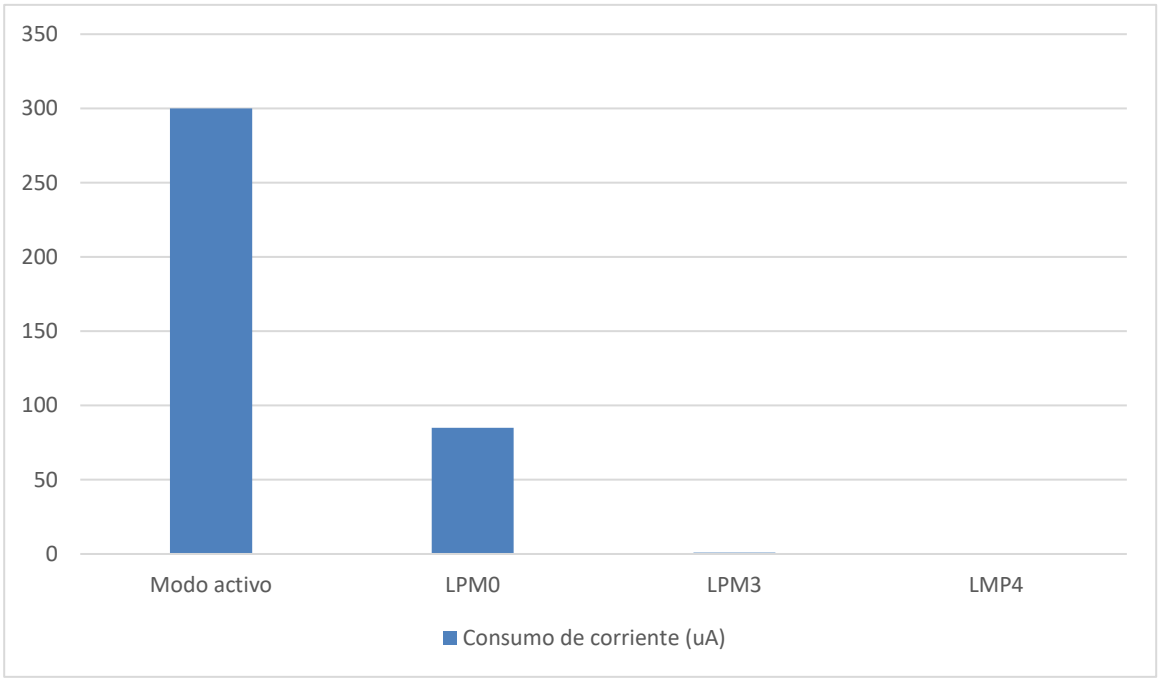

Figura 36. Consumo del MSP430 en función del modo de operación.

En la Figura 36 se puede observar el consumo de corriente del MSP430 en función de su modo de operación (modo activo, LMP0, LPM3 y LPM4). Se puede observar que en el LPM0 el consumo de corriente es menos de la tercera parte del consumo del modo activo, para los LPM3 y LPM4 el consumo de corriente es tan bajo en comparación con el del modo activo que prácticamente no tienen una representación visual debido a la escala de la gráfica.

3.7.2 FRAM Cypress CY15B10Q

De acuerdo con [26] la memoria CY15B104Q es una memoria no volátil que implementa un avanzado proceso ferroeléctrico. Provee almacenamiento hasta por 151 años mientras que elimina los problemas de complejidad y confiabilidad causadas por el flash seriales, EEPROM, y otras memorias no volátiles.

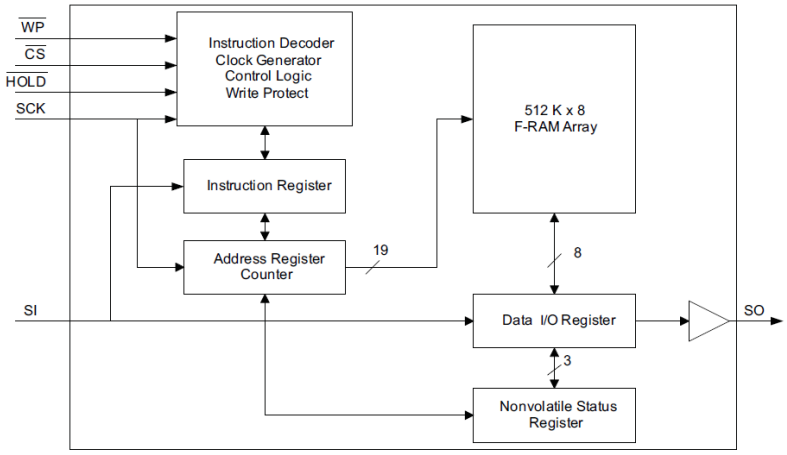

Figura 37.Arquitectura de la FRAM CY15B104Q. Imagen obtenida de [27]

De manera general las características de la memoria son las siguientes:

- 4 Mbit (512 K x 8)
- Gran durabilidad. Hasta 100 billones de operaciones de escritura/lectura.
- Interfaz SPI muy rápida. (Hasta 40 MHz)
- Protección de escritura sofisticada.
- Bajo consumo energético.
    - 300 µA en estado activo a 1 MHz
    - 100 µA en modo de espera.
    - 3 µA en modo de hibernación.

En la Figura 37 se puede observar la arquitectura de la FRAM CY15B10Q. La memoria cuanta con 512 K localidades de 8 bits cada una. Se puede acceder a las localidades de memoria usando el bus SPI. Para realizar una operación se requiere un código de operación y tres bytes de dirección, sin embargo, el direccionamiento a memoria se realiza únicamente por 19 bits, por lo que los primeros 5 bits más significativos del byte más significativo son ignorados.

3.7.2.1 Configuración de SPI.

La memoria puede ser controlada por un microcontrolador con la siguiente configuración de SPI.

- SPI Mode 0 (CPOL = 0, CPHA = 0)
- SPI Mode 1 (CPOL = 1, CPHA = 1)

En ambos modos los datos transmitidos son capturados en el flanco positivo de la línea SCK, empezando desde el primer pulso después de que la línea CS transita a estado bajo.

3.7.2.2 Estructura de comandos.

En la Tabla 2 se pueden observar cada uno de los OPCODES asignados a cada uno de los posibles comandos que se pueden ejecutar en la FRAM. Más adelante se explicarán como deben ser ejecutados los comandos de escritura y lectura los cuales son esenciales para el objetivo de este trabajo.

Tabla 2. OPCODES de la CY15B104Q. Obtenida de [27]

Name	Description	Opcode
WREN	Set write enable latch	0000 0110b
WRDI	Reset write enable latch	0000 0100b
RDSR	Read Status Register	0000 0101b
WRSR	Write Status Register	0000 0001b
READ	Read memory data	0000 0011b
FSTRD	Fast read memory data	0000 1011b
WRITE	Write memory data	0000 0010b
SLEEP	Enter sleep mode	1011 1001b
RDID	Read device ID	1001 1111b
Reserved	Reserved	1100 0011b
		1100 0010b
		0101 1010b
		0101 1011b

3.7.2.3 Ejecución del comando de escritura en la FRAM.

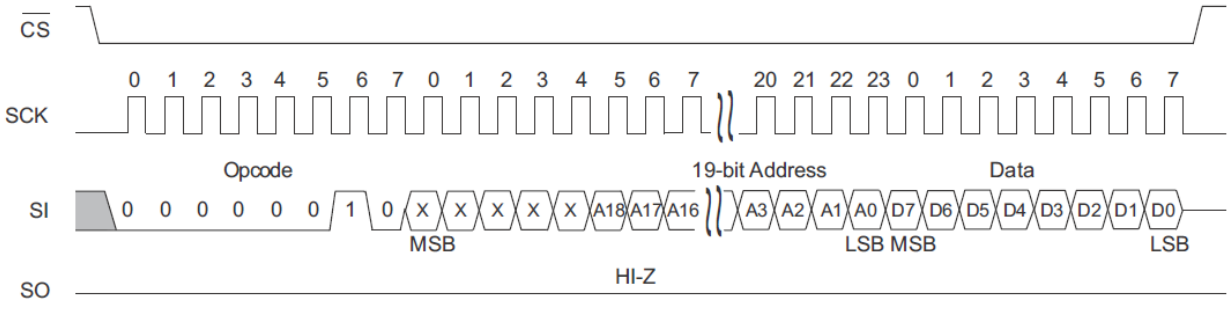

Figura 38. Operación de escritura en memoria (no se muestra la ejecución del comando WREN). Imagen obtenida de [27]

En la Figura 38 se muestra la ejecución de la operación de escritura en la FRAM, es importante mencionar que en esta figura la operación WREN no es mostrada sin embargo es necesaria siempre que se ejecuta una operación de escritura, de manera general la secuencia de pasos a seguir por el maestro del bus es la siguiente.

- Ejecución del comando WREN desde el maestro del bus SPI.
  - Cambiar el estado de la línea CS de alto a bajo.
  - Enviar el código referente a WREN (0110b)
  - Cambiar el estado de la línea CS de bajo a alto.
- Ejecución de comando WRITE desde el maestro de bus SPI.
  - Cambiar el estado de la línea CS de alto a bajo.
  - Enviar el código referente a WRITE (0010b)
  - Enviar la dirección de inicio por medio de 3 bytes empezando por el byte más significativo, los 5 bits más significativos del byte más significativo serán ignorados por la memoria FRAM (Ya que el direccionamiento únicamente requiere de 19 bits)
  - Enviar los bytes a escribir en memoria. (Mientras haya pulsos de CLK), la operación de escritura continuará. Si se quieren escribir 3 bytes el maestro del bus deberá proveer 24 ciclos de reloj.
  - Cambiar el estado de la línea CS de bajo a alto.

3.7.2.4 Ejecución del comando de lectura en la FRAM.

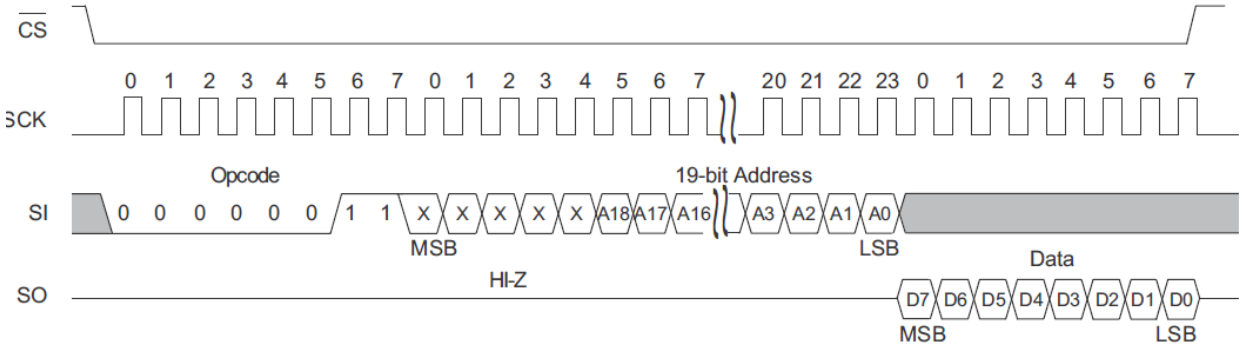

Figura 39. Operación de lectura en la FRAM. Imagen obtenida de [27]

La operación de lectura se puede observar en la Figura 39, a continuación, se explicarán la serie de pasos a seguir desde el maestro del bus:

- Cambiar el estado de la línea CS de alto a bajo.
- Enviar el OPCODE referente al comando de lectura (0011b).
- Enviar la dirección de inicio por medio de 3 bytes empezando por el byte más significativo, los 5 bits más significativos del byte más significativo serán ignorados por la memoria FRAM (Ya que el direccionamiento únicamente requiere de 19 bits).
- Mientras el maestro del bus provea pulsos de reloj, la FRAM enviará bytes. Si se quieren leer 3 bytes el maestro del bus deberá proveer de 24 ciclos de reloj.
- Cambiar el estado de las líneas CS de bajo a alto.

3.8 *Bootloader* del microcontrolador STM32F407.

El *bootloader* es un programa que está almacenado en la memoria ROM de arranque de la serie de microcontroladores STM32, y es cargado durante la producción de ST. Tiene como función principal cargar el código de aplicación a la memoria FLASH interna a través de uno de los periféricos seriales (UART, CAN,

USB, I2C y SPI). Un protocolo de comunicación se define para cada interfaz serial, con un conjunto de comandos y secuencias compatibles

3.8.1 Acceso al bootloader.

En el microcontrolador STM32, de acuerdo con [27] el *bootloader* puede ser activado después de un reset aplicando el siguiente patrón en los pines BOOT1 y BOOT0:

- BOOT0 = 1
- BOOT1 = 0

De manera general el funcionamiento es el siguiente:

- Se ocasiona un reset por medio de una transición de alto a bajo en el pin NRST
- Se configura el patrón necesario en los pines BOOT0 y BOOT1.
- Se vuelve a la operación normal del microcontrolador por medio de una transición de bajo alto del pin NRST

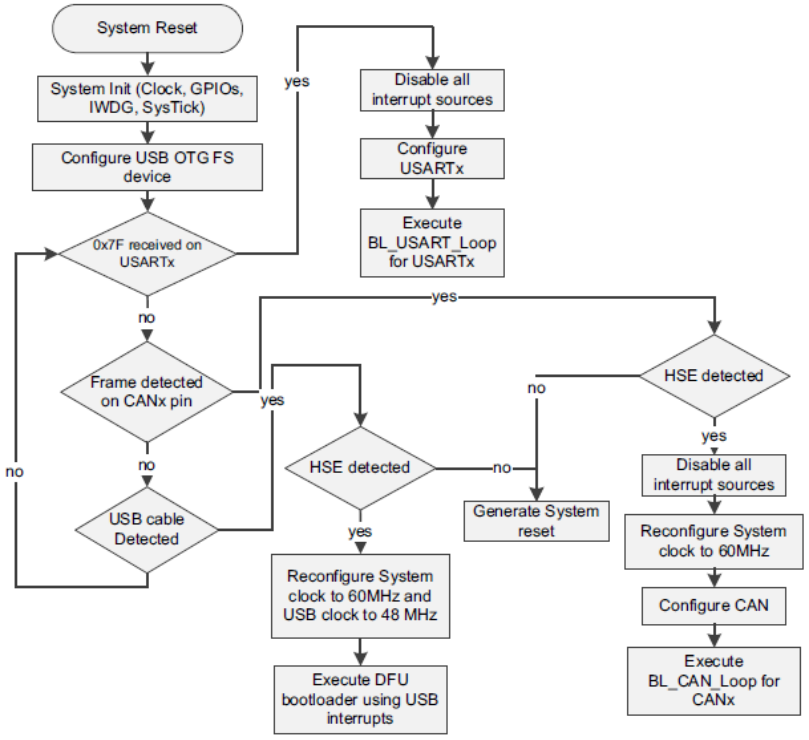

Figura 40. Proceso ejecutado por el uC STM32F407 después del acceso a su bootloader. Imagen obtenida de [28]

Si la operación se ha realizado correctamente se accederá al *bootloader*, esperando a recibir comandos provenientes de un protocolo de comunicación permitido.

En la Figura 40 se puede observarse el proceso ejecutado por el microcontrolador

STM32F43207 después de que se ha accedido a su *bootloader*, de manera muy general el proceso es el siguiente:

- Inicializa los elementos de hardware necesarios (Reloj, GPIOs, IWG, SysTick) para la operación del *bootloader*
- Configura el dispositivo USB OTG FS.
- De ahora en adelante el *bootloader* ejecutará la siguiente rutina de verificaciones hasta que una sea exitosa:
    - ¿Se ha recibido un 0x7F en el módulo USART?
    - ¿Se ha detectado una trama en el pin CANx?
    - ¿Se ha detectado la presencia de un cable USB?
- Cuando una de las condiciones se cumple se ejecuta un proceso correspondiente, sin embargo, solo es de interés de este trabajo la ejecución de comandos por medio de USART, por lo que a continuación se revisará el proceso correspondiente a la primera opción
- Después de que se ha recibido 0x7F en el módulo USART, se termina de configurar el módulo, y se ejecuta la rutina USARTx.
- El *bootloader* está listo para recibir comandos por medio de USART o UART.

Todas las instrucciones que se revisarán a continuación son ejecutadas por medio del protocolo UART.

3.8.2 Restricciones de manejo de memoria por medio del acceso al *bootloader*.

A continuación, se verificará el alcance que tiene cada uno de los comandos, los cuales son: escritura, lectura, borrado y ejecución.

Tabla 3.Comandos soportados por cada área de memoria. Obtenida de [28]

Memory area	Write command	Read command	Erase command	Go command
Flash	Supported	Supported	Supported	Supported
RAM	Supported	Supported	Not supported	Supported
System memory	Not supported	Supported	Not supported	Not supported
Data memory	Supported	Supported	Not supported	Not supported
OTP memory	Supported	Supported	Not supported	Not supported

En la Tabla 3 se pueden observar los comandos permitidos en cada región de memoria.

3.8.3 Instrucción de escritura.

Una vez que se ha accedido al *bootloader* del microcontrolador STM32F407 se puede ejecutar un comando de escritura en memoria el cual de acuerdo con [28] se revisará a detalle en esta sección.

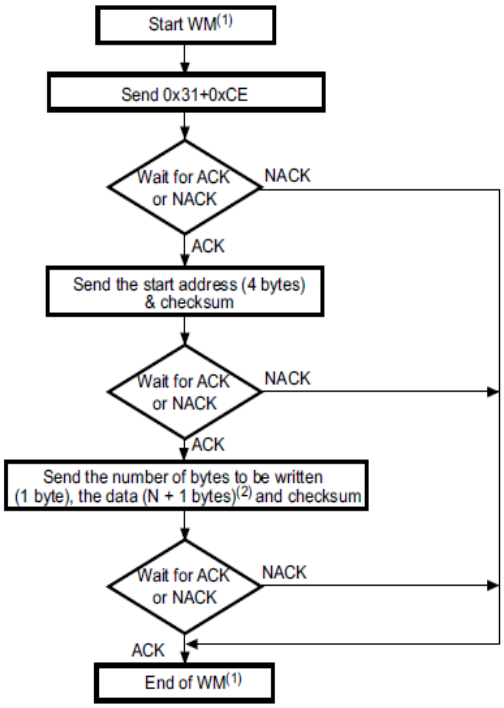

Figura 41. Comando de escritura [29]

En la Figura 41 se puede observar el diagrama de flujo correspondiente al comando de escritura visto desde el dispositivo que desea ejecutar el comando. El dispositivo debe enviar los siguientes datos:

- Byte 1. 0x31
- Byte 2. 0xCE (~0x31)
- Esperar por el byte de ACK, si se recibe continua la ejecución del comando, en caso contrario se sale de la ejecución del comando.
- Bytes del 3 al 6: Dirección de inicio (byte 3: MSB, byte 6: LSB)
- Byte 7: Checksum (byte3, byte4, byte5, byte6)
- Esperar ACK
- Byte 8: Número de bytes a escribir (N+1 bytes, máximo 256 bytes)
- N+1 Bytes de datos.
- Byte de *Checksum*: XOR (N, N+1 bytes de datos)

3.8.4 Instrucción de lectura.

Una vez que se ha accedido al *bootloader* del microcontrolador STM32F407 de acuerdo con [28] se puede ejecutar un comando de escritura en memoria el cual se revisará a detalle en esta sección

En la Figura 42 se puede observar el diagrama de flujo correspondiente al comando de escritura visto desde el dispositivo que desea ejecutar el comando. El dispositivo debe enviar los siguientes datos:

- Byte 1. 0x11
- Byte 2. 0xEE (~0x31)
- Esperar por el byte de ACK, si se recibe continua la ejecución del comando, en caso contrario se sale de la ejecución del comando.

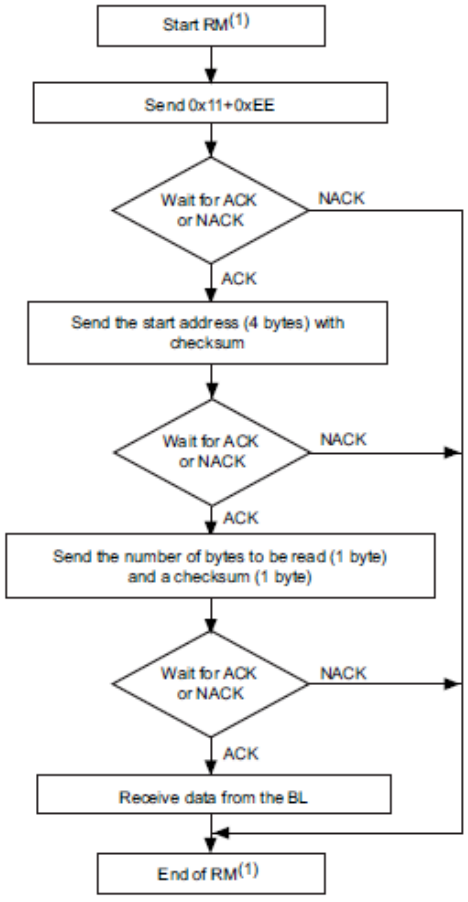

Figura 42. Ejecución del comando de lectura en el bootloader del uC STM32F407F407 [29]

- Bytes del 3 al 6: Dirección de inicio (byte 3: MSB, byte 6: LSB)
- Byte 7: Checksum XOR (byte3, byte4, byte5, byte6)
- Esperar ACK
- Byte 8: Número de bytes a recibir (N+1 bytes, máximo 256 bytes)
- Byte 9: byte de *Checksum*. XOR byte 8 (complemento del byte 8).

3.8.5 Instrucción del comando de borrado.

Tabla 4. Organización de la memoria flash del uC STM32F407 [29]

Block	Name	Block base addresses	Size
Main memory	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
	Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
	Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
	Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbytes
	Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
	Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
	Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
	. . .	. . .	. . .
	Sector 11	0x080E 0000 - 0x080F FFFF	128 Kbytes
System memory		0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP area		0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
Option bytes		0x1FFF C000 - 0x1FFF C00F	16 bytes

Una vez que se ha accedido al *bootloader* del microcontrolador STM32F407 se puede ejecutar un comando de borrado en memoria el cual, de acuerdo con [28], se revisará a detalle en esta sección.

Antes de analizar el comando de borrado como tal es necesario conocer la organización de la memoria flash del microcontrolador y los permisos de acceso que el *bootloader* tiene acceso a esta. En la Tabla 4 se puede ver que la memoria FLASH se organiza en 11 sectores. La operación de borrado solamente se puede realizar a nivel de sector. No es posible borrar solo una localidad de memoria, sino que se tienen que borrar todas las localidades correspondientes al sector. La función de borrado se puede ejecutar únicamente sobre la memoria principal, las regiones de memoria de sistema, región OTP y los bytes de opción quedan restringidos para esta operación.

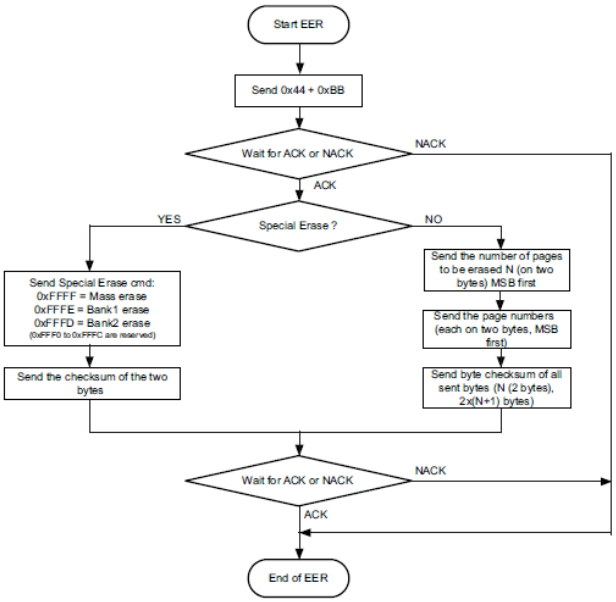

Figura 43. Ejecución del comando de borrado extendido en el bootloader del uC STM32F407F407 [29]

En la Figura 43 se puede observar el diagrama de flujo correspondiente al comando de borrado visto desde el dispositivo que desea ejecutar el comando. El dispositivo debe enviar los siguientes bytes para poder realizar una operación de borrado sobre la memoria flash del microcontrolador:

- Byte 1. 0x44
- Byte 2. 0xBB (~0x44)
- Esperar por el byte de ACK, si se recibe continua la ejecución del comando, en caso contrario se sale de la ejecución del comando.
- Byte 3-4: Borrado especial (0xFFFF, 0xFFFE o 0xFFFD) o Número de páginas a ser borradas (N+1).
- Bytes restantes: *checksum* de bytes 3-4 o (2 x (N+1)) bytes (números de páginas codificados en dos bytes, primero MSB) después el *checksum* de los bytes 3-4 y todos los restantes.

3.8.6 Instrucción de ejecución

Una vez que se ha accedido al *bootloader* del microcontrolador STM32F407 se puede ejecutar un comando de ejecución de programa el cual permite salir del *bootloader* y además ejecutar un programa cargado a partir de cierta localidad de memoria. En esta sección, de acuerdo con [28], se revisará el funcionamiento de este comando.

En la Figura 44 se puede observar el diagrama de flujo correspondiente al comando de ejecución de programa visto desde el dispositivo que desea ejecutar el comando. El dispositivo debe enviar los siguientes bytes para poder realizar una operación de borrado sobre la memoria FLASH del microcontrolador:

- Byte 1. 0x21
- Byte 2. 0xDE (~0x21)
- Esperar por el byte de ACK, si se recibe continua la ejecución del comando, en caso contrario se sale de la ejecución del comando.

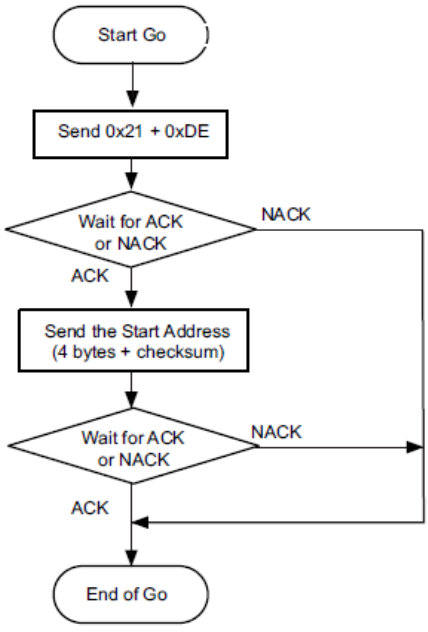

Figura 44. Ejecución del comando de ejecución de programa en el bootloader del uC STM32F407F407 [29]

- Bytes del 3 al 6: Dirección de inicio (byte 3: MSB, byte 6: LSB)
- Esperar por el byte de ACK, si se recibe continua la ejecución del comando, en caso contrario se sale de la ejecución del comando.
- La ejecución del comando ha finalizado.

Después de ejecutar el comando *Go* el microcontrolador saldrá del *bootloader* y cargará el programa cargado a partir de la dirección que hemos proporcionado.

3.9 Formato Intel Hex.

El programa de un microcontrolador al más bajo nivel no es más que unos y ceros cargados en la región de código del flash, el microprocesador lee estás instrucciones de manera secuencial, lo que da lugar a la ejecución del programa. El formato Intel Hex contiene la información necesaria para cargar un programa a un microcontrolador, esta información está compuesta principalmente por datos y la localidad de memoria en la que se tienen que cargar.

A continuación, se presenta un ejemplo:

```
:10008000AF5F67F0602703E0322CFA92007780C361
:10009000089001C6B7EA7CA9200FE10D2AA00477D81
:0B00A00080FA92006F3600C3A00076CB
:00000001FF
```

Cada línea del formato Intel hex se contiene la siguiente información:

- Primer carácter (:) = principio de segmento.
- Siguientes dos caracteres = Longitud del segmento.
- Siguientes cuatro caracteres = Dirección de memoria.
- Tipo de segmento.
- Siguientes N caracteres = datos
- Últimos dos caracteres = Checksum.

Los tipos de segmento admitidos por el formato Intel Hex se presentan a continuación:

- 00 – Segmento de datos.
- 01 – Fin de archivo.
- 02 – Dirección extendida de segmento.
- 03 – Dirección de comienzo de segmento.
- 04 – Segmento de dirección extendida.
- 05 – Inicio de dirección lineal.

4 Diseño y Construcción.

En esta sección, con base en el estado del arte, se determinará cada uno de los bloques funcionales del sistema de monitoreo y corrección de fallas. Con base en los requerimientos del proyecto se buscarán los elementos que mejor se acoplen, haciendo las consideraciones necesarias en cuanto a tecnología, resistencia a la radiación, capacidad de procesamiento y facilidad de obtención. Se busca desarrollar un proyecto que cumpla con lo requerido y además que su implementación resulte viable.

El diseño seguirá una metodología descendente. El diseño se dividirá en los siguientes capítulos:

- Diseño conceptual. En este capítulo se hará una revisión breve del proyecto del que forma parte el trabajo de esta tesis para así entender los problemas que se desean resolver. Se presentará un primer diseño conceptual que pretende solucionar el problema, sin embargo, el diseño será únicamente a nivel cualitativo.
- Diseño a nivel sistema. A partir del diseño conceptual se determinarán cada uno de los módulos que componen al sistema con sus respectivos requerimientos a un nivel cuantitativo.
- Diseño de detalle. Se presentará como se ha diseñado cada uno de los módulos definidos en la sección anterior, justificando la elección del hardware, pero sobre todo haciendo énfasis en la implementación de técnicas de corrección y detección de fallas por medio de software.
- Pruebas y refinamiento. Se reportarán las pruebas que se han realizado en el sistema diseñado y se determinarán las áreas de oportunidad para un trabajo a futuro.

4.1 Diseño conceptual.

4.1.1 Descripción del proyecto.

El proyecto consiste en un SCMI el cual como requerimiento tendrá como unidad de principal de procesamiento al microcontrolador STM32F407VG, fabricado por ST Microelectronics, el cual tiene las capacidades suficientes para realizar las tareas requeridas sin embargo su implementación genera un problema ya que no es resistente a la radiación y debido a las condiciones altamente radioactivas del espacio exterior tiene una probabilidad alta de fallo debido a eventos SEE.

Este proyecto de tesis busca una solución ante el problema anterior, proponiendo un sistema de monitoreo, detección y corrección de fallas aplicado al microcontrolador principal. En el caso particular de este proyecto el sistema a diseñar tendrá que desempeñar las siguientes actividades:

- Capacidad de almacenamiento de un código de respaldo del microcontrolador principal.
- Capacidad de aplicar técnicas de tolerancia a fallas en el microcontrolador principal.
- Capacidad de recibir comandos de la computadora principal de la misión

4.1.2 Diseño a nivel concepto.

Como se ha visto en el estado del arte, algunas de las arquitecturas implementan un microcontrolador supervisor que vigila que el estado funcional de la computadora principal se encuentra bien. Debido a las condiciones de la misión explicadas en 4.1.1 este proyecto seguirá la misma tendencia de utilizar un microcontrolador supervisor.

Por otra es parte es importante mencionar que los recursos energéticos en un satélite son limitados y por lo tanto es necesario que el consumo de energía del sistema a diseñar sea lo más bajo posible. Para cumplir con los requerimientos energéticos de la misión se seleccionarán elementos que permitan la operación en

bajo consumo de energía. Por otra parte, es necesario tener memorias redundantes para poder tener un respaldo del código del microcontrolador maestro, y poderlo reprogramar en caso de que la memoria flash sea corrompida debido a un SEE.

El esquema de supervisión por medio de un microcontrolador supervisor debe tener las siguientes características:

- Construcción con tecnología resistente a la radiación.
- Suficientes módulos de comunicación serial para la comunicación con los siguientes elementos.
    - Computadora a bordo.
    - Microcontrolador principal.
    - Bancos de memoria.
- Bajo consumo energético.
- Poder de procesamiento suficiente para aplicar técnicas de tolerancia a fallas en el microcontrolador principal.

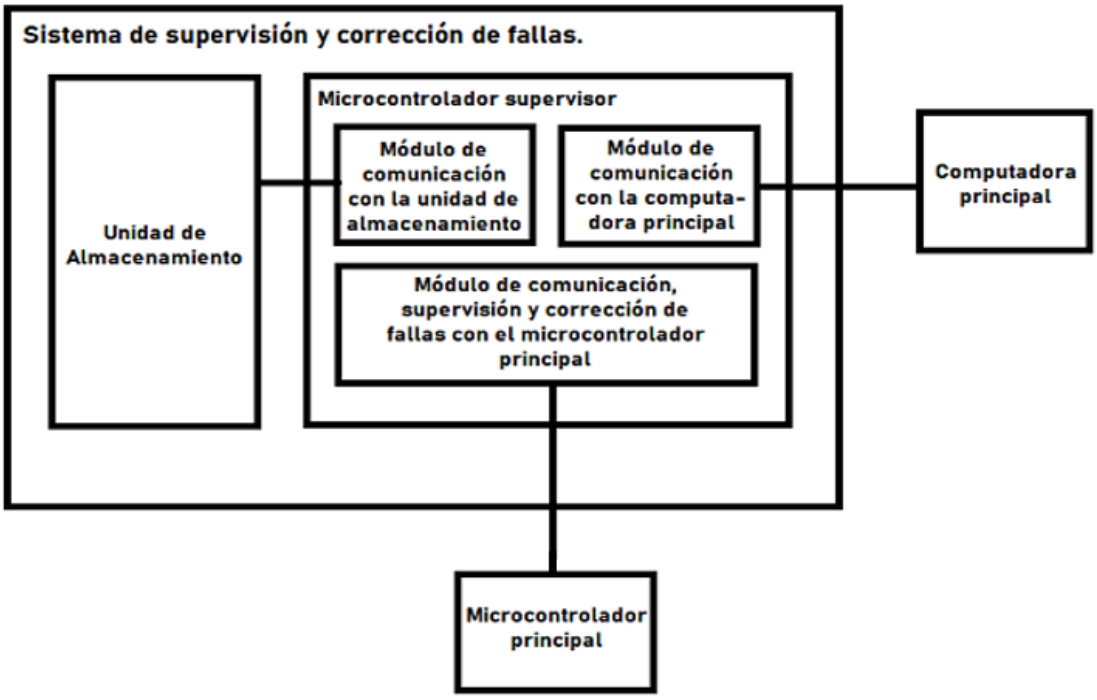

Figura 45. Diagrama a bloques del diseño conceptual

En la Figura 45 se puede observar el diseño conceptual propuesto, consiste en dos grandes bloques: La unidad de almacenamiento y el microcontrolador supervisor, el cual a su vez se divide en tres módulos (Interfaz con la unidad de almacenamiento, interfaz con la computadora principal y el módulo de supervisión y corrección de fallas. A continuación, se presentan los requerimientos de cada uno de los módulos del sistema:

- Unidad de almacenamiento:
  - La unidad de almacenamiento deberá ser lo suficientemente grande para respaldar el programa del microcontrolador principal.
  - La unidad de almacenamiento deberá ser resistente a la radiación.
  - La unidad del almacenamiento deberá emplear un protocolo de comunicación serial.
- Microcontrolador supervisor
  - El microcontrolador deberá ser resistente a la radiación espacial.
  - El microcontrolador deberá contar con dos módulos de comunicación suficientes para implementar las interfaces requeridas.
  - El microcontrolador deberá contar con un RTC.
  - El microcontrolador deberá tener un bajo consumo energético.
  - El microcontrolador deberá contar con la capacidad suficiente para aplicar técnicas de tolerancia en fallas en el microcontrolador principal.

4.2 Diseño a nivel sistema.

Una vez que en 4.1 se ha planteado el diseño a nivel concepto, en esta sección se planteará el diseño a nivel sistema, para así más adelante dar pie a la selección de componentes.

4.2.1 Definición de las interfaces.

Como se ha visto en la Figura 45 el sistema tiene tres interfaces:

- Interfaz con el sistema de almacenamiento.
- Interfaz con el microcontrolador maestro.

- Interfaz con un dispositivo externo.

Una vez que se saben las interfaces del sistema a diseñar, se procederá a determinar los requerimientos de cada una.

Debido a que la interfaz con el sistema de almacenamiento debe tener la capacidad de transmitir una gran cantidad de datos, es importante seleccionar un protocolo de comunicación que permita tener una transmisión de datos alta.

Tabla 5. Velocidades de transmisión de protocolos de comunicación

Protocolo de comunicación	Velocidad
UART	5 Mbps
SPI	10 Mbps
I2C	400 kbps

En la Tabla 5 se muestra una comparativa de las velocidades de transmisión alcanzables por UART, I2C y SPI, como se puede apreciar la mayor velocidad de transmisión se puede alcanzar con SPI, debido a esto y a que el protocolo de comunicación más utilizado con memorias es SPI, se elegirá este protocolo para la comunicación con el dispositivo de almacenamiento.

Respecto a la interfaz con el microcontrolador principal el criterio de selección es cual protocolo de comunicación implementa el microcontrolador para la ejecución de comandos por medio de su *bootloader*. De acuerdo con 3.8 uno de los protocolos utilizados para el acceso al *bootloader* es UART, por lo tanto, se utilizará este protocolo para la interfaz con el microcontrolador maestro.

Respecto a la interfaz con un dispositivo externo, debido a que no se estarán transmitiendo datos todo el tiempo y a la sencillez de su implementación se implementará UART. En la Tabla 6 se muestran los protocolos de comunicación elegidos para cada una de las interfaces.

Tabla 6. Protocolos de comunicación elegidos para cada una de las interfaces.

Interfaz	Protocolo de comunicación elegido
Dispositivo de almacenamiento	SPI
Microcontrolador maestro	UART
Dispositivo externo.	UART

4.2.2 Componentes.

Una vez que en 4.1 se ha propuesto el diseño conceptual, en esta sección se seleccionará la tecnología necesaria para cada uno de los componentes, por ejemplo, la tecnología de construcción del dispositivo de almacenamiento que resulte óptima para la aplicación requerida.

Respecto al microcontrolador principal, de acuerdo con lo explicado en 4.1.1 es un STM32F407VG07.

Por otra parte, respecto al microcontrolador maestro, como se ha visto en 2, la serie de microcontroladores MSP430FR es ampliamente utilizada en esquemas de supervisión, debido a que su unidad de almacenamiento está hecha con tecnología ferroeléctrica y como se ha visto en 3.6.4, las FRAM tienen un desempeño muy bueno en ambientes de radiación, por lo anterior se elegirá un microcontrolador de la serie MSP430FR, una vez que se ha determinado la familia a implementar, el modelo en específico tiene que contar con las interfaces presentadas en la Tabla 8 y además contar con los periféricos necesarios para desempeñar el esquema de supervisión, a continuación se enlistan los requerimientos:

- 2 UART
- 1 SPI
- 1 RTC
- 1 Timer.

Finalmente, para la unidad de almacenamiento, como ya se ha mencionado anteriormente las memorias FRAM tienen un gran desempeño en ambientes de

radiación, debido a esto se implementará una memoria bajo está tecnología. Otro criterio importante que mencionar es que la capacidad de la memoria también resulta un parámetro de interés ya que debe ser lo suficientemente grande para almacenar por lo menos una copia del programa del microcontrolador maestro, de acuerdo con [29] el tamaño de la memoria flash es de 1 Mbyte, por lo tanto, este es el tamaño máximo de programa que se puede almacenar, por lo tanto, la memoria FRAM debe ser mayor a 1 Mb. En la Tabla 7, de manera general, se muestra un resumen de los requerimientos necesarios para los dispositivos a emplear en el sistema de supervisión.

Tabla 7. Requerimientos generales de los dispositivos a usar en el sistema de supervisión

Dispositivo	Tecnología o familia	Requerimientos
Microcontrolador maestro	STM32F407VG0	-
Microcontrolador supervisor	MSP430FRxxxx	• 1 SPI • 2 UART • 1 RTC • 1 Timer
Unidad de almacenamiento	FRAM	• Almacenamiento mayor a 1 Mb

4.2.3 Arquitectura.

Finalmente, el diseño a nivel sistema se puede observar en la Figura 46 el cual más adelante servirá para seleccionar los componentes específicos que serán parte de la arquitectura.

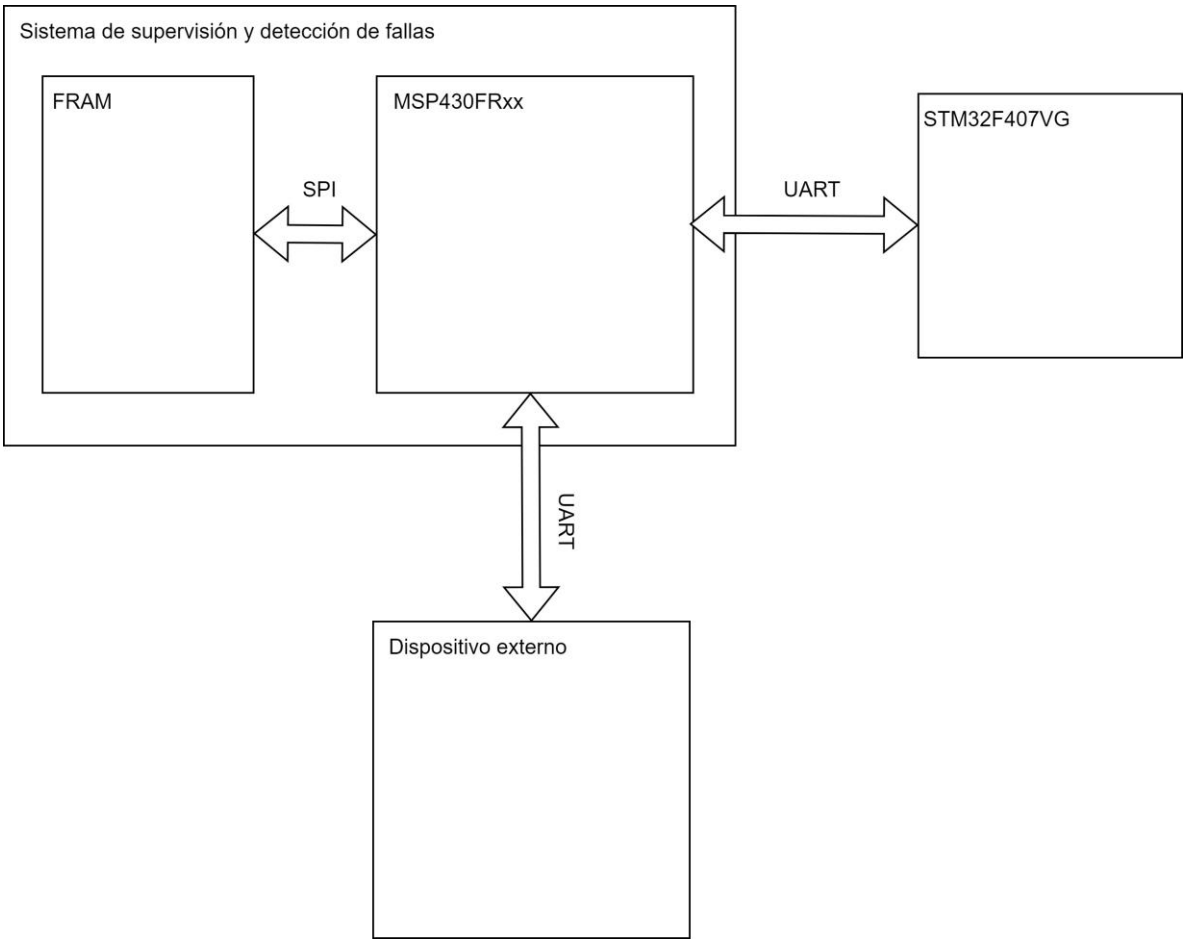

Figura 46. Diseño a nivel sistema

4.3 Diseño de detalle – hardware.

En esta sección se profundizará en la selección de componentes para la arquitectura propuesta teniendo en cuenta las características de la misión y el ambiente de radiación espacial al que se someterá el satélite. Para determinar el medio ambiente espacial se realizará una simulación de la misión utilizando la plataforma de SPENVIS la cual es creada por la ESA y además es de uso libre. Una vez teniendo la simulación del medio ambiente de radiación espacial se seleccionarán los componentes requeridos para la arquitectura propuesta teniendo el LET_{TH}, el cual como se ha visto en 3.3.3 resulta muy importante para la selección de componentes electrónicos que trabajarán en ambientes de radiación, como lo es el ambiente del espacio exterior.

4.3.1 Simulación del ambiente de radiación espacial de la misión.

Antes de plantear las especificaciones es necesario tener una aproximación del ambiente radioactivo espacial donde se desempeñará la misión, esta aproximación se obtendrá por medio de una simulación realizada en el software SPENVIS el cual es desarrollado por la ESA y además es de uso libre. SPENVIS permite obtener el flujo de partículas radioactivas debidas al medio ambiente de radiación espacial que experimentará una misión, teniendo en cuenta la duración de la misión, la órbita del satélite, y las fuentes de radiación presentes en esa órbita. Una vez obtenidos los datos de interés, se tendrán los parámetros necesarios para elegir adecuadamente los componentes principales del sistema de monitoreo, detección y corrección de fallas.

A continuación, se realizará una simulación para un nanosatélite que orbitará sobre LEO durante un año. Es importante mencionar que se ha elegido a LEO como órbita a simular, ya que de acuerdo con 3.1.1 es la órbita más utilizada para misiones de percepción remota de imágenes.

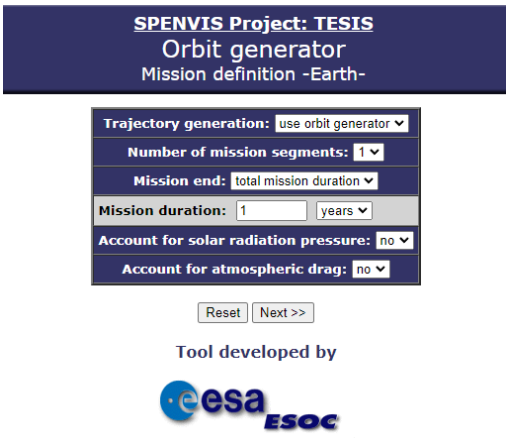

Figura 47. SPENVIS - Simulación de la órbita de la misión

En la Figura 47 se pueden apreciar los primeros parámetros para la simulación de la órbita de la misión que compete a esta tesis. Se puede apreciar que se tendrá un único segmento de la misión para todo el análisis, por otra parte, la simulación se

realizará para un año, considerando que el satélite estará orbitando la Tierra durante este tiempo.

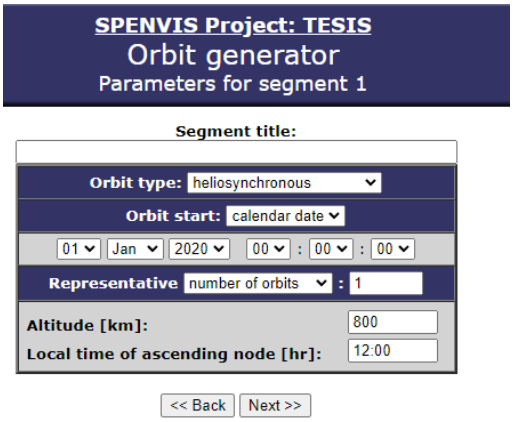

Figura 48. SPENVIS - Configuración de la simulación

Debido a que la misión tiene como objetivo el envío remoto de fotografías para el estudio de los efectos del cambio climático, se seleccionará como órbita a LEO y particularmente se seleccionará una órbita helio síncrona debido a las necesidades de la misión, la simulación se realizará para un año. En la Figura 48 se puede ver la configuración de la órbita a generar, para que la presencia de luz sea máxima se ha elegido las 12:00 pm como hora local en el nodo ascendente, lo que significa que cuando el satélite atreviese el ecuador de sur a norte, siempre lo hará a las 12 am.

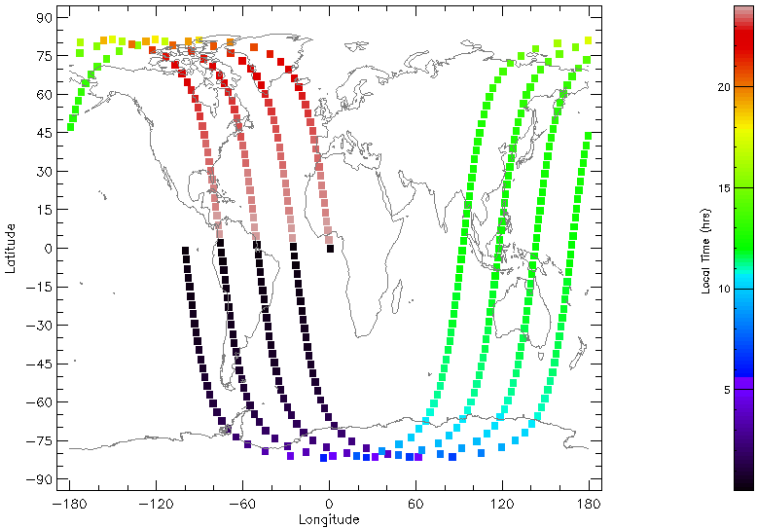

Figura 49. SPENVIS - Proyección de la órbita sobre la Tierra

En la Figura 49 podemos ver una proyección de la órbita generada sobre el mapa de la Tierra, como se puede ver el satélite cruza el ecuador de sur a norte a las 12 am.

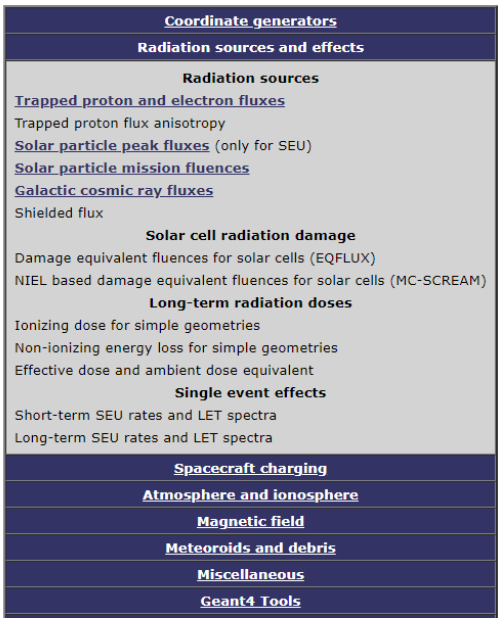

Figura 50. SPENVIS

En la Figura 50 se pueden ver las fuentes de radiación que por medio de modelos se pueden simular en SPENVIS. En este caso se simularán los flujos debidos a electrones y protones atrapados (cinturones de Van Allen), partículas solares y rayos cósmicos galácticos.

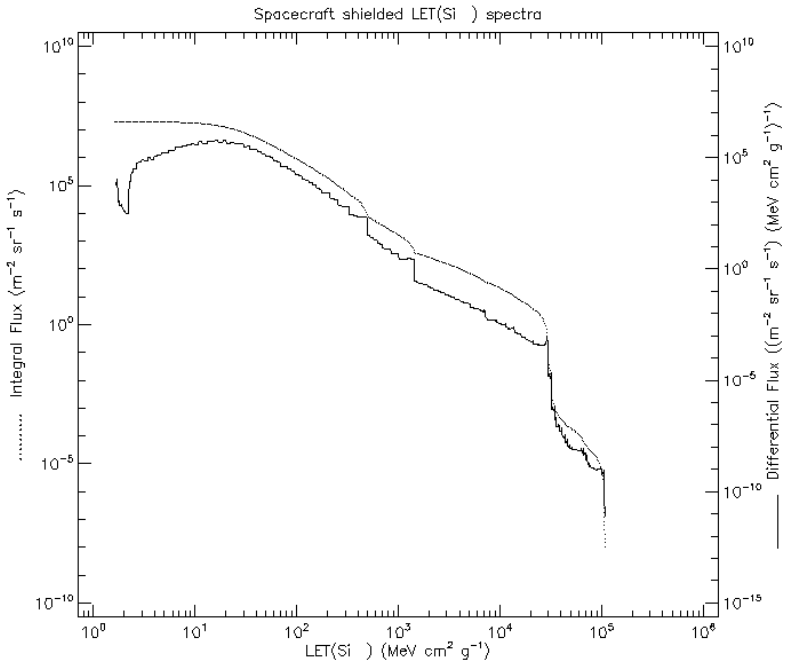

Figura 51. SPENVIS - Flujo integral de partículas LET

En la Figura 51 se puede ver el flujo integral de partículas LET el cual esta descrito por la curva punteada y ha sido obtenido con base a los parámetros anteriormente introducidos en SPENVIS. Se puede ver que las partículas de mayor energía se encuentran menos presentes que las de menor energía, sin embargo, a pesar de que se encuentran en pequeñas medidas es importante considerarlas a la hora de la selección de los componentes. Con la información contenida en la Figura 51 se puede obtener la frecuencia con la que las partículas cargadas energéticamente impactarán al satélite, y por lo tanto analizando este dato junto con el LETth de los dispositivos, se podrá determinar la frecuencia de ocurrencia de SEE en los dispositivos.

4.3.2 Unidad de almacenamiento.

Como se ha visto en 3.6.4 y 3.6.5 las memorias FRAM tienen un gran desempeño en ambientes de radiación espacial, además de esto tienen una gran cantidad de ciclos de escritura y borrado en comparación a otras tecnologías de memorias no volátiles. Debido a los puntos anteriores, se utilizará una memoria FRAM, particularmente la Cypress CY15B104Q, por lo tanto, a continuación, se realizarán

los cálculos para justificar el uso de esta memoria en la misión. De acuerdo con [30] las memorias con la misma tecnología que la FRAM Cypress CY15B104Q tienen un

$$LET_{TH} = 90 \frac{MeV\ cm^2}{mg}$$

En la Figura 51 el LET está expresado en:

$$\frac{MeV\ cm^2}{g}$$

Por lo que será necesario representar el LET_{TH} con estas unidades como se muestra a continuación:

$$LET_{TH} = 90\ \text{MeV}\frac{cm^2}{mg}\left(\frac{1\ mg}{0.001\ g}\right) = 90\ 000\frac{MeVcm^2}{g}$$

$$LET_{TH} = 90\ 000\frac{MeVcm^2}{g}$$

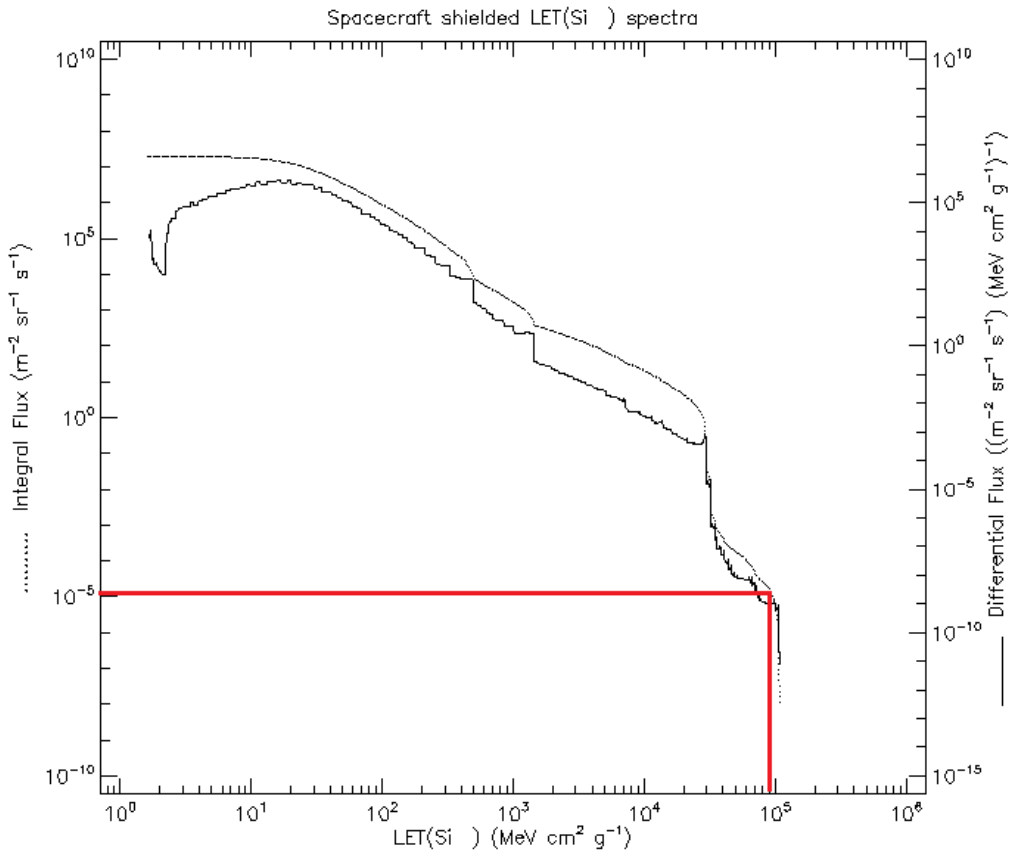

Figura 52. Flujo integral de partículas LET – Memoria FRAM

Finalmente, analizando la Figura 52 se puede apreciar que el flujo integral para el LET_{TH} corresponde a:

$$integral\ flux\ =\ 10^{-5}\ \frac{iones}{m^2\ sr\ s}$$

No se ha encontrado el área sensible para de la memoria FRAM, por lo tanto, se estimará la sección transversal saturada, utilizando la escala de integración de las FRAM y además el número de celdas de memoria. De acuerdo con [31], el tamaño de una celda de memoria FRAM es de:

$$L\ =\ 130\ nm$$

Se puede calcular la sección transversal saturada del integrado de la siguiente manera:

$$A\ =\ (Número\ de\ celdas\ de\ memoria)(Area\ de\ la\ celda)$$

Cada celda puede almacenar un bit, por lo tanto, el número de celdas es igual al tamaño de la memoria en bits.

$$Número\ de\ celdas\ de\ memoria\ =\ 4\ MB\ (8) =\ 32\ 000\ 000$$

Finalmente, la sección transversal saturada del integrado es:

$$A\ =\ (32\ 000\ 000)((130\ nm)^2)$$

$$A\ =\ 540.8x10^{-9}\ m^2$$

Debido a que el flujo de partículas provendrá de todos lados, se deben considerar tanto como la superficie superior e inferior del integrado, por lo tanto

$$A_{omnidireccional} =\ \ 1.08x10^{-6}\ m^2$$

A continuación, se calculará el tiempo en el que un ion con una energía suficiente para causar un SEE puede presentarse en la denominada área sensible.

$$tiempo\ entre\ upset\ =\ \frac{1}{4\pi(1.08x10^{-6}m^2)(10^{-5}\ \frac{iones}{m^2\ sr\ s})}$$

$$tiempo\ entre\ upset\ =\ \ 7.3682x10^9\ S\ =\ 233.64\ años$$

Como se puede ver el tiempo entre upset resulta mucho mayor que la duración de la misión, por lo tanto, se puede considerar a la memoria FRAM propuesta como un dispositivo fiable para la implementación en la misión ya que la probabilidad de falla es prácticamente nula.

4.3.3 Microcontrolador supervisor.

Como se ha observado en el estado del arte, microcontroladores de Texas Instruments de la serie MSP430FR han sido implementados en las misiones que se plantean en 2.1.2 y 2.1.3, debido a la herencia de vuelo que tiene esta serie de microcontroladores se propondrá el siguiente microcontrolador: MSP430FR5969. A continuación de acuerdo con 4.3.1 se realizará el análisis para determinar si el uso

de este microcontrolador resulta viable para la implementación en la misión que compete a esta tesis.

De acuerdo con 3.7 se tiene que el microcontrolador MSP430FR5969 tiene un:

$$LET_{TH} = 72 \text{ MeV } \frac{cm^2}{mg}$$

En la Figura 51 el LET está expresado en:

$$\frac{MeV\ cm^2}{g}$$

Por lo que será necesario representar el LET_{TH} con estas unidades como se muestra a continuación:

$$LET_{TH} = 72\text{MeV}\frac{cm^2}{mg}\left(\frac{1\ mg}{0.001\ g}\right) = 72\frac{GeVcm^2}{g}$$

$$LET_{TH} = 72\frac{GeVcm^2}{g}$$

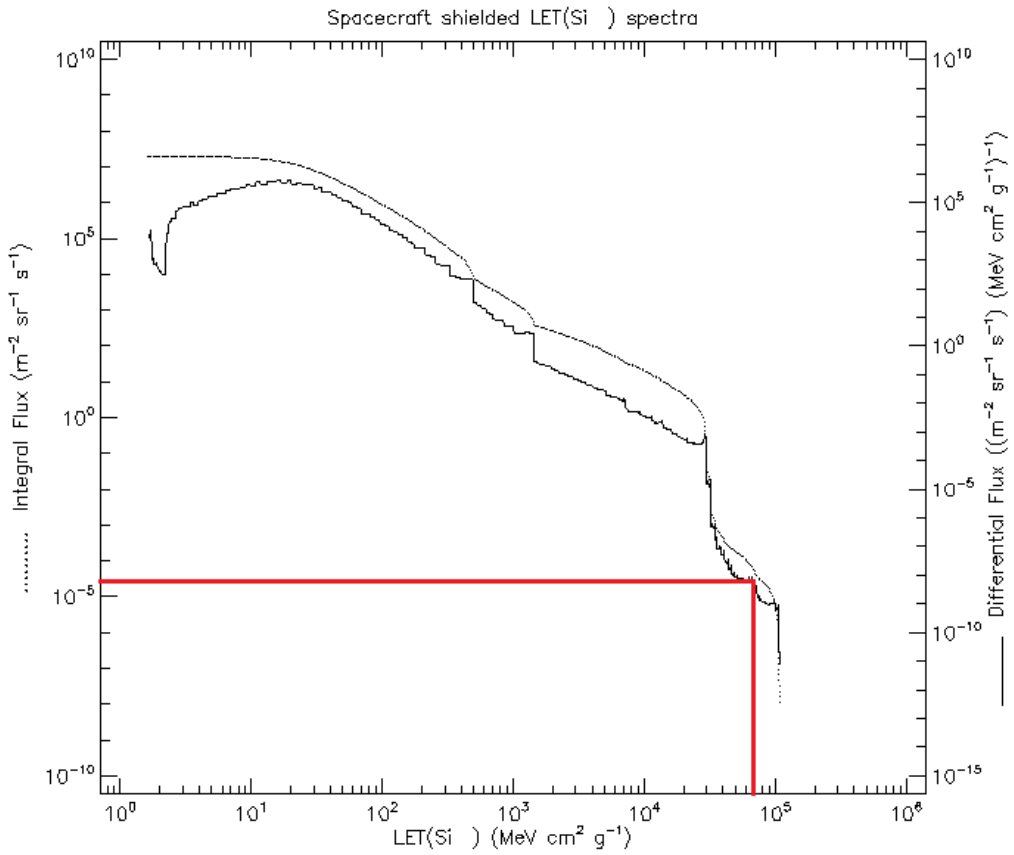

Figura 53. Flujo integral de partículas LET - Microcontrolador supervisor

Finalmente, analizando la Figura 53 se puede apreciar que el flujo integral para el LET_{TH} corresponde a:

$$integral\ flux\ =\ 5^{-5}\ \frac{iones}{m^2\ sr\ s}$$

No se ha encontrado el área sensible para el microcontrolador MSP430FR5069, por lo tanto, se estimará el área sensible del integrado, de acuerdo con [31], el tamaño de una celda de memoria FRAM es de

$$L\ =\ 130\ nm$$

Se puede calcular la sección transversal saturada del integrado de la siguiente manera:

$$A\ =\ (Número\ de\ celdas\ de\ memoria)(Area\ de\ la\ celda)$$

Cada celda puede almacenar un bit, por lo tanto, el número de celdas es igual al tamaño de la memoria en bits.

$$Número\ de\ celdas\ de\ memoria\ =\ 64\ kB\ (8) =\ 512\ 000$$

Finalmente, la sección transversal saturada del integrado es:

$$A\ =\ (512\ 000)((130\ nm)^2)$$

$$A\ =\ 8.65x10^{-9}\ m^2$$

Debido a que el flujo de partículas provendrá de todos lados, se deben considerar tanto como la superficie superior e inferior del integrado, por lo tanto, la sección transversal saturada considerando un flujo omnidireccional resulta la siguiente:

$$A_{omnidireccional} =\ 17.3x10^{-9}\ m^2$$

A continuación, se calculará el tiempo en el que un ion con una energía suficiente para causar un SEE puede presentarse en la denominada área sensible.

$$tiempo\ entre\ upset\ =\ \frac{1}{4\pi(17.3x10^{-9}m^2)(5^{-5}\ \frac{iones}{m^2\ sr\ s})}$$

$$tiempo\ entre\ upset\ =\ 91.99x10^9\ S = 2\ 916.98\ años$$

Como se puede ver el tiempo entre upset resulta mucho mayor que la duración de la misión, por lo tanto, se puede considerar al microcontrolador propuesto como un dispositivo fiable para su implementación en la misión.

4.3.4 Microcontrolador maestro.

De acuerdo con 4.1.1 el microcontrolador STM32F407VG será utilizado como microcontrolador principal de la misión, por lo tanto, es necesario hacer un análisis del desempeño de este microcontrolador en ambientes de radiación, la integridad de la memoria flash resulta critica ya que es ahí donde se almacena el programa que lleva a cabo la misión del satélite, debido a lo expuesto anteriormente el desempeño del STM32F407 en ambientes de radiación se ve resumido al

desempeño de su FLASH en ambientes de radiación. De acuerdo con [32] las memorias FLASH con una alta escala de integración (25 nm) comienzan a comprometer su integridad presentando errores a partir de:

$$LET_{TH} = 1\ MeV\ \frac{cm^2}{mg}$$

En la Figura 51 el LET está expresado en:

$$\frac{MeV\ cm^2}{g}$$

Por lo que será necesario representar el LET_{TH} con estas unidades como se muestra a continuación:

$$LET_{TH} = 1\ \text{MeV}\frac{cm^2}{mg}\left(\frac{1\ mg}{0.001\ g}\right) = 1000\frac{MeVcm^2}{g}$$

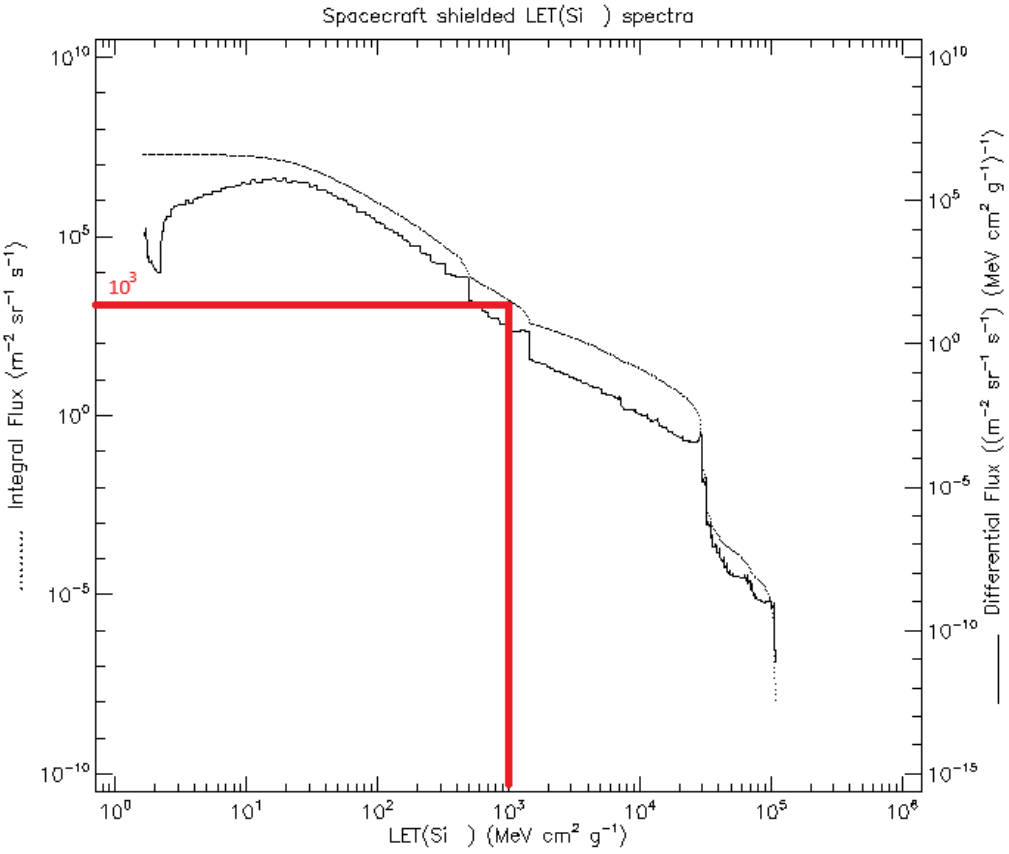

Figura 54. Flujo integral de partículas LET - Microcontrolador maestro

Analizando la Figura 54 se tiene que el flujo integral de partículas para ese LET es:

$$integral\ flux\ =\ 1000\ \frac{iones}{m^2\ sr\ s}$$

No se ha encontrado el área sensible para el microcontrolador STM32F407VG, por lo tanto, se estimará el área sensible del integrado, considerando una escala de integración de 65 nm

$$L\ =\ 22\ nm$$

Se puede calcular la sección transversal saturada del integrado de la siguiente manera:

$$A\ =\ (Número\ de\ celdas\ de\ memoria)(Area\ de\ la\ celda)$$

Cada celda puede almacenar un bit, por lo tanto, el número de celdas es igual al tamaño de la memoria en bits.

$$Número\ de\ celdas\ de\ memoria\ =\ 1MB\ (8)\ =\ 8\ 000\ 000$$

Finalmente, la sección transversal saturada del integrado es:

$$A\ =\ (8\ 000\ 000)((22\ nm)^2)$$

$$A\ =\ 3.872x10^{-9}\ m^2$$

Debido a que el flujo de partículas provendrá de todos lados, se deben considerar tanto como la superficie superior e inferior del integrado, por lo tanto, la sección transversal saturada considerando un flujo omnidireccional resulta la siguiente:

$$A_{omnidireccional}\ =\ 7.744x10^{-9}\ m^2$$

A continuación, se calculará el tiempo en el que un ion con una energía suficiente para causar un SEE puede presentarse en la denominada área sensible.

$$tiempo\ entre\ upset\ =\ \frac{1}{4\pi(7.744x10^{-9}m^2)(1000\ \frac{iones}{m^2\ sr\ s})}$$

$$tiempo\ entre\ upset\ =\ 102.76x10^3\ S = 28\ horas$$

Como se puede apreciar el tiempo entre upset es de 28 horas, mientras que la duración de la misión es de 1 año, debido a lo anterior se tiene que implementar un esquema de supervisión que permita mantener al microcontrolador en operación durante toda la duración de la misión. El esquema de supervisión deberá determinar el estado funcional del microcontrolador maestro

4.4 Selección de tarjetas y módulos de desarrollo

Una vez que en 4.3 se han seleccionado los dispositivos para implementarse en el proyecto, esta sección presentará el módulo FRAM y la tarjeta de desarrollo del MSP430FR5969 que se utilizará para la etapa de desarrollo del proyecto. Una vez teniendo en cuenta las características del módulo de la FRAM y la tarjeta de desarrollo se definirá la arquitectura de hardware, mapeando las interfaces a los puertos específicos para satisfacer las necesidades del proyecto.

4.4.1 Hardware.

4.4.1.1 Modulo Cypress CY15B104Q.

Se ha seleccionado una FRAM debido a su inherente resistencia a la radiación, particularmente se ha elegido la siguiente: Cypress CY15B104Q.

Figura 55. Módulo FRAM Cypress CY15B104Q

En la Figura 55 se puede observar el módulo fabricado por la empresa Mikroe, el cual tiene embebida la FRAM Cypress CY15B1104Q y todo lo necesario para su funcionamiento. Para ver más detalles de esta FRAM vea la sección 3.7.2. Es importante mencionar que además de ser resistente a la radiación, también permite la implementación de modos de bajo consumo lo cual es de gran utilidad en aplicaciones espaciales donde los recursos energéticos son limitados.

4.4.1.2 Tarjeta de desarrollo MSP-EXP430FR5969.

De acuerdo con lo visto en el estado del arte se ha decidido utilizar el microcontrolador MSP430FR6959 fabricado por Texas Instruments el cual también se utiliza en 2.1.3 y tiene una versión espacial que de acuerdo con [25] es altamente confiable.

Para el desarrollo del proyecto se utilizará la tarjeta de desarrollo MSP-EXP430FR5969 la cual se puede observar en la Figura 56

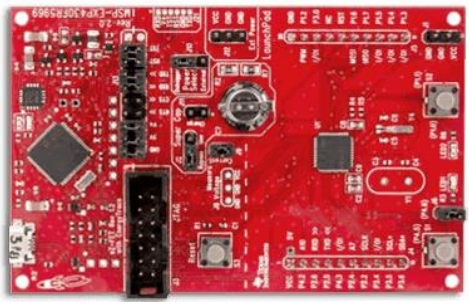

Figura 56. MSP-EXP430FR5969

El MSP-EXP430FR5969 incluye la interfaz para programación, depuración y mediciones de energía, incluye botones y leds embebidos para una interfaz sencilla con el usuario.

Por otra parte, de acuerdo con 3.7.1.7 la serie de microcontroladores MSP430 permite la implementación de modos de bajo consumo energético lo cual resulta de gran utilidad para aplicaciones espaciales donde los recursos son muy limitados.

4.4.1.3 Arquitectura de Hardware

Una vez que se han determinado los componentes a utilizar en el proyecto, teniendo en cuenta las características físicas de los dispositivos (distribución de puertos) y la arquitectura funcional descrita en la Figura 45 se puede determinar la arquitectura de hardware requerida para el proyecto.

Antes de determinar la arquitectura de hardware se hará un repaso sobre los requerimientos de hardware del proyecto, analizando principalmente las interfaces que debe tener el microcontrolador supervisor, las cuales se pueden observar en la Figura 45 y son las siguientes: interfaz con el microcontrolador maestro, interfaz con un dispositivo externo, interfaz con el banco de memorias.

Para la interfaz con el microcontrolador maestro es necesario considerar que se deben respetar los requerimientos de puertos para el acceso a su bootloader, los cuales de acuerdo con 3.8.1 son:

- Un puerto UART para el envío de comandos y datos hacia el bootloader del microcontrolador principal.
    - UART RX
    - UART TX
    - GND
- 3 GPIOS para realizar la secuencia de acceso al bootloader en los puertos (BOOT0, BOOT1 y NRST) del microcontrolador principal.

Para la interfaz del supervisor con las memorias FRAM es necesario considerar los requerimientos para la escritura y lectura de datos en la FRAM, los cuales de acuerdo con 3.7.2 son:

- Un puerto SPI para el envío de comandos y datos hacia las FRAM:
    - SPI MOSI
    - SPI MISO
    - SPI CLK
    - SPI CS

- GND

Para la interfaz del supervisor con el dispositivo externo se implementará un puerto UART, ya que de acuerdo con 3.7.1 es el único periférico de comunicación serial disponible, ya que el SPI y uno de los UART ya han sido utilizados por las interfaces anteriores.

En la Tabla 8 se pueden observar todos los puertos requeridos para la implementación de las interfaces requeridas. De acuerdo con la distribución de puertos del MSP430FR5969 presentada en [33] y las características de la tarjeta de desarrollo presentadas en [29] se determinará que puertos específicos del MSP430FR5969 serán utilizados para la realización de este proyecto.

Tabla 8. Resumen del hardware necesario para la implementación de las interfaces requeridas.

	Puertos requeridos

Interfaz con el microcontrolador maestro	• Modulo UART.
	  ○ TX
	  ○ RX
	  ○ GND
	• GPIO
	  ○ NRST
	  ○ BOOT0
	  ○ BOOT1

Interfaz con las memorias FRAM	• Modulo SPI
	  ○ MISO
	  ○ MOSI
	  ○ CLK
	  ○ CS

Interfaz con un dispositivo externo	• Modulo UART.
	  ○ TX
	  ○ RX
	  ○ GND

Finalmente, la arquitectura de hardware se presenta en la Figura 57

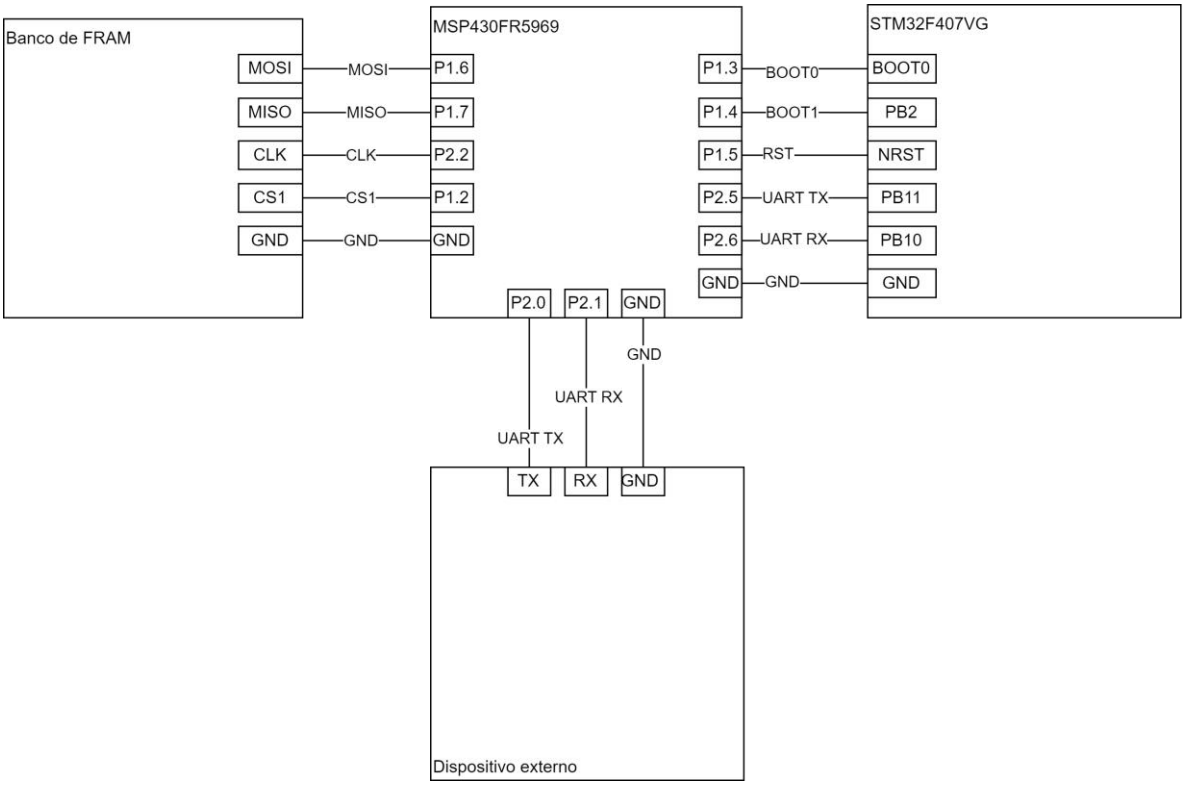

Figura 57. Arquitectura de hardware

En la Figura 57 se muestra la arquitectura final del hardware, hay cuatro bloques principales que a continuación se explicarán:

- Banco de FRAM. Contendrá la copia del programa del microcontrolador maestro. Interfaz:
- MSP430FR5969. Microcontrolador supervisor que determinará el estado del microcontrolador maestro, en caso de fallas, lo reprogramará utilizando la copia del programa que se encuentra almacenado en la FRAM.
- STM32F407VG. Microcontrolador maestro que lleva a cabo las funciones relacionadas a la misión.
- Dispositivo externo. Gracias a la interacción con el MSP430FR5969, el dispositivo externo podrá actualizar el código del uC maestro, primero sobrescribiendo el programa en la FRAM, para que posteriormente este programa sea cargado al microcontrolador maestro.

4.5 Diseño de detalle – software.

Antes de empezar a programar es importante primero determinar el diseño de la arquitectura de software para el sistema. La arquitectura de software debe estar diseñada para satisfacer los requerimientos que presenta el sistema de detección y corrección de fallas, para hacer un buen diseño de la arquitectura es necesario que los requerimientos sean claros, concisos y describan toda la funcionalidad del sistema. Debido a lo anterior en esta sección se plantearán los requerimientos del sistema para así poder diseñar la arquitectura, y posteriormente diseñar cada uno de los módulos que requerirá la arquitectura.

4.5.1 Requerimientos.

En esta sección se determinarán los requerimientos funcionales de la arquitectura de software. Para eso se tomarán como base los requerimientos mencionados en 4.1.1. Los requerimientos funcionales son los siguientes:

1) Reprogramar al microcontrolador maestro en caso fallas en su funcionamiento debidas a radiación.
2) Determinar el estado funcional del microcontrolador maestro.
3) Almacenar una copia del programa del microcontrolador maestro en la memoria FRAM.
4) Actualizar el código del microcontrolador maestro por medio de un dispositivo externo.

4.5.2 Diseño de la arquitectura.

Una vez que en 1), 2), 3) y 4) se han determinado los requerimientos funcionales, es necesario determinar los requerimientos de software que harán posible el funcionamiento requerido, a continuación, se determinarán los drivers de software necesarios para cumplir cada uno de los requerimientos funcionales.

Para el requerimiento 1) y 2) es necesario implementar una interfaz serial que permita la comunicación con el bootloader del microcontrolador maestro, de acuerdo con lo expuesto en 3.8, UART es un protocolo compatible, por lo tanto, es necesario desarrollar un driver de UART para la interfaz de comunicación con el microcontrolador maestro, además de esto es necesario realizar un driver para la ejecución de comandos en el bootloader del microcontrolador maestro.

Debido a que el esquema de supervisión será un *watchdog*, para el requerimiento 2) también es necesario la implementación de un *timer*, por lo tanto, es necesario desarrollar un driver para el uso de un *timer.*

Para el requerimiento 3) es necesario implementar una interfaz serial que permita la comunicación con la memoria FRAM, de acuerdo con 3.7.2, la interfaz se debe realizar por medio de SPI, por lo tanto, es necesario un driver de SPI y un driver para la ejecución de comandos en la FRAM.

Para el requerimiento funcional 4) son necesarios los drivers involucrados en los requerimientos 1) y 3), esto para poder almacenar en FRAM el código de respaldo y posteriormente cargarlo a la memoria flash del microcontrolador maestro. Además de esto, con el fin de establecer comunicación con el dispositivo externo, será necesario otro driver UART.

Tabla 9. Drivers necesarios para la arquitectura

Driver	Requerimiento funcional que satisface
UART1	1) y 2)
UART2	3) y 4)
SPI	3)
STM32F4xx bootloader	1) y 2)
FRAM	3) y 4)
Timer	2)

En la Tabla 9 se pueden observar los drivers necesarios para lograr la funcionalidad requerida, con esta información se puede hacer la primera aproximación a la

arquitectura de software requerida, es importante mencionar que para que el software sea escalable es necesario realizar un diseño modular, con esto en mente se presenta la siguiente arquitectura de software.

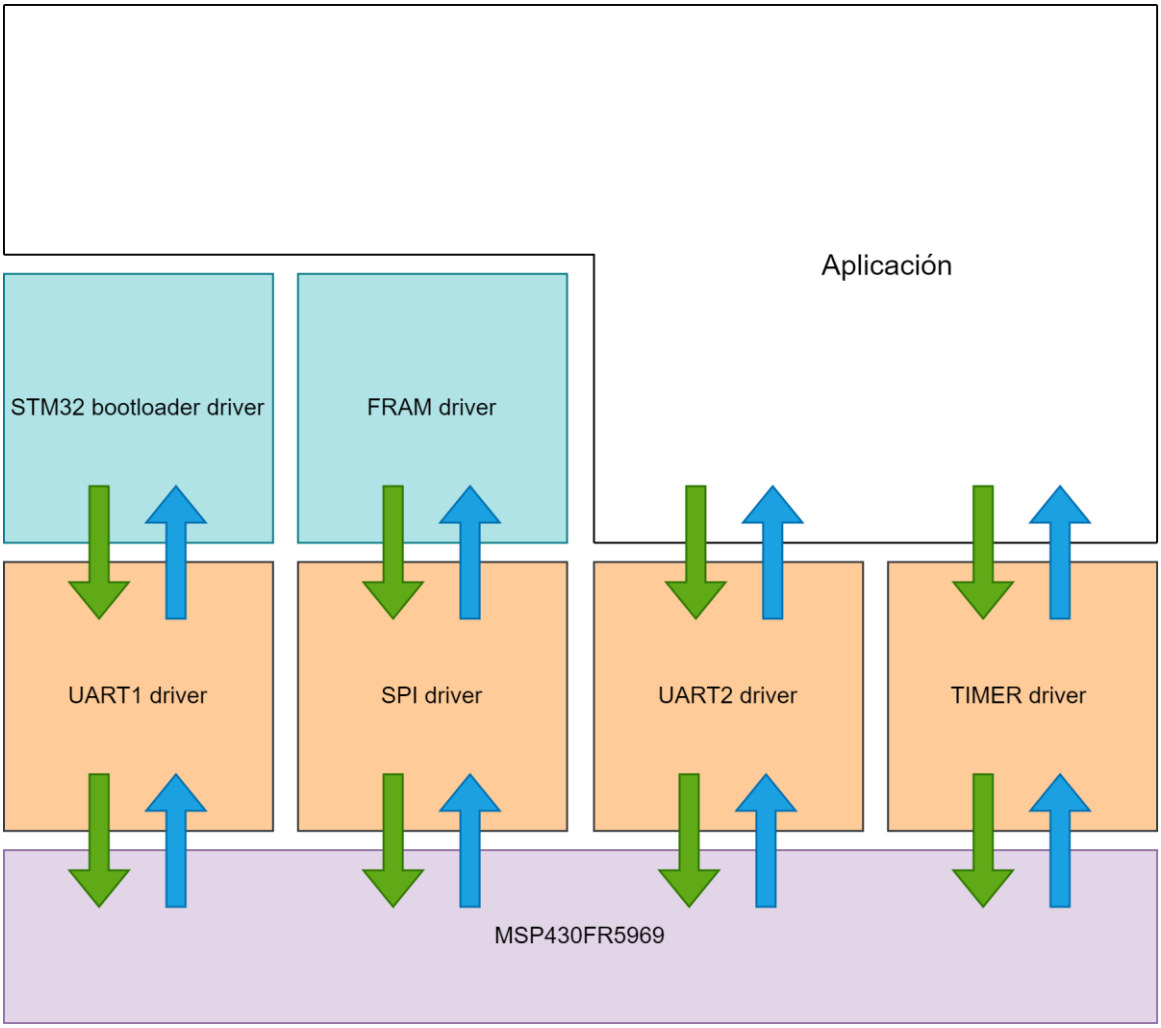

Figura 58. Arquitectura de software

En la Figura 58 se puede apreciar la arquitectura de software propuesta, la cual está compuesta por bloques de software. A continuación, se explicará la arquitectura:

- El bloque color morado corresponde a la representación del microcontrolador supervisor, el cual solo interactuará con los drivers de la capa superior (En color naranja).
- Los bloques color naranja corresponden a los drivers que interactuarán directamente con el microcontrolador o también conocidos como *bare metal*

drivers. Estos drivers son los únicos que interactuarán directamente con los registros del microcontrolador.

- Los bloques de color azul son *drivers* para el uso de la FRAM y el *bootloader* del microcontrolador maestro, su interacción con el microcontrolador es a través de los drivers *baremetal,* jamás interactúan directamente con los registros del microcontrolador. Su objetivo es eliminar la complejidad del control de los dispositivos para capas superiores.
- Finalmente, el bloque más grande y en color blanco es donde estará la aplicación que cumplirá los requerimientos funcionales presentados en 4.5.1, en este bloque de software es donde se implementan los algoritmos de que permitirán satisfacer los requerimientos, por ejemplo, la determinación del estado funcional del microcontrolador maestro.

La arquitectura propuesta busca un diseño modular que sea fácilmente adaptable, mantenible, y reduzca la complejidad de la aplicación.

4.5.3 Controladores del microcontrolador.

En esta sección se explicará el diseño de los controladores de bajo nivel que interactúan directamente con el microcontrolador MSP430FR5969. Los drivers por programar son los siguientes:

- UART0.
- UART1.
- SPI.
- TIMER.

A continuación, se determinarán las funciones que deben tener estos drivers de acuerdo con las especificaciones, ya que para acortar el tiempo de desarrollo solo se implementaran las funciones requeridas en los drivers para cumplir el objetivo del sistema de detección y corrección de fallas, sin embargo, la modularidad del código permitirá modificarlo de una manera fácil, si es que así se requiere.

4.5.3.1 eUSCIA1_UART

Empezando con el driver UART0 el cual será utilizado para la interfaz con el microcontrolador maestro, a continuación, se presentarán las funciones de este driver.

Tabla 10. Driver eUSCIA1_UART

eUSCIA1_UART	
Función	**Descripción.**
void eUSCIA1_UART_Init_Master_Reprog()	Inicializa UART en el módulo eUSCIA0 con la siguiente configuración: 9600 bps, 8 bits de datos, paridad impar, un bit de stop.
void eUSCIA1_UART_Init_ _MasterFunctionalStatus_Determination()	Inicializa UART en el módulo eUSCIA0 con la siguiente configuración: 9600 bps, 8 bits de datos, sin bit de paridad, un bit de stop.
void eUSCIA1_UART_send(uint8_t data_Tx);	Envía un byte por UART. Ejemplo: Enviar 0x10 por UART: eUSCIA0_UART_send(0x10)
uint8_t eUSCIA1_UART_receive()	Recibe un byte, controlado el flujo de datos por medio de banderas de interrupción. uint8_t dato = eUSCIA0_UART_receive()

En la Tabla 10 se pueden apreciar las funciones desarrolladas para el driver las cuales se alinean a los objetivos del driver, los cuales serán cumplidos por bloques superiores de software, es importante hacer notar que este driver contiene dos funciones de inicialización:

- eUSCIA1_UART_Init_Master_Reprog: Inicializa el driver para cumplir con la tarea de la reprogramación del microcontrolador maestro una vez que se determine que la memoria ha sido corrompida.
- eUSCIA1_UART_Init_MasterFunctionalStatus_Determination: Inicializa el driver para cumplir con la tarea de detección de fallas en el microcontrolador maestro.

Se tienen dos drivers de inicialización ya que durante el desarrollo de los drivers se descubrió que el módulo eUSCIA1 del microcontrolador supervisor tiene un problema con la activación del bit de paridad, ya que, al activarlo, el baudrate sufre alteraciones, por lo tanto, para la tarea de determinación del estado funcional del microcontrolador maestro se ha decidido que no se usará bit de paridad. Por otra parte, la tarea de reprogramación del microcontrolador maestro por medio de su bootloader requiere la implementación del bit de paridad, sin embargo, al tener un mecanismo de autodetección de baudrate por parte del bootloader del microcontrolador maestro, la inconsistencia en el baudrate generado por el microcontrolador supervisor al activar el bit de paridad, no tiene impacto funcional.

Por otra parte, se han desarrollado funciones de escritura y lectura de bytes, las cuales servirán a bloques superiores de software.

4.5.3.2 eUSCIA0_UART

El driver eUSCIA0_UART tiene la función de ser parte de la interfaz con el dispositivo externo, debido a que este dispositivo deberá ajustarse a los requerimientos proporcionados por la arquitectura supervisora, la configuración de la comunicación UART es de elección libre, sin embargo, la interfaz UART debe ser confiable, por lo tanto, se elegirá una velocidad de transmisión baja. Teniendo en cuenta lo anterior, se propone la siguiente configuración del UART:

- Baudrate: 9600.
- 8 bits de datos.
- Bit de paridad impar.
- Un bit de stop.

Tabla 11. Driver eUSCIA0_UART

eUSCIA0_UART	
Función	**Descripción.**
void USCIA0_UART_Init()	Inicializa UART en el módulo eUSCIA0 con la siguiente configuración: 9600 bps, 8 bits de datos, paridad impar, un bit de stop.
void eUSCIA0_UART_send(uint8_t data_Tx);	Envía un byte por UART. Ejemplo: Enviar 0x10 por UART: eUSCIA0_UART_send(0x10)
uint8_t eUSCIA0_UART_receive()	Recibe un byte, controlado el flujo de datos por medio de banderas de interrupción. uint8_t dato = eUSCIA0_UART_receive()

En la Tabla 11 se pueden observar las funciones para el driver eUSCIA0_UART, para la rutina de inicialización del periférico se ha decidido dejar una configuración debido a que esta interfaz solo se implementará para la comunicación con el dispositivo externo por lo que no es necesario cambiar la configuración del periférico

Finalmente, es importante mencionar que a pesar de que UART es un protocolo que implementa un mecanismo muy sencillo (bit de paridad) para determinar errores, la integridad de la comunicación será asegurada por las capas superiores de software, implementando algoritmos de detección de errores.

4.5.3.3 eUSCIB0_SPI

En el caso del driver eUSCIB0_SPI primero se realizó la configuración del módulo SPI teniendo en cuenta los requerimientos de la FRAM CY15B104Q. Las características más importantes del módulo SPI configurado en el microcontrolador supervisor se muestran en la Tabla 12

Tabla 12. Inicialización requerida del módulo SPI

Modulo implementado	eUSCIB0
Pines asignados:	• MOSI: P1.6 • MISO: P1.7 • CLK: P2.2 • CS: P1.2
Modo de SPI	SPI Modo 0: Se considera al estado bajo como estado inactivo del reloj y se realiza el muestreo en el flanco positivo.
Primer bit enviado	MSB.
Fuente de reloj	SM CLOCK
Velocidad de transmisión	1 Mbit/s
Otras observaciones	CS se controla por medio de un GPIO externo al módulo es eUSCIB0

En la Tabla 12 se puede observar un resumen de las configuraciones, se realizarán en la interfaz SPI del microcontrolador supervisor, la determinación de las especificaciones está dada por los requerimientos de la FRAM CY15B104Q (vea 3.7.2.1). De acuerdo con 3.7 se ha implementado este protocolo de comunicación a través del módulo eUSCIB0, como fuente de reloj se ha elegido a la señal SMCLOCK lo que permite la implementación de modos de bajo consumo (vea 3.7.1.7). Es importante hacer notar que CS ha sido implementado por medio de un GPIO externo al módulo eUSCIB0, a pesar de que el módulo tiene la opción de implementar este puerto internamente se ha optado por implementarlo independiente para tener un mejor control y poder escalar el proyecto a utilizar memorias FRAM redundantes.

Tabla 13. Driver eUSCIB0_SPI

eUSCIB0_SPI	
Función	**Descripción.**

void eUSCIB0_SPI_init()	Inicializa SPI en el módulo eUSCIB0 con la siguiente configuración: • MSB primero. • SPI de 3 líneas (SCK, MOSI, MISO). • SPI Mode 0 (UCCKPL = 0, UCCKKPH =0) • 8 bits de datos. • Modo maestro. • Habilita interrupciones de transmisión y recepción. • ACLOCK como fuente de reloj. • Configura la función alterna en puertos necesarios: • P1.6 (MOSI) • P1.7 (MISO) • P2.2 (SCK) • P1.3 (CS) Nota: CS no se configura dentro del módulo eUSCIB0 sino como un puerto digital de salida.

void eUSCIB0_SPI_writeByte(int dato);	Envía un byte por medio del bus SPI.
uint8_t eUSCIB0_SPI_readByte();	Recibe un byte por medio del bus SPI.
void eUSCIB0_CS1_set_state(uint8_t state);	Cambia el estado de la línea CS. Si *state* es 0 la línea CS se pone en estado bajo. Si *state* es 1 la línea CS se pone en estado alto.

Las funciones de software realizadas para la implementación de SPI en el microcontrolador supervisor se muestran en la Tabla 13. Debido a que la interfaz SPI será únicamente con la memoria FRAM se ha decidido que no exista posibilidad de cambiar la configuración de SPI a través de la función eUSCIB0_SPI_init(), por lo que la función no recibe ningún parámetro y solo se encarga de inicializar el módulo SPI con la configuración necesaria para establecer comunicación con la memoria FRAM.

Las funciones relativas a la recepción y envío de bytes a través de SPI controlan el flujo de datos por medio de las banderas que el módulo eUSCIB0 provee las cuales se ponen en alto cuando se está realizando una transferencia de datos ya sea en la línea MISO o MOSI, por lo que no se enviará ni recibirá ningún byte mientras las líneas se encuentran ocupadas.

4.5.3.4 Driver RTC.

También se implementará un driver para poder controlar el módulo RTC del microcontrolador MSP430FR59 y así, con fines de estudio, poder tener el dato de cuando suceden los SEE.

Tabla 14. Driver RTC

RTC	
Función	**Descripción.**
void RTC_disable();	Deshabilita el RTC para configurarlo o cambiar cualquiera de sus parámetros: hora, fecha, alarma o cualquier otro parámetro.
void RTC_setTime(int hour, int min)	Configura la hora en el RTC (los segundos no son configurables). La hora tiene que estar en formato de 24 hrs, y los valores se tienen que pasar como un tato hexadecimal. Ejemplo. Configurar la hora 1:43 pm en el RTC. 13:43 hrs 13 = 0xD 43 = 0x2A Finalmente. RTC_setTime(0x0D,0x2A)
void RTC_setDate(int day, int month, int year);	Configura la fecha en el RTC. Ejemplo. Configurar la fecha 16 de Enero del 2021 en el RTC. RTC_setDate(16,01,2021)
void RTC_enable();	Habilita el RTC, siempre se tiene que invocar después de configurarlo.

En la Tabla 14 se pueden observar las funciones realizadas para este módulo, como se puede ver las funciones realizadas tienen dos propósitos principales:

- Configurar la fecha y hora en el RTC.
- Habilitar y deshabilitar el RTC.

Los bloques superiores de software podrán usar está librería para cualquier función requerida, por ejemplo, tener datos acerca de cuando ocurren los SEE.

4.5.3.5 TimerA0.

El driver TimerA0 tiene será ocupado por bloques superiores de software para poder implementar el esquema de watchdog con el microcontrolador maestro, teniendo en cuenta esto, se deben determinarán las funciones referentes al timer.

Tabla 15. Driver TimerA0

TimerA0	
Función	**Descripción.**
void timer_Init(void);	Inicializa el TimerA0. • Selecciona ACLK como fuente de reloj. • Configura un divisor con un factor de 8 (SMCLK/8) • Configura para cuenta hacia arriba.
void timer_setPeriod(uint16_t sec)	Configura el periodo del timer, por ejemplo, para configurar el timer con un periodo de 1s, la función se tiene que invocar de la siguiente manera: timer_setPeriod(1)

void timer_Stop(void);	Para el timer. El timer deja de contar y reinicia el valor del timer a 0.
void timer_Enable(void);	Habilita el timer, el timer comienza a contar.

En la Tabla 15 se presentan las funciones desarrolladas para el timer, para la inicialización del timer se ha escogido al ACLK como fuente de reloj el cual puede tomar como fuente de reloj el oscilador de baja frecuencia de 32.7638 kHz

$$f_{timer} = f_{SMCLK} = 32.7638\, kHz$$

A continuación, se determinarán las limitaciones funcionales que esto implica. Al tener una señal de reloj de 125 KHz esto implica el periodo es el siguiente:

$$T_{timer} = \frac{1}{f_{timer}} = 30.52\, uS$$

Es importante mencionar que el periodo del reloj es equivalente al tiempo que tarda el timer en cambiar su valor de cuenta al siguiente valor. Por ejemplo, para contar del 0 al 10 al timer le tomará 305.2 uS.

Por otra parte, se tiene que el microcontrolador MSP430FR5969 tiene una longitud de palabra de 16 bits, lo cual repercute en el valor máximo de cuenta, el cual al ser un entero sin signo se puede calcular de la siguiente manera:

$$count_{max} = 2^{16} - 1 = 65\,535$$

Una vez teniendo el valor máximo de cuenta y el periodo del reloj que sirve como fuente de reloj al timer, se puede calcular el tiempo máximo que el timer puede alcanzar antes de que se desborde el valor de cuenta. A continuación, se presenta el cálculo:

$$t_{max} = (count_{max})(T_{SMCLK}) = 20\, S$$

Este dato tiene impacto directamente el esquema de watchdog, por lo tanto, es importante que se considere a la hora de desarrollar el software que supervisará al microcontrolador maestro.

En la Tabla 15 también podemos ver funciones para:

- Configurar el periodo del timer, esta función recibe como parámetro los segundos correspondientes al periodo del timer e internamente calcula el valor de cuenta necesario para obtener el periodo requerido.
- Parar el timer. Esta función permite parar la cuenta del timer y reiniciarla a cero.
- Habilitar el timer. Esta función permite habilitar el timer.

4.5.4 Drivers para los dispositivos.

En esta sección se presentará el diseño de los drivers para la FRAM y el bootlooader del microcontrolador STM32F407VG. Es importante mencionar que el objetivo de estos drivers es disminuir la complejidad del uso de estos elementos para el software de aplicación o capas superiores, por otra parte, estos drivers utilizaran los drivers bare metal, los cuales interactúan directamente con el microcontrolador MSP430FR5969.

4.5.4.1 FRAM_Commands

Las funciones del driver FRAM_Commands se han desarrollado con base en los diagramas de flujo vistos en 3.7.2 y utilizando las funciones del driver eUSCIB0_SPI explicado en 4.5.3

Tabla 16. Driver FRAM_Commands

FRAM_Commands	
Función	**Descripción.**
void FRAM_write(int ADDRESS_1, int ADDRESS_2,	Escribe en la FRAM el arreglo de tamaño *arrayTxSize* apuntado por **arrayTx* a partir de la dirección compuesta por *ADDRESS_1, ADDRESS_2 y ADDRESS_3.*

int ADDRESS_3, int* arrayTx, int arrayTxSize)	
void FRAM_read(int ADDRESS_1, int ADDRESS_2, int ADDRESS_3, uint16_t* arrayRx, int arrayRxSize)	Lee en la FRAM el arreglo de tamaño *arrayRxSize* partir de la dirección compuesta por *ADDRESS_1, ADDRESS_2 y ADDRESS_3 y* lo guarda en el arreglo apuntado por **arrayRx.*

En la Tabla 16 se pueden apreciar las funciones de escritura y lectura desarrolladas, el diseño de las funciones se fundamentó en los requerimientos de la FRAM para ejecutar operaciones de lectura y escritura los cuales se han presentado en 3.7.2.3 y 3.7.2.4. Es importante mencionar que de acuerdo con 3.7.2 el direccionamiento de memoria es por medio 19 bits por lo que se requieren 3 bytes para poder apuntar a una dirección de memoria, sin embargo, de acuerdo con 3.7.1.1 el microprocesador del MSP430FR5969 solo es de 16 bits por lo que existió la necesidad de partir la dirección de 3 bytes en bloques de un byte cada uno (*ADDRESS_1, ADDRESS_2 y ADDRESS_3*) empezando por el MSB.

Por otra parte, como se ha visto en 3.6.4 las FRAM, a diferencia de las Flash, tienen la ventaja de que soportan la sobreescritura de datos por lo que no es necesario realizar una operación de borrado de memoria por lo que no se ha implementado una función con este fin.

Más adelante las funciones relacionadas con la interfaz de la FRAM serán ocupadas por el software de aplicación para desarrollar un algoritmo que permita guardar una copia de respaldo del microcontrolador principal en la FRAM.

4.5.4.2 STMF407xx_bootloaderCommands

Una vez que se ha desarrollado el software referente a la configuración básica y a la transmisión y recepción de datos por medio de UART. Se han desarrollado las funciones referentes a la ejecución de comandos en el bootloader del microcontrolador maestro.

Tabla 17. Driver STMF407xx_bootloaderCommands. Funciones que interactúan con otros bloques de software.

STMF407xx_bootloaderCommands. Funciones principales.	
Función	**Descripción.**
int BootloaderAccess(void);	Ejecuta la secuencia en los GPIO para acceder al bootloader del microcontrolador maestro. • Si el acceso fue correcto devuelve un byte de ACK. • Si el acceso fue incorrecto, devuelve un byte de NACK.
void readMemoryCommand(int ADDRESS_MSB, int ADDRESS_LSB, uint8_t* arrayRx, int arrayRxSize)	Lee *arrayRxSize* bytes a partir de la dirección de memoria definida por *ADRESS_MSB y ADRESS_LSB,* los datos obtenidos son guardados en el vector apuntado por *arrayRx Ejemplo. Leer n bytes a partir de la dirección de memoria 0x08000060: readMemoryCommand(0x0800,0x0060,4)
void writeMemoryCommand(int ADDRESS_MSB, int ADDRESS_LSB, uint32_t* arrayTx, int arrayTxSize)	Escribe el vector apuntado por *arrayTx de longitud *arrayTxSize* bytes a partir de la dirección de memoria dada por ADRESS_MSB y ADRESS_LSB Ejemplo. Escribir 4 bytes a partir de la localidad de memoria 0x0800004.

	writeMemoryCommand(0x0800,0x0004,4)
void goCommand(int ADDRESS_MSB, int ADDRESS_LSB)	Sale del bootloader y lleva al contador de programa PC a cierta dirección. Ejemplo. Salir del bootloader y ejecutar el programa cargado a partir de la localidad de memoria 0x08001000. goCommand(0x0800,0x1000)
void eeraseCommand(int FlashSectorCode);	Borra cierto sector de la memoria flash. Ejemplo. Borrar el sector 6 de memoria flash: eeraseCommand(0x06)

En la Tabla 17 se pueden observar las funciones desarrolladas para la ejecución de comandos en el bootloader del microcontrolador principal, estas funciones han sido programadas teniendo en cuenta las especificaciones para la ejecución de comandos, presentadas en 3.8

Es importante mencionar que las funciones para ejecutar comandos de escritura y lectura utilizan apuntadores, esto se ha decidido hacer así para que el programa tenga una gran versatilidad y se puedan leer o escribir vectores de cualquier dimensión en la memoria Flash del microcontrolador maestro.

Las direcciones de la memoria Flash del microcontrolador maestro están compuestas de cuatro bytes, sin embargo, de acuerdo con 3.7.1.1 el microprocesador del supervisor es de 16 bits por lo que se ha dividido la dirección de memoria en bloques de dos bytes cada uno (ADRESS_MSB y ADRESS_ LSB).

Por otra parte, es importante mencionar que la Tabla 17 solo muestra las funciones más importantes tomando como criterio su utilidad para realizar las funciones de monitoreo detección y corrección de fallas en el microcontrolador supervisor. Sin embargo, hay más funciones que permiten hacer modular el código y evitar tener

dos bloques de código haciendo la misma función las cuales se presentan en la Tabla 18.

Tabla 18. Driver STMF407xx_bootloaderCommands. Funciones internas

STMF407xx_bootloaderCommands. Funciones secundarias.	
sendCommand(int command)	Esta función es auxiliar, en principio nunca será usada por el usuario. Esta función envía el comando a ejecutar en el bootloader del microcontrolador principal. Si el comando se ejecuta correctamente en el microcontrolador principal se recibe un bit de ACK. Ejemplo: Ejecutar el comando *WRITE MEMORY* en el bootloader del microcontrolador principal. Se tiene que el código de comando asociado es 0x31 sendCommand(0x31)
static void writeData (uint32_t* arrayTx2, int arrayTxSize2);	Función auxiliar ocupada por la función writeMemoryCommand(), esta función es la que se encarga de mandar por uart los datos que seráni escritos en la memoria.
static void send_4bytes_wChecksum(int WORD_MSB, int WORD_LSB)	Esta función envía 4 bytes consecutivos con su respectivo checksum.
static void send_startAddress(int ADDRESS_MSB int ADDRESS_LSB)	Esta función es auxiliar, en principio nunca será usada por el usuario. Esta función es ocupada por las funciones goCommand(), writeMemoryCommand() y readMemoryCommand() y sirve para enviar

	la dirección en la que se realizará la operación correspondiente

static void receiveCommand_dataRx(uint32_t* arrayRx2, int arrayRxSize2)	Esta función es auxiliar, en principio nunca será usada por el usuario. Esta función recibe los datos que responde el microcontrolador principal después de que ha ejecutado un comando, esta función tiene flexibilidad respecto al número de datos que recibirá. Puede recibir desde 0 hasta *numData datos.*
	
	Ejemplo de utilización.
	
	sendCommand(0x00) //Envía comando get
	receiveCommand_dataRx() //Recibe los datos respondidos del comando GET.

Las funciones de la Tabla 18 se han desarrollado con el fin de ser utilizadas por las funciones principales del driver. Si se analiza lo presentado en 3.8.3, 3.8.4, 3.8.5 y 3.8.6 la ejecución de comandos de escritura, lectura y borrado de memoria tienen que realizar algunos pasos en común, para estos pasos en común se han creado las funciones de la Tabla 18

Tabla 19. Relaciones entre las funciones primarias y secundarias del driver STM32_BootloaderCommands

Función secundaria	Funciones primarias que la utilizan.
sendCommand();	1. writeMemoryCommand 2. readMemoryCommand 3. goCommand 4. eeraseCommand
send_4bytes_wChecksum();	1. writeMemoryCommand

send_startAddress();	1. writeMemoryCommand 2. readMemoryCommand 3. goCommand
writeData()	1. writeMemoryCommand
receiveCommand_dataRx()	2. readMemoryCommand

En la Tabla 19 se muestran las relaciones entre la funciones primarias y secundarias.

4.5.5 Aplicación

Una vez que en 4.5.3 y 4.5.4 se han presentado los drivers del microcontrolador y de los dispositivos, respectivamente, se tienen los bloques de software necesarios para construir la aplicación requerida por la misión. En esta sección de la tesis primero se presentará el diseño de la aplicación, y posteriormente como este diseño será llevado a la realidad utilizado los bloques de software diseñados anteriormente. Como se ha visto en 4.5.1 los requerimientos funcionales son los siguientes:

1) Reprogramar al microcontrolador maestro en caso fallas en su funcionamiento debidas a radiación.

2) Determinar el estado funcional del microcontrolador maestro.

3) Almacenar una copia del programa del microcontrolador maestro en la memoria FRAM.

4) Actualizar el código del microcontrolador maestro por medio de un dispositivo externo.

Para términos de la estructuración de este capítulo, se juntarán los requerimientos 1) y 2) así como los requerimientos 2) y 4), la razón de hacer esto es que estos

requerimientos se encuentran relacionados entre sí por lo que la aplicación atenderá dos grandes funcionalidades principales:

1) Determinación del estado funcional del microcontrolador maestro y su reprogramación en caso de fallas.

2) Actualización del programa del microcontrolador maestro por medio de un dispositivo externo, así como el almacenamiento del nuevo programa en la memoria FRAM.

Una vez aclarado este punto, la estructuración de esta sección será la siguiente.

1) Diseño de la aplicación.

a) Determinación del estado funcional del microcontrolador maestro y su reprogramación en caso de fallas.

b) Actualización del programa del microcontrolador maestro por medio de un dispositivo externo, así como el almacenamiento del nuevo programa en la memoria FRAM.

2) Implementación de la aplicación.

a) Determinación del estado funcional del microcontrolador maestro y su reprogramación en caso de fallas.

b) Actualización del programa del microcontrolador maestro por medio de un dispositivo externo, así como el almacenamiento del nuevo programa en la memoria FRAM.

4.5.5.1 Diseño de la aplicación: Determinación del estado funcional del microcontrolador maestro y su reprogramación en caso de fallas.

De acuerdo con el estado del arte, en 2.1.1 y 2.1.4 se han presentado esquemas de supervisión basados en watchdog, en este proyecto se diseñará un esquema similar, sin embargo, se aumentará la seguridad del monitoreo permitiendo reprogramar al microcontrolador maestro ante cualquier cambio en cualquier bit de la sección de código en el flash.

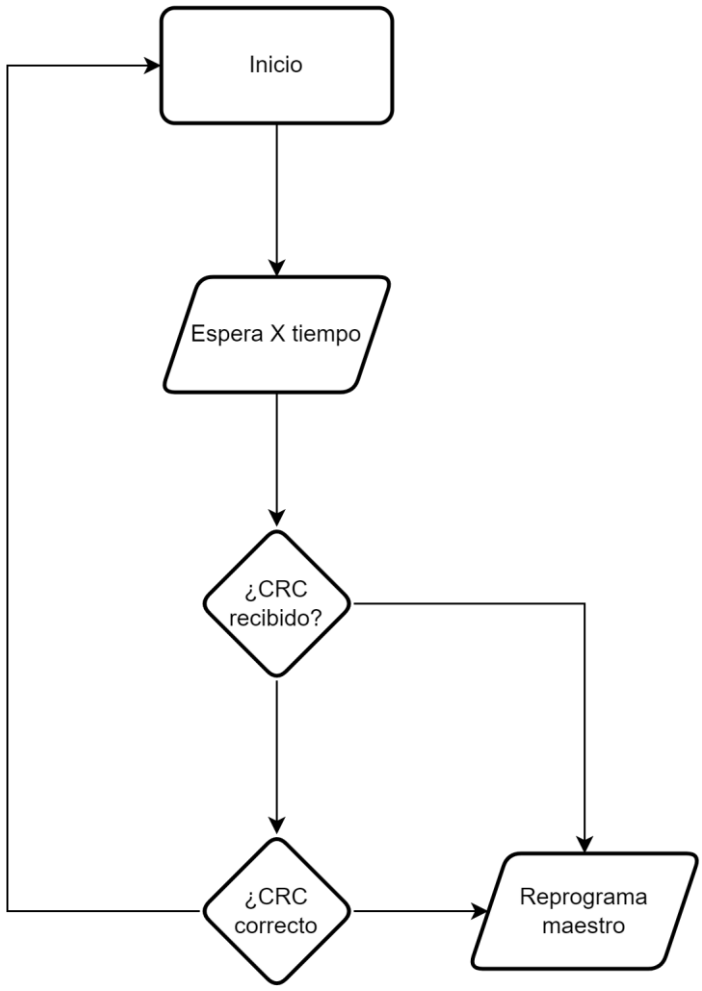

Figura 59. Determinación del estado funcional del maestro - microcontrolador supervisor

En la Figura 59 se puede observar el esquema de supervisión propuesto, como se puede ver consiste en un *watchdog* pero además de esto la señal de alivio será el

CRC de la memoria flash del microcontrolador maestro, permitiendo así, reprogramarlo ante una falla total o una falla parcial como el cambio de estado en algún bit de la memoria flash. También en el diagrama de flujo se puede apreciar que se esperará X tiempo, el cual será determinado por la probabilidad de ocurrencia de un SEE en la región sensible del microcontrolador, lo cual se ha determinado en 4.3.4. Una vez que se ha esperado X tiempo, se verifica que el CRC haya llegado, en caso contrario se reprogramará al microcontrolador maestro. Si el CRC ha llegado, se verifica que coincida con el CRC del programa inicial, en caso contrario se reprogramará al microcontrolador maestro. En resumen, el esquema de supervisión verifica dos cosas, que el CRC de la región de código del flash haya llegado a tiempo y que sea correcto, si cualquiera de estas condiciones no se cumple se reprogramará al microcontrolador maestro. La función de reprogramación del microcontrolador maestro se explica más adelante en la sección 4.5.5.4.2 y se puede observar en Figura 62

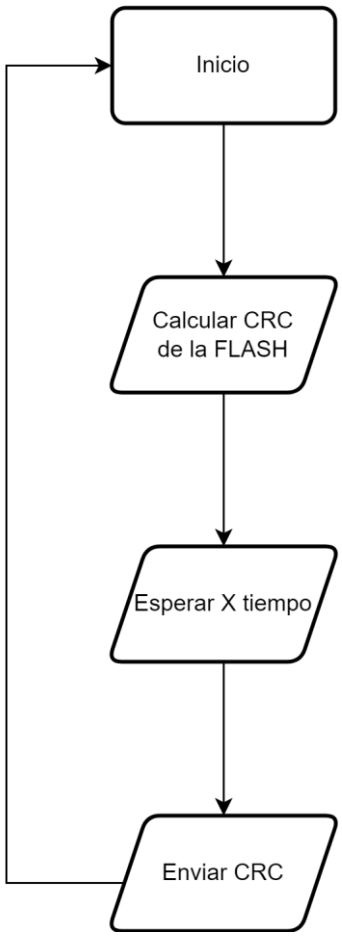

Figura 60. Determinación del estado funcional del maestro - microcontrolador maestro

Como se puede indagar, el esquema de supervisión requiere también la participación del microcontrolador maestro. En la Figura 60 se presenta el esquema de supervisión desde la perspectiva del microcontrolador maestro el cual estará calculando su CRC de manera periódica y lo enviará hacia el supervisor. Si se presenta una falla total el maestro no enviará el CRC hacia el supervisor, por lo tanto, el maestro será reprogramado, si se presenta una falla parcial en la que algún bit que cambie de estado en la región de memoria flash correspondiente al código será detectado en el cálculo del CRC, ya que como se ha visto en 3.5.1, el CRC cambia ante cualquier cambio en los datos que el algoritmo recibe como entrada, por lo tanto, al cambiar el estado de un bit, también cambiará el CRC calculado, por lo tanto no coincidirá con el CRC esperado, por lo tanto, el microcontrolador maestro será reprogramado.

Es importante mencionar que a pesar de que el esquema de supervisión también se estará ejecutando en el microcontrolador maestro, se buscará minimizar el tiempo que el microprocesador gasta en esta tarea, utilizando el periférico incluido en el microcontrolador para el cálculo del CRC y además también se utilizará el DMA, para así quitar la mayor carga de trabajo posible y no se interfiera con la tarea principal de la misión.

A continuación, se hará un breve análisis para determinar el impacto que tendrá el esquema de supervisión en el tiempo de procesamiento del microcontrolador maestro:

El tamaño reservado para el código en la memoria flash es el siguiente:

$$Code_{size} = 1\ MB$$

El CRC procesa palabras de 32 bits o, lo que es lo mismo, 4 bytes. El número de ciclos de procesamiento del módulo CRC requerido para calcular el CRC de todo el flash corresponde al siguiente calculo:

$$Num\_crc\_cycles = \frac{Code_{size}}{4\ Bytes} = \frac{1\,000\,000\ Bytes}{4\ Bytes} = 250\,000$$

Una vez que se ha determinado el número de ciclos necesarios en el periférico para calcular el CRC de todo el flash, se puede determinar el impacto que esto tendrá en cuestiones de tiempo. De acuerdo con [34] se tiene que para un dato 256 palabras de 32 bits, al módulo CRC le ha tomado 1287 ciclos de reloj realizar el cálculo.

Tabla 20. Comparación de los ciclos de reloj invertidos para el cálculo del CRC utilizando un algoritmo y el periférico

Optimization level	CRC algorithm (system clock cycle)	CRC peripheral (system clock cycle)
Level 3 + Optimize for time	78094	1287

A continuación, se calculará el tiempo de procesamiento por palabra:

$$T\_procesamiento\_palabra = \frac{1287\ clock\ cycles}{256\ words} = 5.02\ clock\ cycles\ /\ palabra$$

Añadiendo un factor de seguridad se tiene que:

$$ciclos\_procesamiento\_palabra = \ 6\ clock\ cycles\ /\ palabra$$

A continuación, se calcularán los ciclos de reloj necesarios para determinar el CRC de todo el flash correspondiente a la sección de código.

$$ciclos\_procesamiento\_flash = \ 250\,000\ *\ 6\ clock\ cycles\ /\ palabra$$
$$= \ 1\,500\,000\ cycles\ /\ flash$$

Asumiendo un reloj de 16 MHz

$$t\_ciclo\ = \frac{1\ s}{16\ M}\ =\ 62.5\ nS$$

$$tiempo\_procesamiento\_flash = \ 1\,500\,000\ (62.5\ nS)\ =\ 93.75\ mS$$

El cálculo anterior presenta el tiempo que le toma al periférico calcular el CRC de toda la región de programa del flash, en este caso se asume que los datos se están cargando al periférico utilizando el microprocesador, sin embargo, para disminuir aún más la carga de trabajo, se utilizará el DMA para cargar los datos, liberando así, al microprocesador de esta tarea. De acuerdo con [34], cuando el DMA se utiliza para cargar los datos hacia el periférico del CRC, la carga de trabajo del CPU disminuye de un 100% hasta un 0.72%, por lo tanto, si para esta aplicación se utiliza el DMA para mover los datos del flash hacia el periférico del CRC, el tiempo que el CPU empleará en esta tarea se puede calcular como sigue.

$$tiempo\_procesamiento\_flas\_con\_DMA = \ 1\,500\,000\ (62.5\ nS)(0.0072)\ =\ 675\ uS$$

Como se puede apreciar el tiempo que el CPU utiliza para el cálculo del CRC del flash, cuando se utiliza el DMA, resulta muy pequeño y prácticamente despreciable.

4.5.5.2 Diseño de la aplicación: Actualización del programa del microcontrolador maestro por medio de un dispositivo externo, así como el almacenamiento del nuevo programa en la memoria FRAM

La actualización del código del microcontrolador maestro seguirá la siguiente secuencia: Recepción del nuevo programa del microcontrolador maestro a través de un dispositivo externo y su almacenamiento en FRAM. Una vez que la copia del programa se encuentra en FRAM, el microcontrolador supervisor accederá al bootloader del microcontrolador maestro, y actualizará el programa utilizando la copia previamente almacenada en FRAM. En resumen: se ha dividido el diseño de esta función en dos partes principales.

1. Recepción y almacenamiento en FRAM de la nueva versión de programa del microcontrolador maestro.

2. Actualización del programa del microcontrolador maestro utilizando la copia de respaldo previamente guardada en FRAM.

En la Figura 61 se puede observar la primera parte de la funcionalidad, la cual consiste en la recepción y almacenamiento en FRAM de la nueva versión del programa del microcontrolador maestro. Como se puede ver el programa se ha dividido en secciones, esto es debido a que el programa completo del maestro no cabría en la memoria del microcontrolador supervisor, por lo tanto, se ha decidido seccionar el programa del microcontrolador maestro, e ir manipulando estas secciones del programa por medio de un buffer. En resumen, el funcionamiento expresado en la Figura 61 es el siguiente:

- El microcontrolador supervisor recibe un segmento de N bytes del programa del microcontrolador maestro.
- El microcontrolador supervisor comprueba la integridad de los datos recibidos.

- Si los datos recibidos se encuentran en buen estado, el microcontrolador supervisor los escribe en la FRAM. Se comprueba que el segmento recibido sea el segmento final, en caso afirmativo se finaliza la escritura de los datos, en caso contrario se solicita el siguiente segmento del programa.
- Si los datos recibidos han sido corrompidos durante la transmisión, el microcontrolador maestro solicita el reenvío de los datos.

La Figura 61 muestra la recepción y almacenamiento de la nueva versión del programa del microcontrolador maestro, pero solo se muestra a nivel funcional. Más adelante se tomarán en cuenta los detalles técnicos para poder lograr esta funcionalidad.

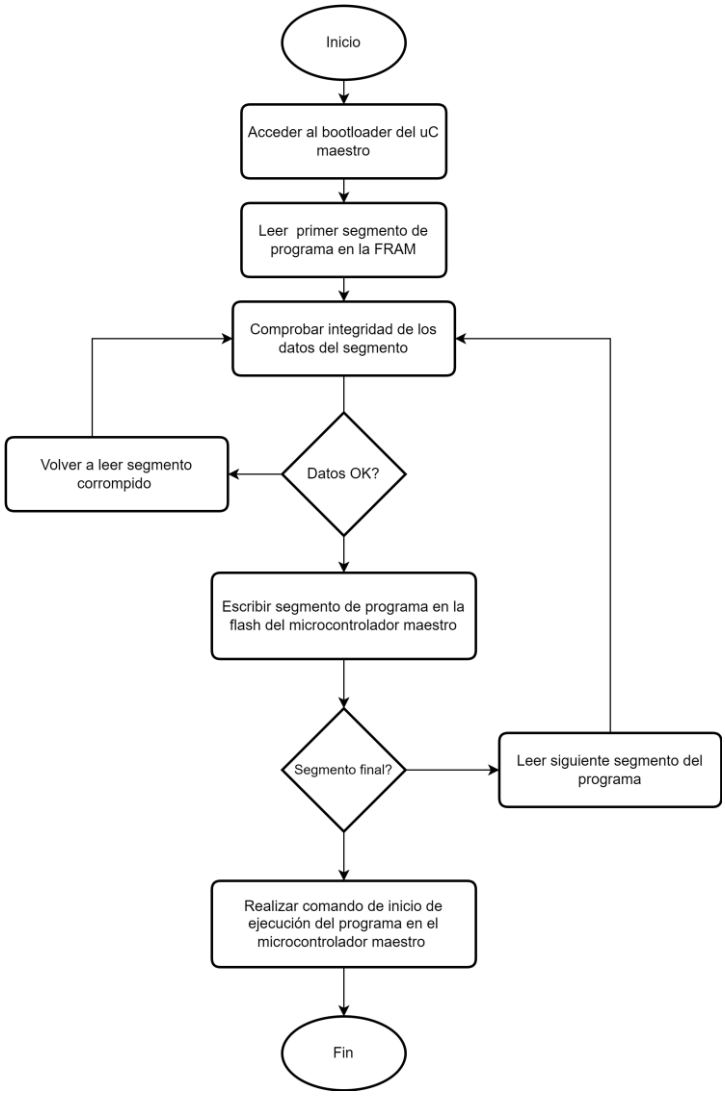

Figura 61. Actualización del respaldo en FRAM del programa del microcontrolador maestro

En la Figura 62 se puede observar la segunda parte de la funcionalidad, la cual consiste en la reprogramación del microcontrolador maestro utilizando la copia del programa que se encuentra en la FRAM. La secuencia para llevar a cabo esta función es la siguiente:

- El microcontrolador supervisor accede al bootloader del microcontrolador maestro.

- El microcontrolador supervisor toma de la memoria FRAM el primer segmento del programa del microcontrolador maestro y comprueba su integridad.
- Si el segmento leído se encuentra en buen estado, será escrito en la memoria flash del microcontrolador maestro. Después, se comprobará si se trata del último segmento del programa, en caso afirmativo, se ejecutará el comando de ejecución de programa en el microcontrolador maestro y se terminará con la ejecución de la funcionalidad, en caso contrario, se leerá el siguiente segmento correspondiente a la copia en FRAM del programa del microcontrolador maestro.
- Si el segmento leído se encuentra en mal estado, se volverá a leer.

La Figura 62 muestra la actualización del programa del microcontrolador maestro, pero solo se muestra a nivel funcional. Más adelante se tomarán en cuenta los detalles técnicos para poder lograr esta funcionalidad.

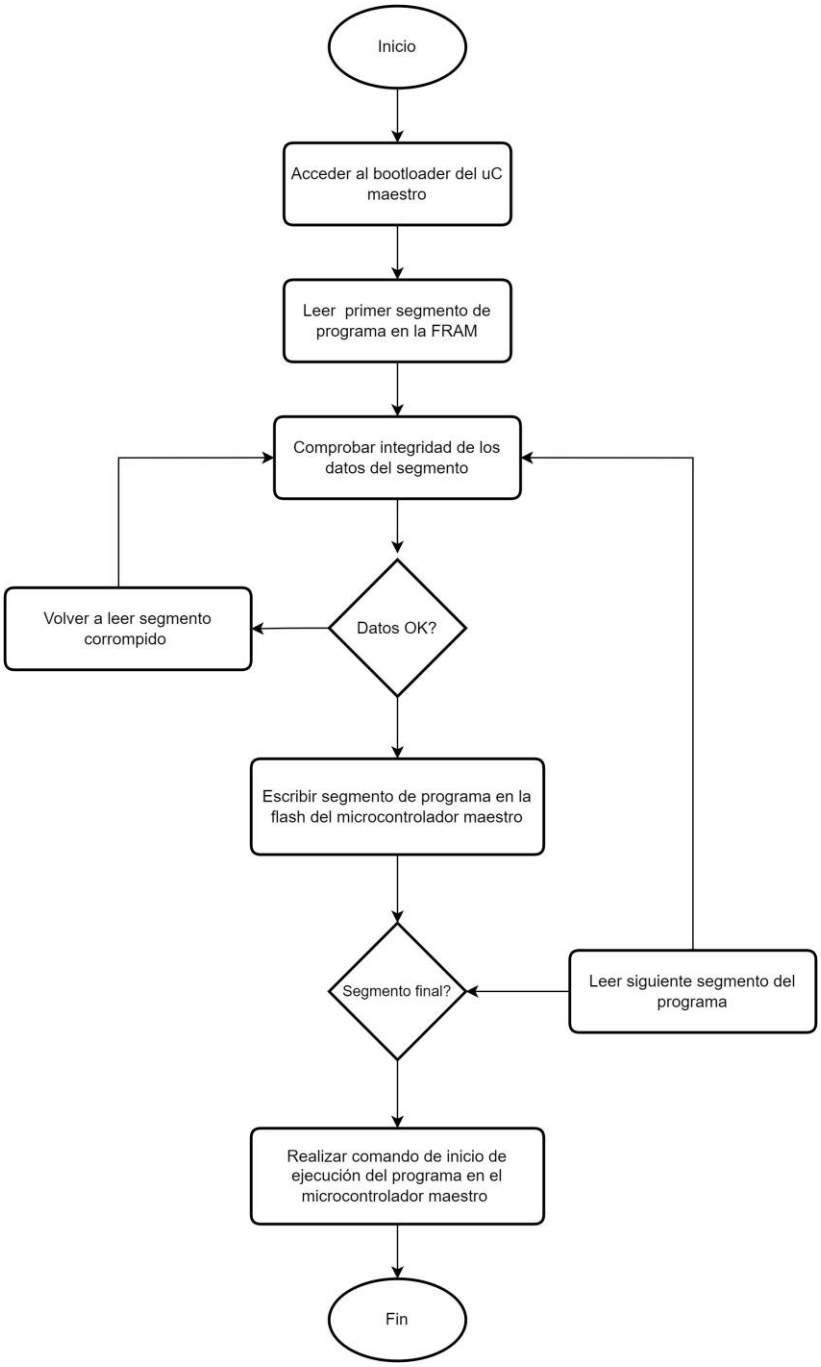

Figura 62. Reprogramación del uC maestro

4.5.5.3 Implementación de la aplicación: Determinación del estado funcional del microcontrolador maestro y su reprogramación en caso de fallas.

Una vez que en 4.5.5.1 se ha definido el funcionamiento requerido en la aplicación, en esta sección se llevará a la práctica lo requerido, utilizando los drivers diseñados en 4.5.3 y 4.5.4. El diseño propuesto en 4.5.5.1 es suficiente para entender la función, sin embargo, resulta demasiado general para llevarlo a la implementación, por lo tanto, es necesario profundizar un poco más en algunos aspectos.

El esquema elegido para el sistema de detección y corrección de fallas está basado en un *watchdog,* esta funcionalidad se puede dividir en dos partes:

- Del lado del microcontrolador maestro: Deberá calcular su CRC de manera periódica, el CRC servirá como indicador para disparar la señal de alivio para el *watchdog,* por lo que el CRC deberá ser enviado por medio de UART cada X tiempo.
- Del lado del microcontrolador supervisor: deberá implementar el *watchdog,* por lo que cada X tiempo se recibirá la señal de alivio la cual dependerá del CRC de la memoria flash del microcontrolador maestro, comprobará que el CRC llegué a tiempo y que sea correcto, si cualquiera de estas dos condiciones no se cumple, el microcontrolador supervisor reprogramará al microcontrolador maestro.

4.5.5.3.1 Rutina de sincronización del esquema de supervisión.

Debido a que cada dispositivo tiene sus propias fuentes de reloj es necesario realizar una rutina de sincronización para poder inicializar el esquema de supervisión. En la Figura 63 se muestra la rutina de inicialización y sincronización del lado del supervisor, mientras que en la Figura 64 se muestra la misma rutina, pero del lado del maestro. Es importante mencionar que el tamaño del programa del

maestro en bytes es necesario para la determinación del CRC, por lo tanto, se le hará llegar este dato al maestro por medio de la rutina.

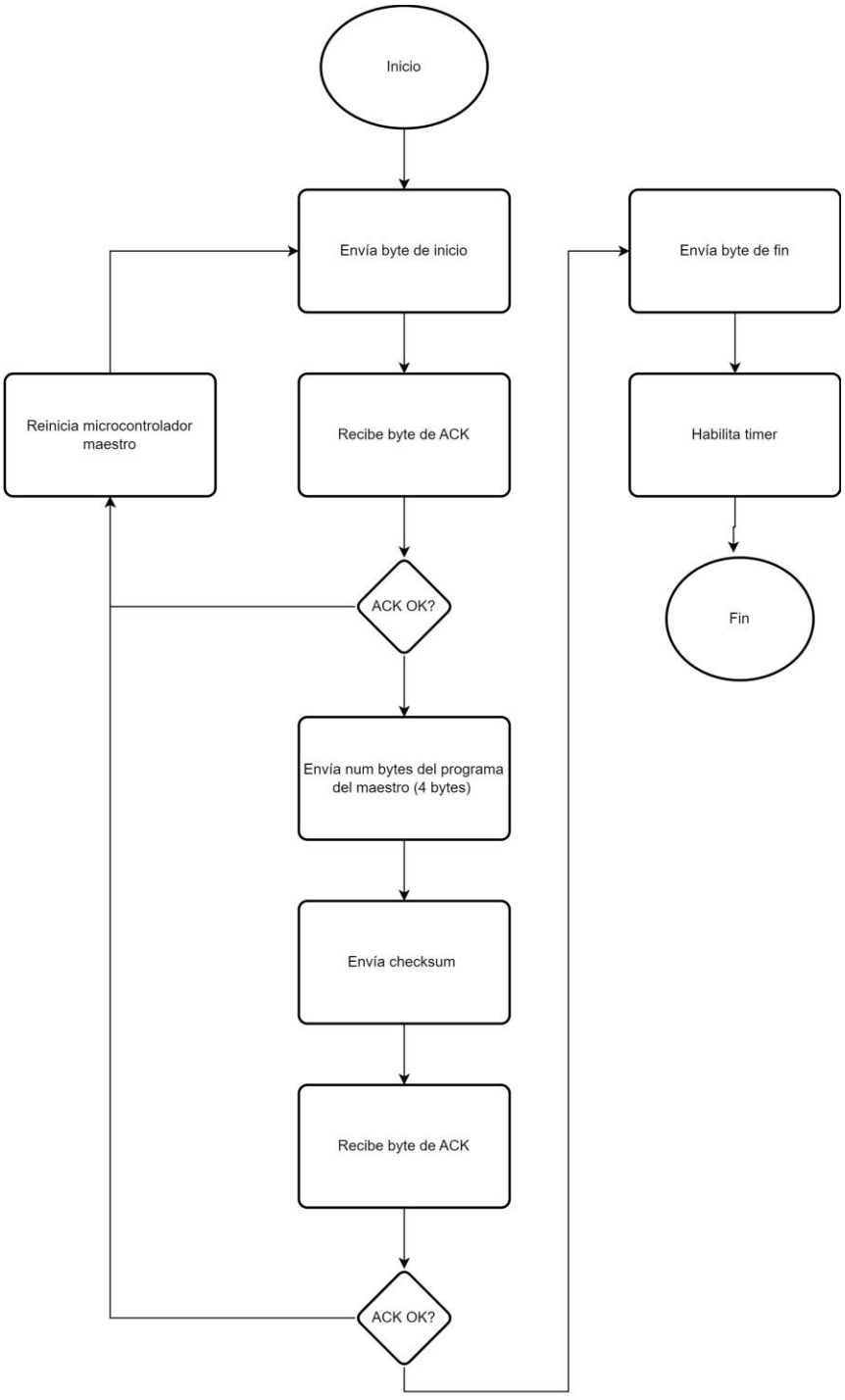

Figura 63. Rutina de inicialización y sincronización del esquema de supervisión - Lado del supervisor

En la Figura 63 se muestra la rutina de inicialización y sincronización del lado del supervisor, a continuación, se explica el diagrama de flujo:

- El supervisor envía un byte de inicio para indicar al maestro que se inicializará la rutina de sincronización.
- El supervisor evalúa la respuesta por parte del microcontrolador maestro. Si se trata de un byte de ACK se procede con la rutina, si se trata de un byte de NACK se volverá a enviar el byte de inicio.
- El supervisor enviará el tamaño del programa del maestro, el cual se trata de un dato de 4 bytes, además de esto enviará el checksum.
- El supervisor evalúa la respuesta por parte del microcontrolador maestro. Si se trata de un byte de ACK se procederá con la rutina, si se trata de un byte de NACK se volverá a enviar el byte de inicio.
- El supervisor envía un byte de fin para indicar al maestro que se finalizará la rutina de sincronización.
- El supervisor habilito el timer que está encargado del *watchdog,* por lo tanto, el esquema de supervisión comienza.

En la Figura 64 se muestra la rutina de inicialización y sincronización del lado del maestro, a continuación, se explica el diagrama de flujo:

- El maestro espera a recibir el byte de inicio
- El maestro comprueba que el byte recibido sea correcto, en caso afirmativo, envía byte de ACK y espera a recibir el dato correspondiente al tamaño del programa con el respectivo checksum, en caso contrario, envía byte de NACK y vuelve al estado anterior: esperar el byte de inicio.
- El maestro recibe el dato correspondiente al tamaño del programa (4 bytes), así como el checksum.
- El maestro calcula el checksum de los datos recibidos.
- El maestro comprueba que el checksum recibido sea igual al checksum calculado, en caso afirmativo, envía byte de ACK y procede a esperar el byte

de terminación, en caso contrario, envía byte de NACK y vuelve al estado inicial: esperar byte de inicio.

- Una vez que el maestro recibe el byte de fin, inicia el timer correspondiente al esquema de watchdog, inicializando así el esquema de supervisión

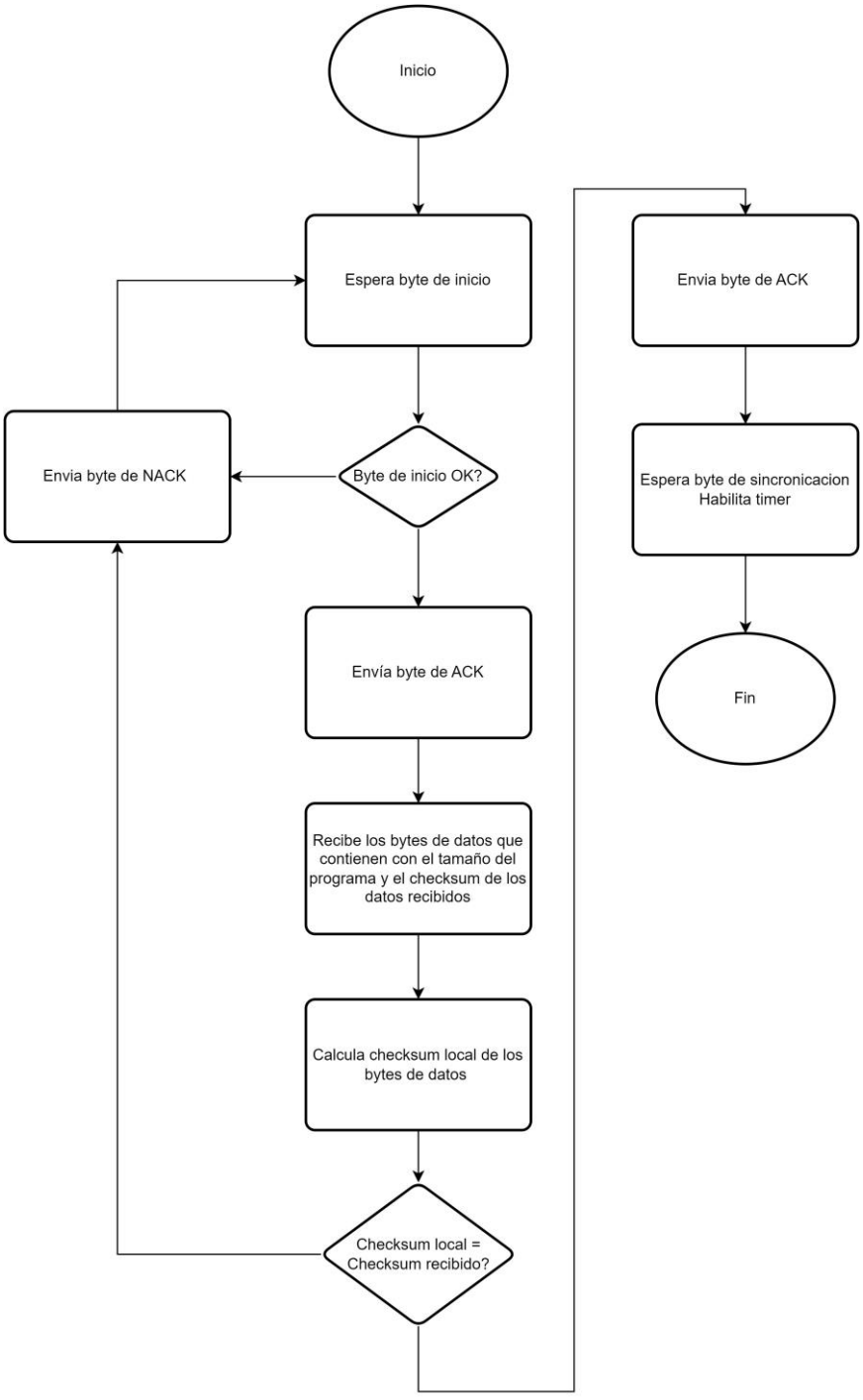

Figura 64. Rutina de inicialización y sincronización del esquema de supervisión - Lado del maestro

4.5.5.3.2 Implementación del esquema de supervisión.

Una vez que se ha diseñado la rutina de inicialización y sincronización del esquema de supervisión, se puede profundizar en el diseño del esquema. Para poder diseñar el esquema de supervisión se tendrán que implementar interrupciones tanto del lado

del maestro como del lado del supervisor, esto es debido a que el esquema de supervisión es, en esencia, un watchdog por lo que los requerimientos de tiempo resultan demasiado estrictos como para implementar esta funcionalidad con código secuencial.

Tabla 21. Interrupciones para el esquema de supervisión - Microcontrolador supervisor

Interrupción	Evento funcional que la ocasiona	Tareas por desarrollar en la subrutina de interrupción
Recepción de datos en el puerto UART	Cualquiera de los bytes correspondientes al CRC del flash del microcontrolador maestro ha llegado al puerto UART.	• Reiniciar al timer, configurando su valor de cuenta en 0. Recuerde que la señal de alivio para el watchdog es la llegada del CRC del flash del maestro. • Verificar que el byte correspondiente al CRC sea correcto. En caso de que el CRC no coincida se levantará una bandera global indicando que la memoria flash del microcontrolador maestro ha sido corrupta.
Valor final en la cuenta del timer	Los bytes correspondientes al CRC del flash del microcontrolador maestro no han llegado en el tiempo esperado	• habilitar una bandera global indicando que la memoria flash del del microcontrolador maestro ha sido corrupta.

En la Tabla 21 se muestran las interrupciones necesarias en el supervisor para lograr la funcionalidad del esquema de supervisión. De manera general se busca que las interrupciones cumplan la siguiente funcionalidad:

- La llega de un byte en el puerto UART se asume como la llegada de la señal de alivio, además de que el byte haya llegado, la subrutina de interrupción tiene que verificar que sea lo esperado, en caso contrario, se levanta una bandera global para indicar que la memoria flash se encuentra corrupta.
- Si se ejecuta la subrutina de interrupción del timer es porque la señal de alivio no ha llegado en el tiempo adecuado, por lo tanto, se levanta una bandera global para indicar que la memoria flash se encuentra corrupta.

Las subrutinas de interrupción únicamente levantan una bandera para indicar que la memoria flash ha sido corrupta, está bandera será utilizada como activación para el módulo de software que se encarga de reprogramar al microcontrolador maestro, el cual será explicado más adelante en esta sección.

Tabla 22. Interrupciones para el esquema de supervisión - Microcontrolador maestro

	Evento funcional que la ocasiona	Tareas por desarrollar en la subrutina de interrupción
Valor final en la cuenta del Timer	Ha llegado el momento de enviar la señal de alivio hacia el microcontrolador supervisor.	• Deberá disparar un nuevo cálculo del CRC de su flash. • Deberá enviar los bytes correspondientes al CRC de su flash.

En la Tabla 22 se observa la interrupción que tiene que ser implementada del lado del maestro para lograr el esquema de supervisión. Cada que esta subrutina de interrupción se ejecuta, se envía la señal de alivio (CRC actual del flash) hacia el microcontrolador maestro y además se dispara un nuevo cálculo del CRC.

Con el fin de explicar la funcionalidad requerida en las interrupciones, algunos detalles de la implementación se han omitido en la Tabla 21 y en la Tabla 22, por

ejemplo, que el CRC calculado por el periférico incluido en el microcontrolador maestro es de 32 bits, por lo que la subrutina de interrupción de recepción en el puerto UART tiene que ser capaz de manejar este tamaño de dato, una vez que de manera general se ha entendido el funcionamiento de las subrutinas de interrupción se explicará en detalle el algoritmo implementado por cada una de las subrutinas.

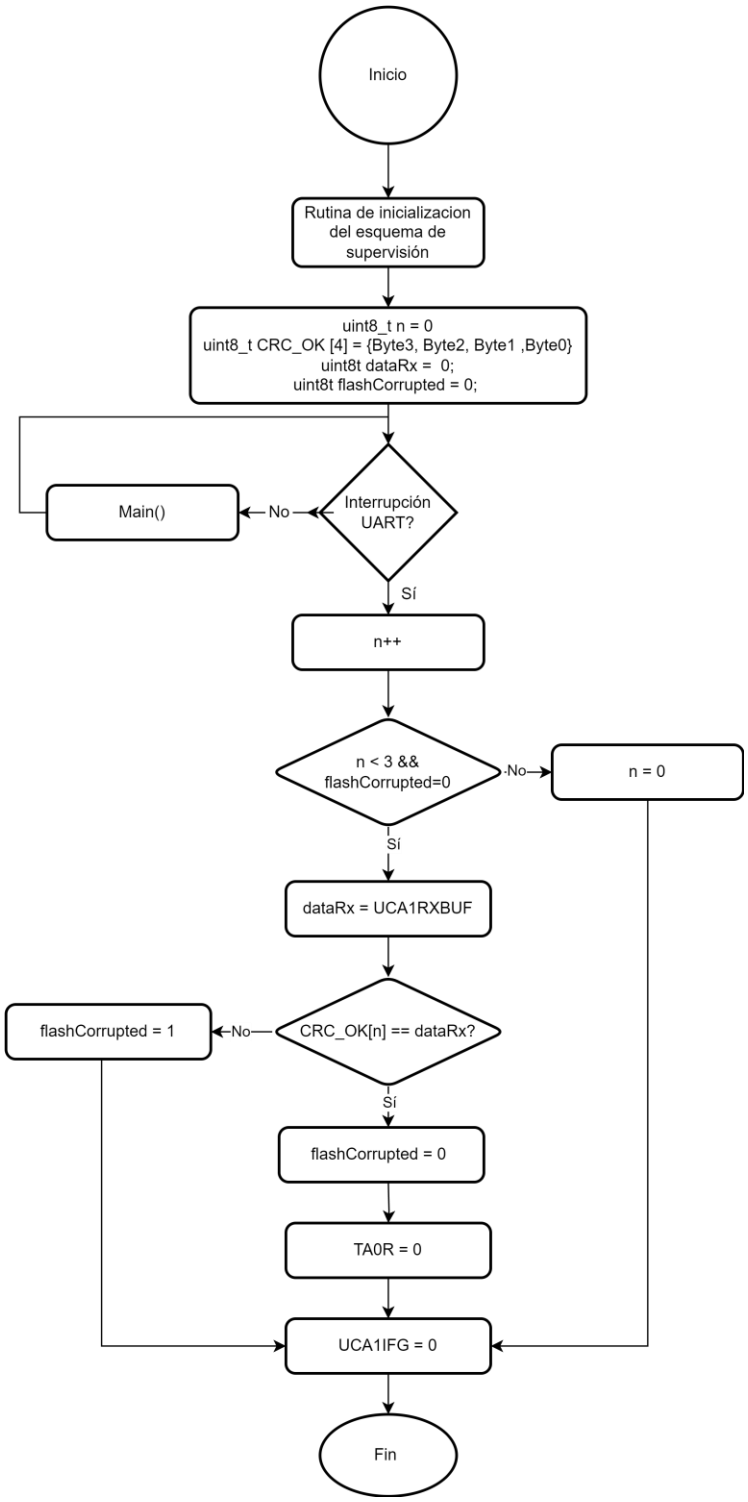

Figura 65. Supervisor - subrutina de interrupción por recepción de dato en el puerto uart0

En la Figura 65 se muestra la subrutina de interrupción debida a la recepción de un byte en el puerto UARTA0 del microcontrolador maestro. Como condiciones previas

al evento que ocasiona la interrupción se tiene que haber ejecutado la rutina de inicialización del esquema de supervisión, además de esto es necesario el uso de las siguientes variables:

- CRC_OK[4]: Arreglo de 4 bytes que contiene el CRC de referencia recibido por medio de la rutina de inicialización. El CRC de la memoria flash es de 32 bits y además solo se puede recibir un byte a la vez utilizando el protocolo UART, por lo tanto, es necesario dividir el CRC en cuatro partes, cada una de un byte, cada elemento del vector corresponde a una parte del CRC. Este vector contiene el CRC inicial de la memoria flash del microcontrolador maestro.
- n: Variable que sirve para llevar la cuenta de las interrupciones, y por lo tanto saber que número de byte correspondiente al CRC se está recibiendo. Así si el n = 0, significa que el byte recibido es el primer byte del CRC, si n = 1, significa que el byte recibido es el segundo byte del CRC.
- flashCorrupted: Variable global que sirve para indicar que la memoria flash del microcontrolador maestro ha sido corrupta. Está variable cambia su valor a ‘1’ cuando el byte correspondiente al CRC recibido no coincide con el de referencia.

Cada que se ejecuta la subrutina de interrupción quiere decir que se ha recibido un fragmento del CRC enviado por el maestro, el algoritmo puede determinar de que fragmento en específico se trata por medio del valor de la variable n, se compara el fragmento recibido con el byte correspondiente del vector CRC_OK, si el valor no es el mismo se puede asumir que la memoria flash ha sido corrupta y por lo tanto la bandera flashCorrupted se levanta. Posteriormente la bandera flashCorrupted será utilizada por otro módulo de software para determinar si se debe reprogramar el microcontrolador maestro.

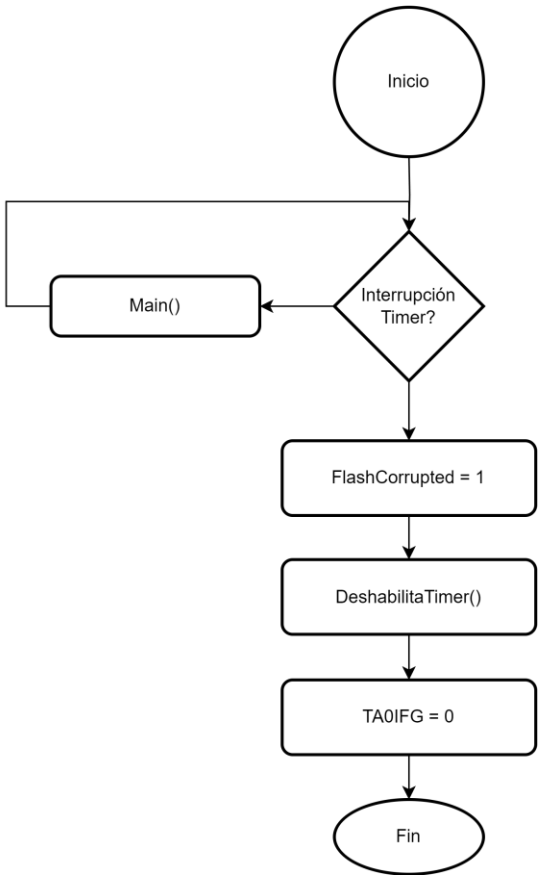

Figura 66.Supervisor - subrutina de interrupción de cuenta final en el timerA0

En la Figura 66 se muestra la subrutina de interrupción de cuenta final en el timer A0, este algoritmo resulta muy sencillo ya que la ejecución de está subrutina es debida a que el watchdog ha llegado a su cuenta final, lo que significa que la señal de alivio no ha llegado y, por lo tanto, la memoria flash ha sido corrompida. La subrutina de interrupción realiza lo siguiente

- cambia el valor de la bandera global *FlashCorrupted* a '1',
- Deshabilita el *timer* debido a que la memoria flash del microcontrolador maestro ha sido corrupta, por lo tanto, el microcontrolador maestro debe ser reprogramado, el esquema de supervisión será implementado nuevamente una vez que el microcontrolador maestro se encuentre funcional después de haber sido reprogramado.
- Se limpia la bandera de interrupción *TA0IFG.*

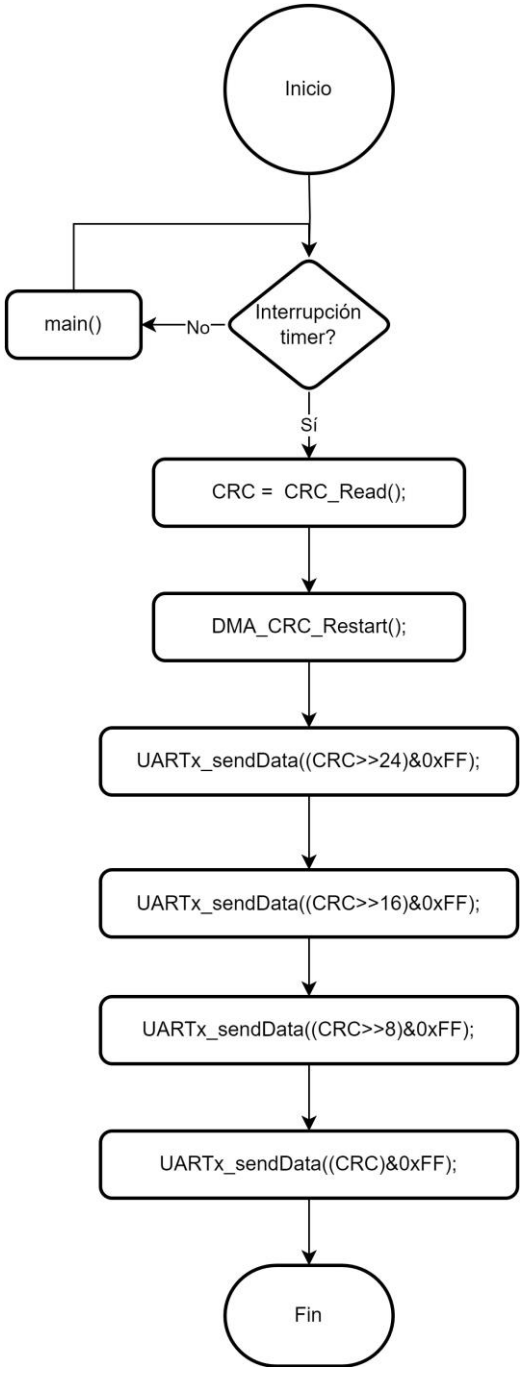

Figura 67. Maestro – Subrutina de interrupción de cuenta final en el timer.

En la Figura 67 se muestra la subrutina de interrupción de cuenta final en el timer implementada en el microcontrolador maestro, el objetivo de esta subrutina es enviar la señal de alivio hacia el microcontrolador maestro, la cual consiste en el CRC de su memoria flash. De manera general el algoritmo realiza lo siguiente:

- Se lee el CRC en el registro de lectura del periférico. En caso de que sea la primera vez que se ejecuta la subrutina de interrupción, el CRC del flash se encuentra disponible ya que la rutina de inicialización debe activar la transacción en el DMA para el cálculo del CRC.
- Se reinicia la transacción de datos del DMA para así calcular un nuevo CRC (El cual será leído la siguiente vez que la subrutina de interrupción se ejecute).
- Se envían los cuatro bytes correspondientes al CRC del microcontrolador maestro. Debido a que el CRC es de 32 bits es necesario realizar un algoritmo que envíe cada uno de los cuatro bytes del dato. El algoritmo realiza lo siguiente:
    - Para el byte más significativo, hace un corrimiento hacia la derecha de 24 bits, el corrimiento se aplica al CRC de 32 bits. ((CRC>>24)&&0xFF)
    - Para el segundo byte, hace un corrimiento hacia la derecha de 16 bits, el corrimiento se aplica al CRC de 32 bits. ((CRC>>16)&&0xFF)
    - Para el tercer byte, hace un corrimiento hacia la derecha de 8 bits, el corrimiento se aplica al CRC de 32 bits. ((CRC>>8)&&0xFF)
    - Para el byte menos significativo o cuarto byte, se realiza una operación AND con 0xFF. ((CRC)&&0xFF)

4.5.5.4 Implementación de la aplicación: Actualización del programa del microcontrolador maestro por medio de un dispositivo externo, así como el almacenamiento del nuevo programa en la memoria FRAM.

4.5.5.4.1 Almacenamiento del nuevo programa en FRAM.

En esta sección se abordará la actualización del programa del microcontrolador maestro, primero se detallará la recepción de este por medio de un dispositivo externo y el microcontrolador supervisor, para su posterior almacenamiento en

FRAM. Como se ha explicado en 4.5.5.2 para poder escribir el programa en FRAM es necesario segmentarlo.

Una vez que en 3.9 se ha explicado la información del formato Intel Hex es importante determinar qué datos son de interés para la copia del programa que estará almacenada en FRAM. En FRAM solo debe haber bytes de datos, por lo tanto, solo se escribirán en FRAM las líneas del archivo Intel Hex que tengan como tipo de registro: Datos. Una vez aclarado esto la información por cada línea del archivo Intel Hex que se escribirá en FRAM es la siguiente.

- Número de datos (En bytes): n
- Datos
- Checksum del número de datos y los datos.

$$Checksum\colon\ n \oplus dato_1 \oplus dato_2 \oplus \ldots \oplus dato_n$$

Se ha decidido no incluir a la dirección de los datos en FRAM, ya que de acuerdo con el mapa de memoria del microcontrolador STM32F407VG, la región del programa en flash comienza a partir de la dirección 0x08000000, por lo tanto, la dirección de los datos en flash puede ser calculada como sigue.

$$direccion\ en\ flash\ =\ direccion\ en\ FRAM\ +\ 0x08000000$$

Adicionalmente a los datos anteriormente mencionados es necesario incluir el número de líneas que componen al programa

0x000 0000					
NumSegmentos1	NumSegmentos2	NumSegmentos3	NumSegmentos4	NumDatos(N)	dato1
dato2	...	datoₙ	Checksum	NumDatos(N)	dato1
...	datoₙ	Checksum	NumDatos(N)	dato1	...
datoₙ	Checksum	NumDatos(N)	dato1	...	datoₙ
Checksum	NumDatos(N)	dato1	...	datoₙ	Checksum
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...

Figura 68. Distribución de datos en FRAM

En la Figura 68 se puede ver la distribución de datos en la FRAM, cada celda corresponde a una localidad de memoria cuyo tamaño es de un byte, los primeros

cuatro bytes corresponden al número de segmentos del programa (o el número de líneas de datos del archivo Intel Hex), la siguiente información después del dato antes mencionado, es la correspondiente a las líneas de datos, por cada línea de datos se tiene la siguiente información:

- Número de datos. (del grupo de datos correspondiente).
- Datos.
- Checksum del número de datos y los datos.

Una vez que se ha definido la distribución de datos en la FRAM, se pueden diseñar los algoritmos que servirán para llevar a cabo la función de actualizar en la FRAM la copia de respaldo del programa del microcontrolador maestro.

En la Figura 69 se muestra el algoritmo para enviar el nuevo programa del microcontrolador maestro, el cual tiene que ser implementado por el dispositivo externo, a continuación, se realizará un breve resumen de las acciones llevadas a cabo por el algoritmo:

- El algoritmo lleva la cuenta de los segmentos de datos presentes en el archivo Intel Hex, la cuenta se lleva por medio de la variable (NumSegmentos). Es necesario saber el número de segmentos, ya que gracias a este dato se puede saber el tamaño del programa del maestro.
- El algoritmo lee uno a uno los segmentos del archivo Intel Hex, determina de que tipo de segmento se trata, si se trata de un segmento de datos se envía la siguiente información del segmento:
    - Número de datos del segmento
    - Datos
    - Checksum del número de datos del segmento y los datos.

  Si se trata del segmento de fin, el dispositivo externo enviará la cuenta del número de segmentos (NumSegmentos). Para cualquier otro caso donde el tipo del segmento no sea de fin o de datos, el segmento será ignorado debido a las razones mencionadas anteriormente en esta sección.

- Se busca aumentar la robustez de la interfaz implementando un checksum correspondiente para cada grupo de datos, este checksum es verificado por parte del supervisor, en dato caso de que el checksum difiera, se asume que el grupo de datos ha sido corrompido y por lo tanto se vuelve a solicitar el grupo de datos.

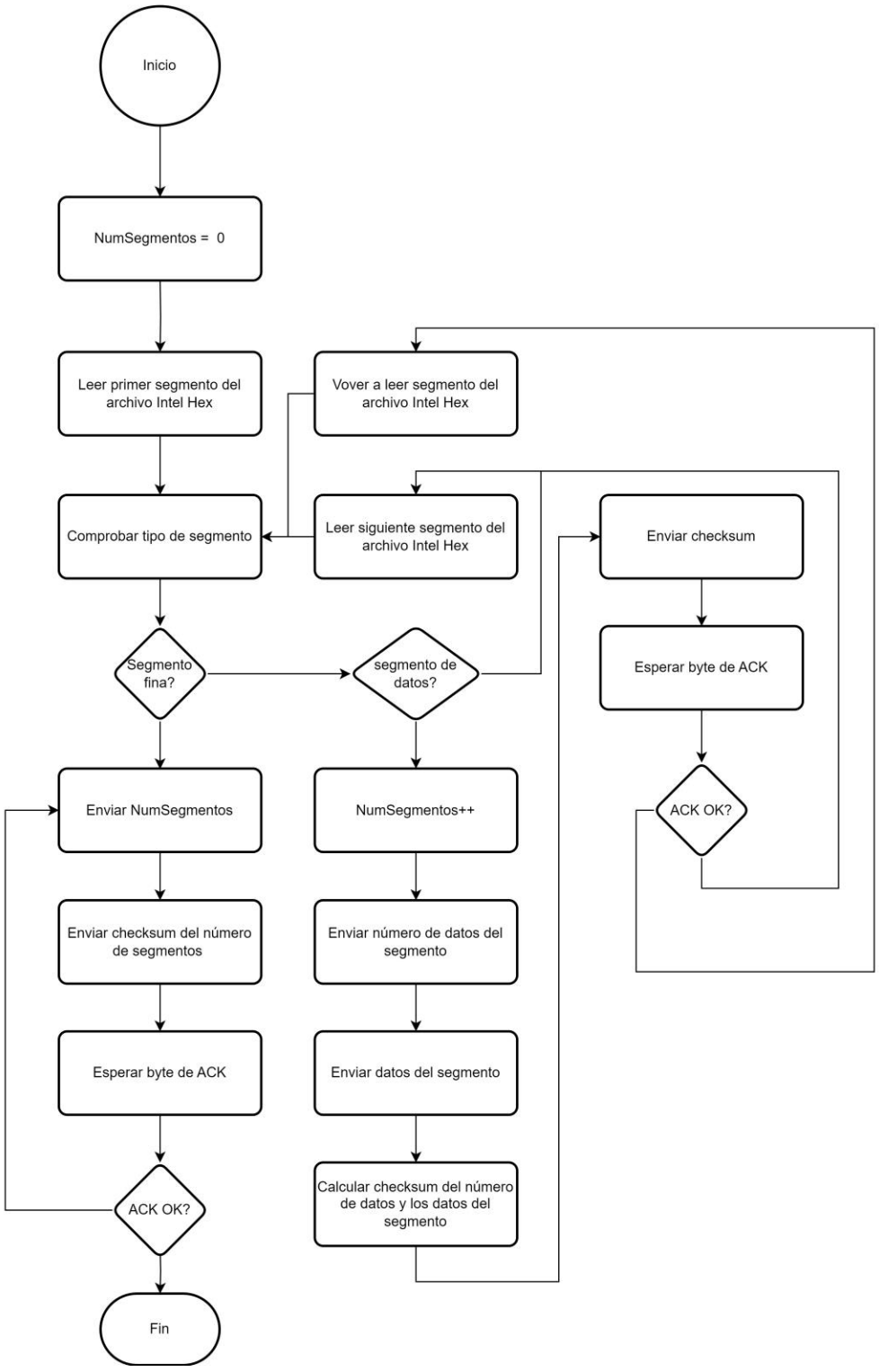

Figura 69. Algoritmo para el envío del nuevo programa del microcontrolador maestro - Dispositivo externo

En la Figura 70 se muestra el algoritmo para recibir el nuevo programa del microcontrolador maestro, el cual tiene que ser implementado por el

microcontrolador supervisor, a continuación, se realizará un breve resumen de las acciones llevadas a cabo por el algoritmo:

- El algoritmo lleva la cuenta de los bytes de datos presentes en el archivo Intel, la cuenta se lleva por medio de la variable (NumBytes). Es necesario saber el número de bytes que componen al programa, ya que gracias a este dato se podrá determinar el CRC de la región de programa en el flash del microcontrolador maestro.
- El algoritmo recibe uno a uno los segmentos del archivo Intel Hex, determina de que tipo de segmento se trata, si se trata de un segmento de datos se recibe la siguiente información del segmento:
    - Número de datos del segmento
    - Datos
    - Checksum del número de datos del segmento y los datos.

  Si se trata del segmento de fin, el microcontrolador supervisor recibirá la cuenta del número de segmentos (NumSegmentos). Para cualquier otro caso donde el tipo del segmento no sea de fin o de datos, el segmento será ignorado debido a las razones mencionadas anteriormente en esta sección.
- Se busca aumentar la robustez de la interfaz implementando un checksum correspondiente para cada grupo de datos, este checksum es verificado por parte del supervisor, en dato caso de que el checksum difiera, se asume que el grupo de datos ha sido corrompido y por lo tanto se vuelve a solicitar el grupo de datos.

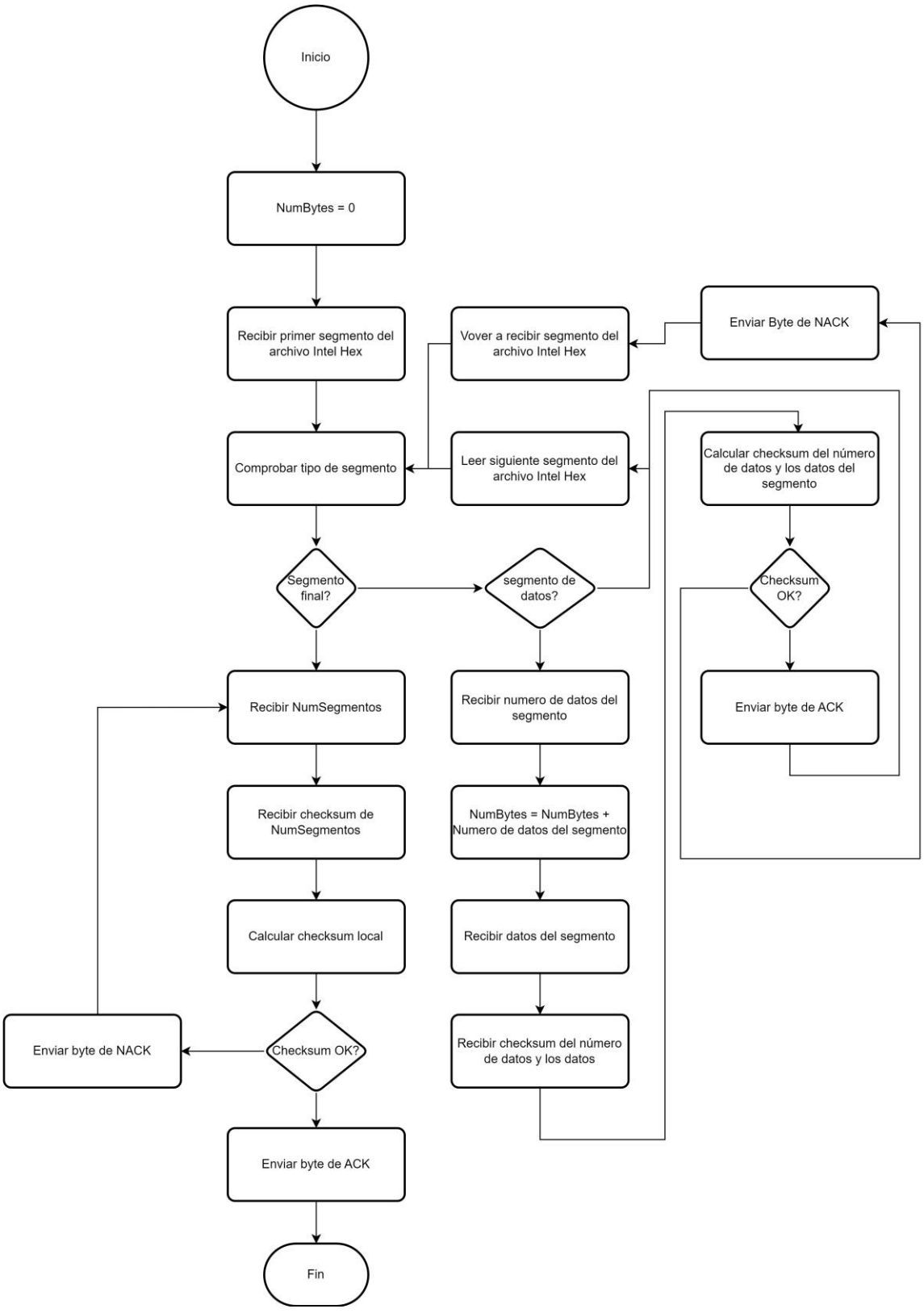

Figura 70. Algoritmo para la recepción del nuevo programa del microcontrolador maestro - Microcontrolador supervisor

4.5.5.4.2 Cargar el nuevo programa en el microcontrolador maestro.

Una vez que el nuevo programa se encuentra almacenado en FRAM, se explicará la rutina de reprogramación del microcontrolador maestro. Como se ha visto en 4.5.5.4.1 el programa se encuentra almacenado en segmentos, teniendo esto en cuenta, de manera general, se propone el siguiente esquema para actualizar el programa del microcontrolador maestro:

- Leer de FRAM un segmento del programa del microcontrolador maestro.
- Comprobar integridad de los datos leídos, en caso de que los datos se hayan corrupto, volver a leerlos.
- Si la integridad de los datos está bien, escribirlos en la memoria flash del microcontrolador supervisor.
- Realizar los pasos anteriores hasta que se termine de cargar el programa del microcontrolador maestro.

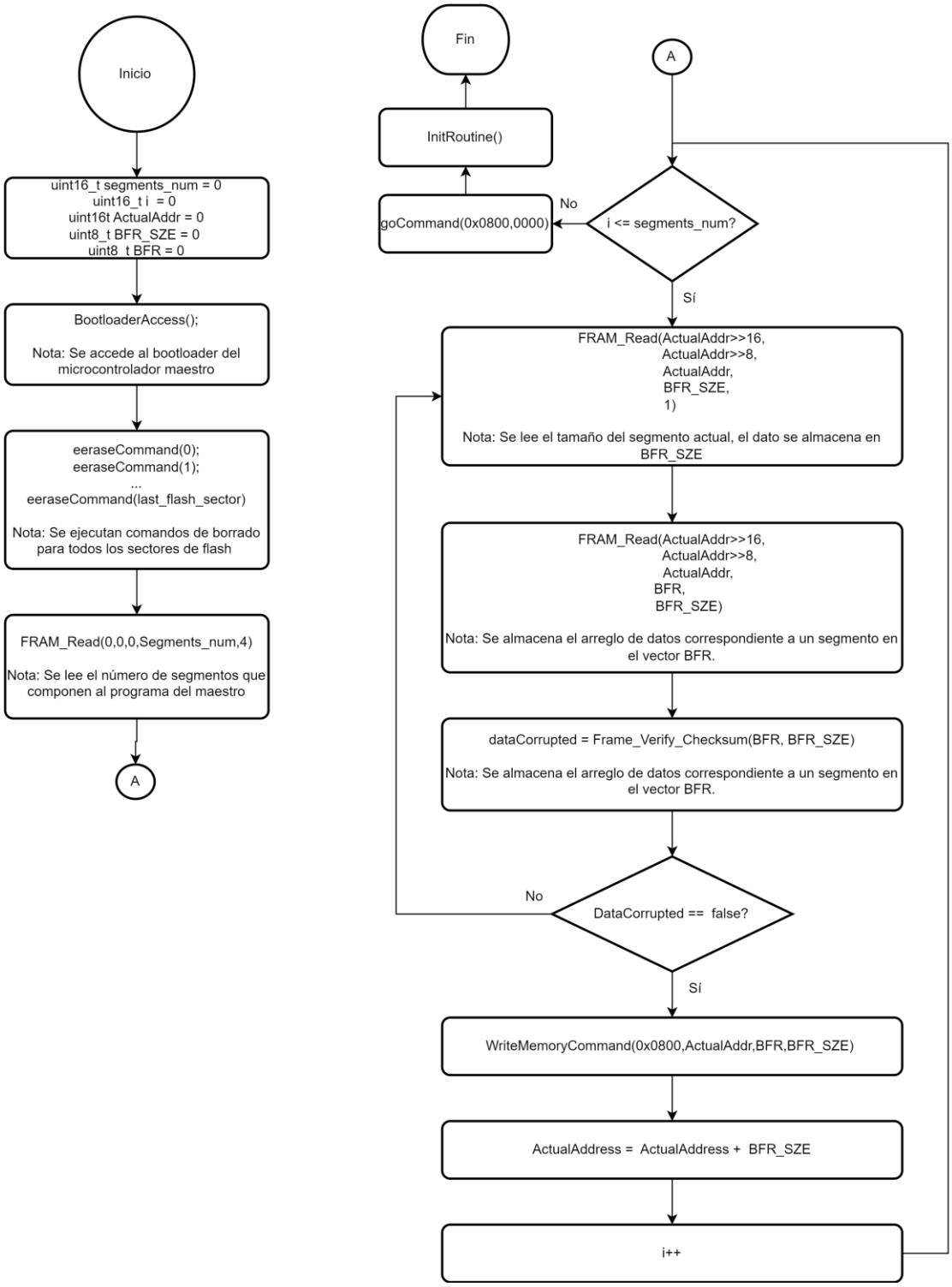

Figura 71. Reprogramación del microcontrolador maestro

En la Figura 71 se muestra la rutina para reprogramar el microcontrolador maestro, considerando que la copia de respaldo del programa ya se encuentra almacenada

en FRAM. Es importante mencionar que está rutina utiliza las funciones desarrolladas para el driver de la memoria FRAM y el del acceso al bootloader del microcontrolador maestro, las cuales se han presentado en 4.5.4. A grandes rasgos: El funcionamiento del algoritmo mostrado en la figura es el siguiente:

- Por medio del driver *STM32F407xx_bootloaderCommands* se accede al bootloader del microcontrolador maestro, se borran las secciones de memoria correspondientes a la región de código de la memoria flash. En este punto el microcontrolador se encuentra listo para ser reprogramado.
- Por medio del driver *FRAM_Commands* se leen el número de segmentos que corresponden al microcontrolador maestro.
- Se comprueba que el iterador *i* no sea mayor al número de segmentos, si esto fuera así significaría que no hay ningún segmento más por leer. Se lee el tamaño del segmento y se almacena en la variable *BFR_SZE.* Teniendo está información se procede a leer el segmento. Se comprueba la integridad del segmento por medio de la función *Frame_Verify_Checksum,* la cual calcula el checksum de los datos del segmento y lo compara con el checksum correspondiente en FRAM.
    - Si los datos se encuentran en buen estado, por medio del driver *STM32F407xx_bootloaderCommands* se escriben los datos del segmento en la memoria flash del microcontrolador maestro.
    - Si los datos se han corrupto, se vuelve a leer el segmento en FRAM.

  Este paso se repite hasta que el valor del iterador *i* sea igual al número de segmentos del programa del maestro.
- Una vez que se ha terminado de cargar el programa en la memoria flash del microcontrolador maestro, por medio del driver *STM32F407xx_bootloaderCommands* se ejecuta el comando *goCommand,* el cual sirve para comenzar la ejecución del programa en el microcontrolador maestro, posteriormente se ejecuta la rutina de inicialización y sincronización del esquema de supervisión llamada *InitRoutine()* la cual se ha explicado en

5 Resultados y conclusiones.

5.1 Resultados

Como se ha visto en 4.5.5 el diseño de la aplicación se ha dividido en dos grandes funcionalidades:

- Actualización del respaldo en FRAM del programa del microcontrolador maestro.
- Determinación del estado funcional del microcontrolador maestro y su reprogramación en caso de fallas.

Los resultados obtenidos se presentarán siguiendo está estructura.

5.1.1 Configuración del hardware para las pruebas.

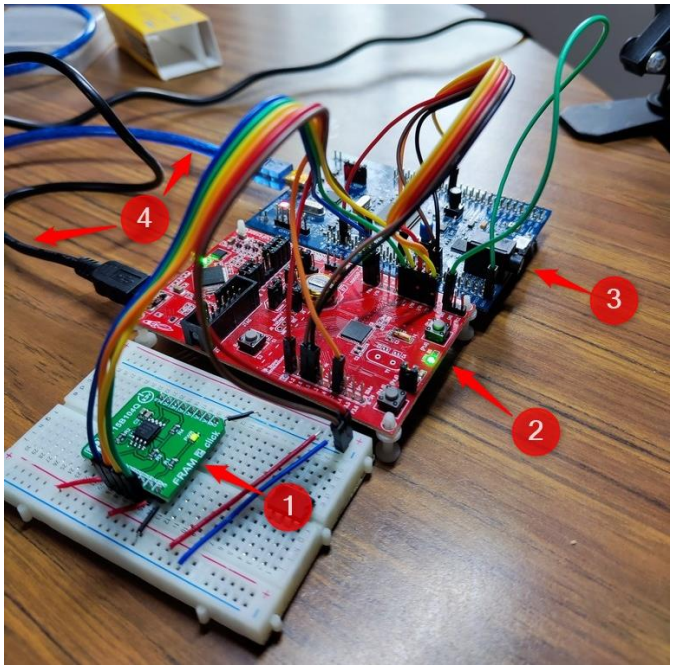

Figura 72. Configuración de hardware para las pruebas

En la Figura 72 se muestra la configuración de hardware utilizada para llevar a cabo las pruebas, es importante mencionar lo siguiente:

- En (1) se puede observar el modulo FRAM.
- En (2) se puede observar la tarjeta de desarrollo MSP-EXP430FR5969.
- En (3) se puede observar la tarjeta de desarrollo STM32F407G-DISC1
- En (4) se puede observar que el debugger de ambas tarjetas de desarrollo se encuentra conectado a una PC.
- Finalmente, también se puede observar el cableado entre las tarjetas de desarrollo y el módulo FRAM, el cableado implementa las interfaces presentadas en la Figura 57.

5.1.2 Actualización del respaldo en FRAM del programa del microcontrolador maestro.

Para probar esta funcionalidad, se ha hecho un script en Python que se ejecuta en una PC, lee el archivo Intel hex referente al programa de respaldo del microcontrolador maestro, accede a un puerto UART (en el cual debe estar conectado el microcontrolador supervisor), y envía el programa de acuerdo con el algoritmo presentado en la Figura 69, la cual presenta los pasos a seguir para enviar el respaldo del programa hacia el microcontrolador supervisor. A continuación, se mostrará la ejecución del script de Python en la consola de Visual Studio Code.

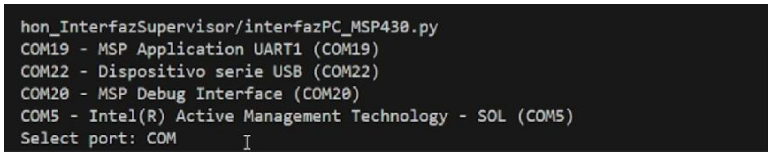

Figura 73. Script de Python - Selección del puerto UART.

En la Figura 73 se observa el inicio de la ejecución del script de Python, como se puede ver el script lista los puertos COM conectados a la PC y solicita seleccionar el puerto para establecer la comunicación con el UART del MSP430, en este caso será necesario seleccionar el puerto COM-19 – MSP Application UART1

Figura 74. Script de Python - Envío de los segmentos del programa

Después de haber seleccionado el puerto correspondiente para la interfaz UART, el script comienza a mandar los segmentos de programa, los cuales coinciden con las líneas del archivo Intel Hex, en la Figura 74 se muestra que el script imprime los datos del segmento que se está enviando en ese segmento (datos separados por comas, entre corchetes), así como el número de segmento que se está enviando (en este caso: 608), además se imprime la respuesta del microcontrolador supervisor, en este caso es 0x79 lo que significa que los datos han sido recibidos y comprobados por medio del algoritmo ejecutado en el microcontrolador supervisor y además, han sido escritos en la memoria FRAM.

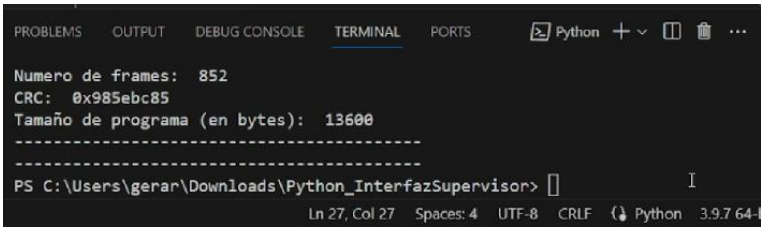

Figura 75. Script de Python. Resumen de la ejecución del programa

Una vez que se han enviado y se han escrito en FRAM exitosamente todos los segmentos de programa, se envía hacia el microcontrolador supervisor el CRC calculado, así como el tamaño del programa en bytes y posteriormente se termina la ejecución del script. En la Figura 75 se muestra lo que imprime el script en la consola de VSC: Número de segmentos, CRC y tamaño del programa. El CRC enviado hacia el microcontrolador supervisor es el que este utiliza como referencia para determinar si el microcontrolador maestro necesita ser reprogramado. Por otra parte, posteriormente el tamaño del programa en bytes se hace llegar al microcontrolador maestro por medio de la rutina de inicialización y sincronización

del esquema de supervisión, este dato es usado por el maestro para el cálculo del CRC del programa.

5.1.3 Determinación del estado funcional del microcontrolador maestro y su reprogramación en caso de fallas.

En esta sección se mostrarán los resultados obtenidos del software diseñado respecto a la determinación del estado funcional del microcontrolador maestro y su reprogramación en caso de fallas, ya que no se tiene un banco de pruebas que permita inducir SEE al sistema, se comprobará el funcionamiento del sistema por medio de la simulación del cambio del CRC de la memoria flash del microcontrolador, lo que simulará que alguna celda de memoria flash ha cambiado. A nivel de código esto se logrará por medio del envío de un vector de CRCs, el cual se muestra en la Figura 76

```
uint32_t CRC[] = {0xF0F0F0F0,0xF0F0F0F0,0xF0F0F0F0,0xF0F0F0F0,0xF0F0F0F0,0xF001800F};
```

Figura 76. Vector de CRCs para la validación del sistema

El vector contiene cinco CRC correctos (0xF0F0F0F0) y un CRC incorrecto (0x80018001), por lo tanto, durante la ejecución del programa en el maestro, se enviarán CRC correctos durante cinco veces lo que reiniciará al *watchdog* del lado del supervisor, sin embargo, el sexto CRC es incorrecto, por lo tanto, cuando este CRC sea recibido por el supervisor, este detectará que el CRC es incorrecto, debido a esto, se asume que la memoria flash del microcontrolador maestro ha sido corrupta, por lo tanto,el *watchdog* del lado del microcontrolador supervisor no será reiniciado y en consecuencia, el microcontrolador maestro será reprogramado.

Una vez se ha explicado el experimento, con un osciloscopio se medirán los puertos digitales que están involucrados en la interfaz entre el microcontrolador supervisor y el maestro:

- Puertos UART (TX y RX)

- Puertos BOOT0, BOOT1 y NRST.

Se considera que se puede ilustrar el funcionamiento del esquema de supervisión y reprogramación analizando el estado de los puertos digitales en los siguientes instantes:

- Primera medición: Antes de que se presente una falla.
- Segunda medición: En el instante en el que se presenta una falla.
- Tercera medición: El instante en el que el microcontrolador supervisor ejecuta la secuencia de acceso al bootloader del microcontrolador maestro, dando así inicio a la rutina de reprogramación.
- Quinta medición: Después de que se presenta una falla y el microcontrolador maestro ha sido reprogramado.

5.1.3.1 Primera medición: Funcionamiento sin fallas.

Esta medición busca mostrar la interacción entre el microcontrolador maestro y el supervisor antes de que la memoria del primero sea corrupta.

Puntos de medición:

- Canal 1: Puerto TX del microcontrolador maestro.
- Canal 2: Puerto NRST del microcontrolador maestro.

Figura 77. Interacción entre el microcontrolador supervisor y el maestro - Antes de la falla.

En la Figura 77 se muestra una fotografía de la medición del osciloscopio para esta etapa, como se puede apreciar, en el canal 1 se puede observar el dato enviado por UART el cual es 0xF0F0F0F0 y corresponde a un CRC correcto, debido a que el CRC es el esperado, la rutina de reprogramación no se ejecuta por lo que, como se puede observar en el canal 2 el puerto NRST permanece en estado bajo.

5.1.3.2 Segunda medición: Presencia de una falla.

Esta medición busca mostrar la interacción entre el microcontrolador maestro y el supervisor en el instante en el que la memoria flash del primero ha sido corrupta.

Puntos de medición:

- Canal 1: Puerto TX del microcontrolador maestro.
- Canal 2: Puerto NRST del microcontrolador maestro.

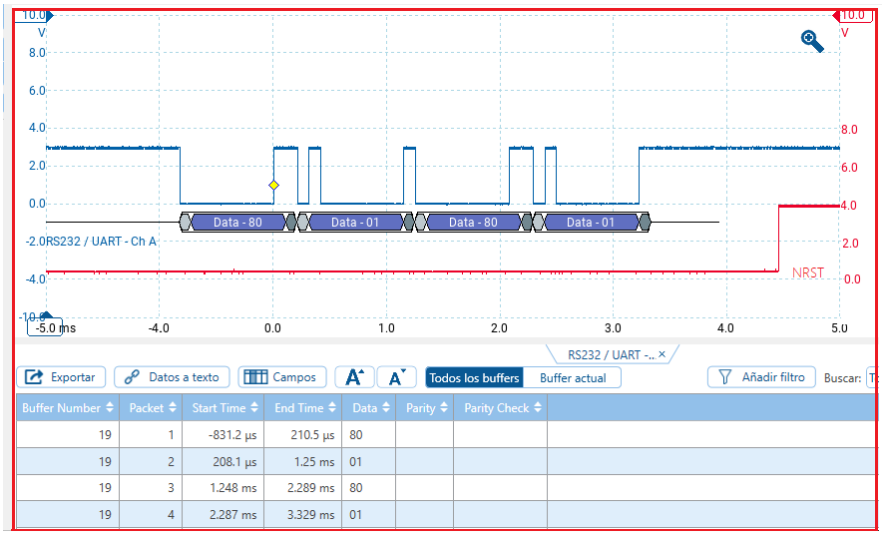

Figura 78. Interacción entre el microcontrolador supervisor y el maestro - Presencia de la falla (CRC incorrecto).

En la Figura 78 se muestra una fotografía de la medición del osciloscopio para esta etapa, como se puede apreciar, en el canal 1 se puede observar el dato enviado por UART el cual es 0x80018001 y corresponde a un CRC incorrecto, debido a que el CRC no es el esperado, la rutina de reprogramación se ejecuta por lo que, como se puede observar en el canal 2 el puerto NRST después de cierto tiempo pasa a estado alto, dando inicio a la secuencia de acceso al bootloader.

5.1.3.3 Tercera medición: Ejecución de la secuencia de acceso al bootloader del microcontrolador maestro.

Esta medición busca mostrar la interacción entre del microcontrolador maestro y el supervisor, cuando se está ejecutando la secuencia de acceso al bootloader del microcontrolador maestro.

Puntos de medición:

- Canal 1: BOOT0 del microcontrolador maestro.
- Canal 2: Puerto NRST del microcontrolador maestro.

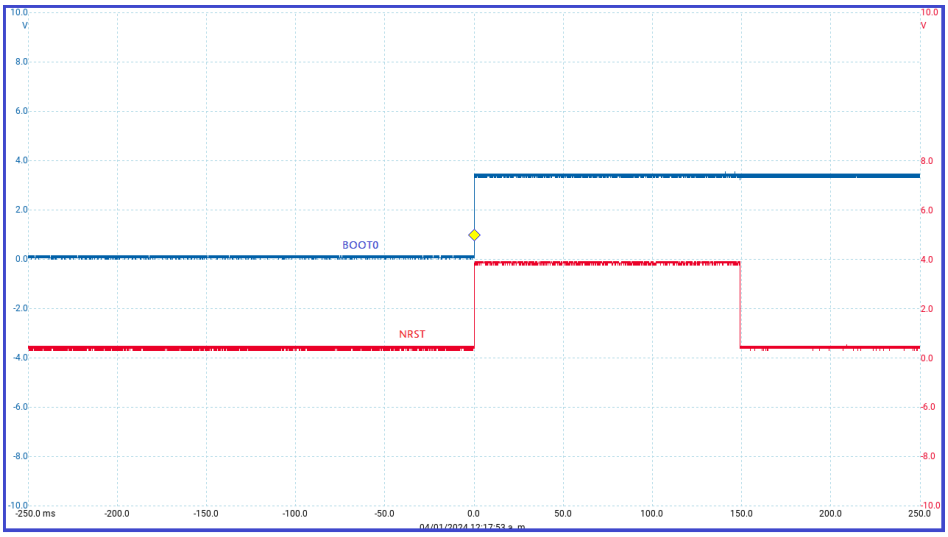

Figura 79. Interacción entre el microcontrolador supervisor y el maestro - Secuencia de acceso al bootloader.del microcontrolador maestro

En la Figura 79 se muestra una fotografía de la medición del osciloscopio para esta etapa, como se puede apreciar, en el canal 2 se observa el puerto NRST en estado bajo, mientras que en el canal 1 se observa el puerto BOOT0 en también en estado bajo, se puede ver que al mismo tiempo el puerto NRST y BOOT0 transitan a estado alto, después de unos instantes el puerto NRST vuelve a estado bajo, ejecutando así, la secuencia de acceso al bootloader y dando inicio a la rutina de reprogramación.

5.1.3.4 Cuarta medición: funcionamiento del microcontrolador maestro después de ser reprogramado.

Esta medición busca mostrar la interacción entre del microcontrolador maestro y el supervisor, cuando el primero ha sido reprogramado.

- Canal 1: Puerto UART TX del microcontrolador maestro.
- Canal 2: Puerto NRST del microcontrolador maestro.

Figura 80. Interacción entre el microcontrolador supervisor y el maestro - después de que el microcontrolador maestro ha sido reprogramado.

En la Figura 80 se muestra una fotografía de la medición del osciloscopio para esta etapa, como se puede apreciar, en el canal 1 está conectado el puerto UART TX del microcontrolador supervisor y se puede observar el dato enviado, el cual es 0xF0F0F0F0 y corresponde a un CRC correcto, por lo tanto, se puede asumir que el microcontrolador maestro ha sido reprogramado correctamente.

5.2 Conclusiones.

El desarrollo de nanosatélites con componentes COTS ha ido en aumento durante los últimos años, ya que estás tecnologías, debido a su bajo costo (relativamente), hacen a la industria espacial accesible para universidades e institutos de investigación. Los nanosatélites reducen el tiempo de desarrollo y el costo, en relación con satélites de mayor tamaño, sin embargo, como se ha visto en el estado del arte, la implementación de componentes COTS en nanosatélites da como resultado la necesidad de implementar esquemas de supervisión que monitoreen el estado funcional de la computadora principal de la misión, ya que las misiones satelitales se desarrollan en ambientes de radiación por lo que la probabilidad de fallas debidas a esto es considerable.

Este trabajo de tesis presenta el diseño de una arquitectura supervisora para la misión descrita en 4.1.1, esta arquitectura tiene la función de determinar el estado

funcional del microcontrolador maestro y reprogramarlo en caso de fallas, por otra parte, gracias a esta arquitectura también será posible actualizar el programa del microcontrolador maestro, una vez que el satélite se encuentra en órbita. De acuerdo con los resultados obtenidos se puede decir que el esquema de supervisión funciona correctamente, ya que cumple los objetivos descritos en esta tesis, un dispositivo externo puede enviar un nuevo programa hacia el microcontrolador supervisor, y este a su vez puede reprogramar al microcontrolador maestro con el programa nuevo. Por otra parte, el microcontrolador supervisor puede reprogramar al microcontrolador maestro ante cualquier cambio no esperado en su memoria flash.

La arquitectura supervisora desarrollada en este trabajo de tesis permitirá implementar componentes COTS en la computadora de a bordo, ya que, gracias al esquema de supervisión propuesto, se podrán corregir fallas causadas por el medio ambiente de radiación espacial.

La arquitectura de software desarrollada para este proyecto sigue un diseño por capas, lo cual permite a las capas más altas de la arquitectura ser independientes del hardware, ya que estas capas no interactúan directamente con este, sino que lo hacen a través de los controladores diseñados en las capas más bajas de software. La modularidad de esta arquitectura tiene muchas ventajas ya que permite realizar cambios de hardware sin grandes impactos en el software, por ejemplo, si se decidiera cambiar el microcontrolador, solo sería necesario realizar modificaciones a los controladores que interactúan directamente con el microcontrolador, mientras que todos los demás bloques de software permanecerían iguales, por otra parte esta arquitectura también resulta flexible en cuanto a la aplicación, por ejemplo, los controladores del microcontrolador y de los dispositivos podrían ser utilizados en cualquier otra aplicación que haga uso de los mismos.

6 Referencias

[1] K. T. Ulrich, Diseño y desarrollo de productos, McGraw Hill, 2013.

[2] Innovative Solutions In Space (ISIS), «ISISPACE,» 04 Noviembre 2022. [En línea]. Available: https://www.isispace.nl/product/on-board-computer/.

[3] Innovative Solutions In Space (ISIS), «ISIS-OBC Datasheet,» 2018.

[4] Group of Astrodynamic for the Use of Space Systems, «OBC ABACUS Flyer,» Rome-Italy, 2017.

[5] Group of Astrodynamics for the Use of Space Systems, «OBC ABACUS Datasheet,» 2017.

[6] C. M. Fuchs, «Fault Tolerant Nanosatellite Computing on a Budget,» de *33rd Annual AIAA/USU Conference on Small Satellites*, Utah, USA, 2019.

[7] T. Neubert, «In-flight Reconfiguration with System-on-Module based Architectures for Science Instruments on Nanosatellites,» de *35th Annual Small Satellite Conference*, 2021.

[8] S. May, «NASA,» 7 Agosto 2017. [En línea]. Available: https://www.nasa.gov/audience/forstudents/5-8/features/nasa-knows/what-is-a-satellite-58.html. [Último acceso: 04 Noviembre 2022].

[9] Erik Kulu, «Nanosats database,» [En línea]. Available: https://www.nanosats.eu/cubesat. [Último acceso: 04 Noviembre 2022].

[10] European Space Agency, «ESA,» [En línea]. Available: https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/CubeSats#:~:text=CubeSats%20are%20now%20commo

nly%20used,beginning%20to%20venture%20farther%20afield.. [Último acceso: 04 Noviembre 2022].

[11] European Space Agency, «ESA,» [En línea]. Available: https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Onboard_Computers_and_Data_Handling/Architectures_of_Onboard_Data_Systems. [Último acceso: 27 Noviembre 2022].

[12] National and Aeronautics and Space Administration, «NASA,» 31 Enero 2018. [En línea]. Available: https://www.nasa.gov/feature/goddard/2018/studying-the-van-allen-belts-60-years-after-america-s-first-spacecraft. [Último acceso: 27 Noviembre 2022].

[13] National Aeronautics and Space Administration, «nasa,» [En línea]. Available: https://www.nasa.gov/missions/analog-field-testing/why-space-radiation-matters/. [Último acceso: 27 Noviembre 2022].

[14] M. R, de *Encyclopedia of Quartenary Science (Second Edition)* , Elsevier , 2013, pp. 353-360.

[15] National Aeronautics and Space Administration, «radhome,» 19 Octubre 2021. [En línea]. Available: https://radhome.gsfc.nasa.gov/radhome/see.htm. [Último acceso: 27 Noviembre 2022].

[16] National Aeronautics and Space Administration, «NASA,» [En línea]. Available: https://www.nasa.gov/analogs/nsrl/why-space-radiation-matters. [Último acceso: 15 12 2022].

[17] European Space Agency, «ESA,» 12 Mayo 2019. [En línea]. Available: https://indico.cern.ch/event/777129/contributions/3249529/attachments/184

4695/3026130/6th_EIROforum_school_on_instrumentation_cpoivey.pdf. [Último acceso: 14 Diciembre 2022].

[18] European Cooperation for Space Standarization, «ECSS,» [En línea]. Available: https://ecss.nl/item/?glossary_id=69. [Último acceso: 15 Diciembre 2022].

[19] N. Murphy, «Watchdog Timers,» de *Emebedded Systems Programming*, 2000.

[20] W. Bjorndahl, «Redundancy Implementations and consideration of Related Failures in Spacecraft Electronic Systems,» *The Aeroespace Corporation.*

[21] T. Schimidt, *AN730 CRC Generating and Checking,* Microchip Technology Inc, 2000.

[22] T. L. Floyd, Sistemas Digitales, Madrid: Pearson, 2006.

[23] V. Rzehak, *SLAA502 Low-Power FRAM Microcontrollers and Their Applications,* Texas Instruments, 2019.

[24] Texas Instruments , *SLAT151- FRAM FAQs,* 2014.

[25] Texas Instruments, TI Space Products Guide, 2022.

[26] Cypress Semiconductor Corporation , *001-92249 Rev E CY15B104Q Datasheet,* California, 2017.

[27] STMicroelectronics, *STM32 microcontroller sytem boot mode. AN2606 Rev47,* 2021.

[28] STMicroelectronics, *USART protocol used in the STM32 bootloader. AN3155 rev 13,* 2020.

[29] STMicroelectronics, «DS8626 Rev 9.».

[30] B. A. Dahl, «Radiation Evaluation of Ferroelectric Random Access,» *IEEE,* p. 4, 2016.

[31] Texas Instruments, «FRAM FAQS - SLAT151».

[32] M. Bagatin, «Single and Multiple Cell Upsets in Memory cells,» *IEEE Transactions on nuclear science,* vol. 60, Agosto 2013.

[33] Texas Instruments, «SLAS704G. MSP430FR596x, MSP430FR594x Mixed-Signal Microcontrollers».

[34] STMicrocroelectronics, «AN4187. Using the CRC peripheral on STM32 microcontrollers,» 2022.

[35] R. Sobti, «Cryptographic Hash Function: A rewiew,» *IJCSI International Journal of Computer Science Issues,* vol. 9, nº 2, 2012.

[36] SANDIA National Laboratories, «Assessment of Commenrcial-Off-The-Shelf Electronics for use in a Short-Term Geostatationary Satellite,» Albulquerque, New Mexico , 2018.

[37] Agencia Espacial Mexicana, «Hacia el espacio,» 01 Enero 2014. [En línea]. Available: https://haciaelespacio.aem.gob.mx/revistadigital/articul.php?interior=86. [Último acceso: 20 Diciembre 2022].

[38] STMicroelectronics, *Reference Manual. RM0090,* 2021.

[39] STMicroelectronics, «DS8626 Rev 9. STM32F405xx STM32F407xx Datasheet,» 2020.

[40] W. Bjorndahl, «Redundancy Implementations and consideration of Related Failures in Spacecraft Electronic Systems,» *The Aeroespace Corporation,* 2011.