UNIVERSIDAD DE LA REPÚBLICA
Facultad de Ingeniería
Instituto de Computación – Instituto de Ingeniería Eléctrica

Informe de Proyecto de Grado

SOFTWARE Y PROTOCOLOS PARA CUBESAT

AUTOR:
GUSTAVO DE MARTINO

TUTORES:
Juan Pechiar – Instituto de Ingeniería Eléctrica
Ariel Sabiguero – Instituto de Computación

TRIBUNAL:
Andrés Aguirre
Federico Rodríguez
Leonardo Steinfeld

MONTEVIDEO – URUGUAY
2013

A mi familia y en particular a mi hija, por sobrellevar mis ausencias durante el largo camino que me ha llevado a trabajar en este proyecto.

A mis amigos, a los que también resté invalorable tiempo.

Agradecimientos

A mis compañeros del equipo de trabajo del IIE, por su generosa colaboración.

A Cecilia, Julio y Joaquín, por sus exhaustivas revisiones.

A mis tutores, por confiar en mí para desarrollar esta parte del proyecto.

Resumen

El trabajo presentado en este documento es parte del proyecto Antelsat desarrollado principalmente por el Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería de la Universidad de la República en Uruguay en colaboración con la empresa de comunicaciones estatal ANTEL.

El objetivo del proyecto es diseñar, construir y operar un pequeño satélite experimental en el que estudiantes, profesores y colaboradores trabajan en los diferentes módulos y su integración.

Al tratarse de un sistema crítico fue necesario utilizar y promover en el equipo un enfoque metodológico adecuado, además de realizar exhaustivas revisiones.

El documento describe el desarrollo del software para el módulo de control principal, los lineamientos para la realización del software de los restantes módulos, la definición de los protocolos de comunicación dentro del satélite, el protocolo de comunicación entre el satélite y las estaciones de tierra, así como el proceso de integración.

Dentro del conjunto de piezas de software construidas se destaca el kernel, un pequeño sistema operativo especializado para el ambiente de operación.

El énfasis se ha puesto en las decisiones de diseño y su motivación, entendiendo que es el mejor aporte que se puede realizar a futuros trabajos.

Palabras clave

Antelsat; Cubesat; Kernel; MSP430; Satélite; Sistemas críticos; Sistemas embebidos; Uruguay

Tabla de Contenido

Índice de figuras

Índice de tablas

Capítulo 1.

Introducción

Contenido

1.1 Organización del documento

El documento está organizado en capítulos. Al principio de cada uno se encuentra un índice detallado del contenido.

Los primeros capítulos: Introducción, Requerimientos y Estado del Arte, presentan el contexto, trabajo requerido, y la situación al inicio del proyecto.

El siguiente capítulo está dedicado al diseño. Describe las líneas generales de la solución planteada.

A continuación los capítulos “Kernel”, “Fecha y hora”, “Parámetros”, “Comandos”, “Eventos” y “Protocolos” describen los detalles más importantes de la implementación con una orientación enfocada a las necesidades de los futuros usuarios del sistema.

Finalmente los capítulos “Verificación e Integración”, “Gestión del Proyecto” y “Conclusiones” relatan el trabajo realizado y los resultados obtenidos.

Gran parte del trabajo fue dedicado a una pieza esencial del sistema: el Kernel. Los detalles relativos a sus funcionalidades uso y configuración se presentan por separado en un documento adjunto.

1.2 El proyecto Antelsat

Antelsat es un proyecto multidisciplinario en el que participan la Facultad de Ingeniería de la universidad pública uruguaya (Universidad de la República) y la empresa estatal de telecomunicaciones ANTEL.

El objetivo de este proyecto es diseñar, construir, poner en órbita y operar un pequeño satélite experimental de la familia CubeSat.

El proyecto Antelsat se construye con dos módulos CubeSat (10 x 10 x 20 cm), para alojar la carga científica y los sistemas de soporte (Aviónica)

1.3 El proyecto CubeSat

El proyecto CubeSat tiene como propósito proveer un estándar para diseñar pequeños satélites reduciendo costos y tiempo de desarrollo, además de facilitar el acceso al espacio a pequeñas cargas científicas. Este proyecto comenzó como un esfuerzo conjunto entre el Prof. Jordi Puig-Suari de la Universidad de California (California Polytechnic State University), San Luis Obispo, y el Prof. Bob Twiggs del Laboratorio de desarrollo de sistemas espaciales de la Universidad de Stanford.

Este estándar [5] –llamado "CubeSat Design Specification"– define un módulo CubeSat como un cubo de 10 cm de lado, con una masa de hasta 1.33 Kg., además de una cantidad de especificaciones relativas a la estructura, materiales y comportamiento.

Los CubeSat se incluyen como carga secundaria de otros lanzamientos.

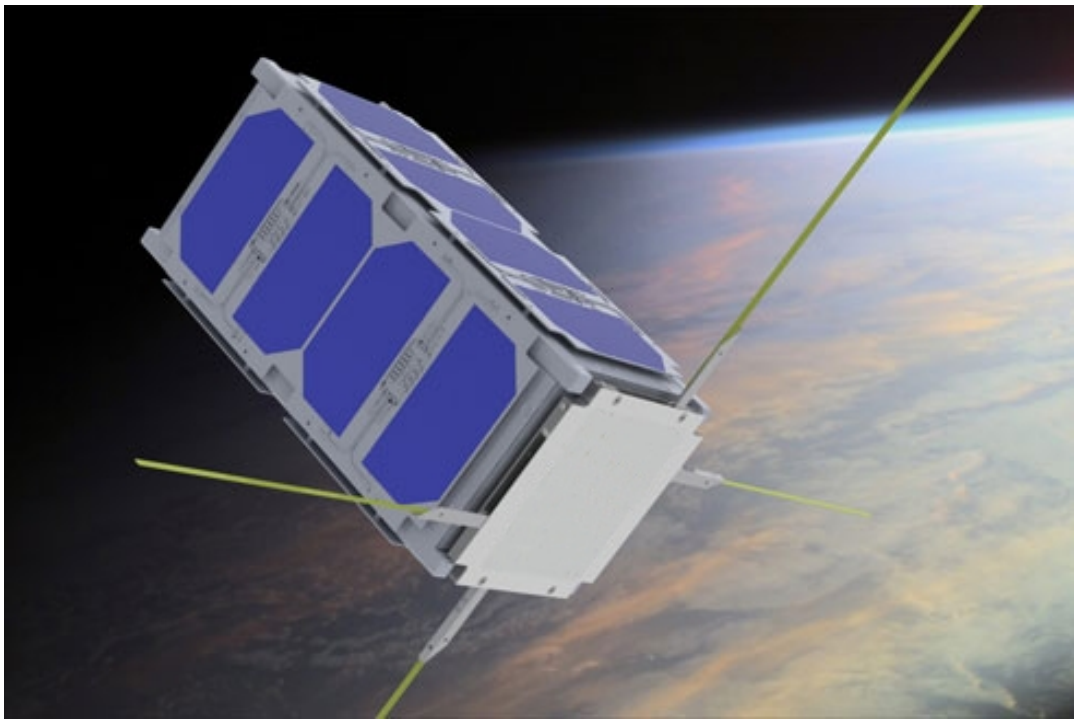

Figura 1 – CubeSat de dos módulos

Para garantizar la seguridad del satélite, el vehículo de lanzamiento, la carga primaria y los restantes CubeSat, los desarrolladores están obligados a cumplir los requisitos mínimos de diseño, verificación y operación definidos en el estándar.

La universidad de California desarrolló un lanzador específico para estos pequeños satélites (P-POD) que puede alojar hasta tres módulos CubeSat en uno o varios satélites.

Los satélites CubeSat pueden construirse en uno, dos o tres módulos.

Este lanzador es un sistema mecánico que incluye una puerta y un mecanismo de resortes donde se alojan los módulos para su transporte. Su diseño determina las características mecánicas de los satélites y su operación define limitaciones eléctricas y de comportamiento que se reflejan en los requisitos.

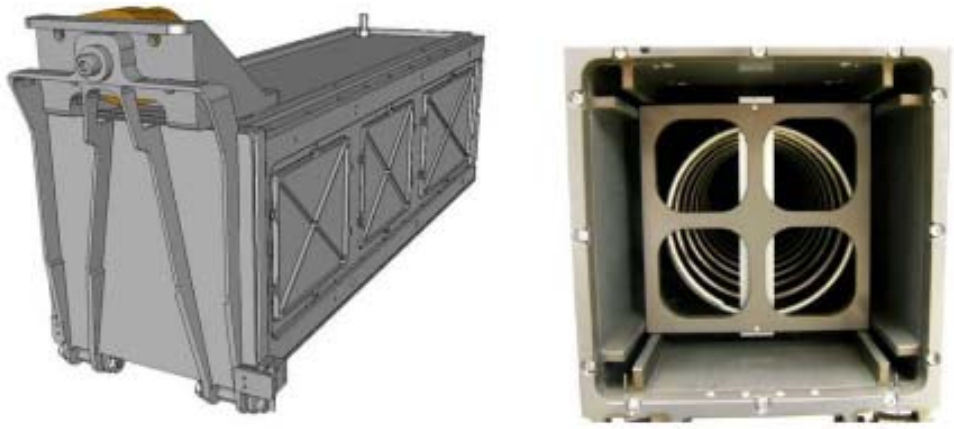

Figura 2 – Lanzador P-POD

1.4 Antecedentes

La propuesta de fabricar un satélite surge a finales de 2006. Desde entonces, varios grupos han estado trabajando en distintas partes del proyecto identificado como LAÍ.

Entre 2007 y 2008 el trabajo “Telemetría para Globosats” alcanzó el objetivo de desarrollar el sistema de telemetría para Globos Satelitales. Los Globos Satelitales son sistemas autónomos suspendidos de un globo de tipo meteorológico el cual permite llevar a cabo ensayos a gran altura. Este trabajo incluyó el diseño y desarrollo del sistema de comunicación entre el GloboSat y las estaciones terrestres, así como el software de control de los diferentes módulos para la plataforma.

En 2008 se liberaron exitosamente GloboSat 01, 02 y 03 y en 2009 GloboSat 4.

Entre 2009 y 2010 se desarrolló el proyecto “Sistema de gestión de energía para un satélite” con el objetivo de garantizar el suministro energético del satélite durante su vida útil.

Entre 2010 y 2011 se desarrolló el proyecto “Módulo de determinación de actitud” que determina los instrumentos a utilizar e implementa un prototipo de hardware y software. En esos años se realizó también el proyecto “Sistema de telemetría, telecomando y control”.

Entre 2011 y 2012 se desarrolló el proyecto “Módulos de control principal y control de actitud” durante el cual se construyó el prototipo de hardware y software de la placa electrónica que implementará los módulos ADCS (Detección y control de actitud) y MCS (Control principal) así como también el proyecto “Estación terrena”.

En agosto de 2011, Antel y la Facultad de Ingeniería de la Universidad de la República firmaron un acuerdo para la construcción del primer satélite uruguayo: Antelsat. Para el desarrollo del proyecto que abarca las etapas de diseño, construcción y operación, Antel aporta capital, se ocupa de la carga científica del satélite y de parte de las comunicaciones.

Figura 3 – GloboSat 4 en noviembre de 2009

1.5 Ambiente de operación

El satélite orbitará en los límites entre la termosfera y la exosfera. Esta última es la capa más externa de la atmósfera en la que la concentración de gases es extremadamente baja, así como la protección de radiación electromagnética, muy especialmente en la parte de la órbita que nos interesa, sobre la Anomalía del Atlántico Sur (SAA).

La SAA –el área grande roja en la figura 4– es una depresión en el campo magnético de la Tierra que permite que los rayos cósmicos y partículas cargadas alcancen niveles inferiores de la atmósfera. Esta depresión del campo magnético coincide con el aumento de interferencias en la comunicación con los satélites, aviones y transbordadores espaciales Si bien hay teorías en cuanto a por qué ocurre esto, el origen geológico aun se desconoce.

Los datos fueron recogidos por el detector Anomalía del Atlántico Sur (SAAD) a bordo del satélite ROSAT.

La misión ROSAT (1990-1999), basada en el satélite Röntgen, fue un observatorio de rayos X desarrollado por Alemania, Estados Unidos y el Reino Unido que, entre otras mediciones, relevó datos precisos acerca de la anomalía magnética del Atlántico Sur.

La radiación puede causar errores en dispositivos lógicos. Esto ocurre cuando una partícula de alta energía golpea al dispositivo y pueden acontecer una o dos situaciones:

Cuando la partícula atraviesa el dispositivo, ioniza el material en su camino. Esta ruta actuará como un corto circuito. El fenómeno se conoce como un Single Event Latch-up (SELU)

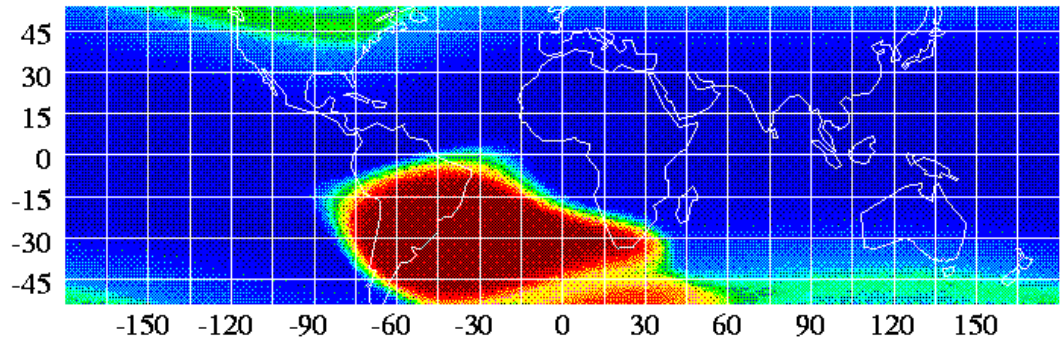

Figura 4 – Anomalía del Atlántico Sur

Por otra parte, cuando la partícula golpea un átomo con energía suficiente como para dividirlo, cada parte traza un camino de partículas ionizadas a través del material del dispositivo.

Este fenómeno se conoce como un Single Event Upset (SEU) y puede alterar el estado de las memorias, tanto RAM como FLASH, registros del procesador o periféricos, provocando errores de datos y programa.

La ausencia de gases impide la transferencia de calor por convección provocando grandes diferencias de temperatura entre las caras del satélite.

Las temperaturas diferirán en el tiempo según el satélite esté expuesto o no al sol.

1.6 Relevancia del trabajo

El objeto de este trabajo no es novedoso, existe mucha experiencia en desarrollo espacial a nivel internacional, sin embargo, la información publicada no lo refleja. Los motivos están en la mayoría de los casos relacionados con los objetivos bélicos o de seguridad.

El desarrollo de sistemas críticos como este no es aplicable únicamente en el área en particular. Las características de este sistema son muy similares desde el punto de vista de la calidad del producto a las de un equipo biomédico implantable. En cualquiera de esos casos un error en el diseño puede tener consecuencias fatales. En este caso, las consecuencias pueden afectar no solo a la misión sino también a las demás misiones que la acompañan en el lanzamiento.

Para Uruguay y su Universidad estos son los primeros pasos. El proyecto se ha desarrollado desde el año 2006 pero el software comienza a considerarse seriamente con este trabajo y sus contemporáneos.

1.7 El problema a resolver

A principios del año 2012 se estaban completando los primeros prototipos de hardware de los módulos que componen el satélite.

El diseño de esos prototipos fue realizado por un equipo de especialistas en electrónica y telecomunicaciones. Ese diseño consideró ligeramente los requerimientos del software que debería ejecutar las operaciones sobre ese hardware.

La necesidad inicialmente planteada era el desarrollo del software para el Módulo de Control Principal –una de las piezas centrales de la operación del sistema– pero el problema era mucho mayor. Las necesidades no estaban totalmente definidas, ni tampoco la interfaz con el resto de los sistemas.

El problema era entonces: determinar las necesidades de información y comunicación de los subsistemas que componen el satélite, diseñar los protocolos para comunicar esos subsistemas entre si y con las estaciones en tierra, satisfacer las necesidades de información, además de diseñar e implementar el software del Módulo de Control Principal.

1.8 Objetivos generales

El objetivo de este trabajo es obtener una versión de vuelo del software del Módulo de Control Principal.

Además incluye el diseño e implementación de los protocolos de comunicación entre módulos y el protocolo de comunicación con tierra.

1.9 Metodología utilizada

Los prototipos antecedentes carecían de modularidad y las escasas piezas desarrolladas tenían muy alto grado de acoplamiento. Parte del proceso inicial fue identificar los módulos a construir.

Una vez individualizados los módulos, se diseñó cada uno de ellos sin definir su interfaz. Estando definidos los componentes se implementaron prototipos que ayudaron a determinar las entradas y salidas requeridas.

A partir de los resultados de ese trabajo se definió la arquitectura del sistema, sus módulos e interfaces. Posteriormente se construyeron nuevos módulos, se implementaron los protocolos, se verificaron las piezas, se integraron y validaron.

Para poder separar estas piezas resultó imprescindible construir un subsistema que permitiera trabajar modularmente –el Kernel–.

El proceso de integración se realizó partiendo de un sistema básico al que se agregaron uno a uno los módulos, validando su funcionamiento con el hardware y con las partes integradas previamente.

Capítulo 2.

Requerimientos

Contenido

2.1 Requisitos generales

El objetivo de este trabajo fue obtener una implementación robusta del software del Módulo de Control Principal. El trabajo realizado incluyó el diseño de los protocolos de comunicación del módulo con los restantes, el refinamiento del protocolo de comunicación del satélite con las estaciones terrenas y la promoción de algunos cambios en el software y hardware de los otros módulos.

A continuación se detallan los requisitos globales de la implementación.

2.1.1 Manejador de parámetros

Los módulos del satélite realizan su operación dependiente de parámetros que se necesita modificar desde tierra. Deben identificarse estos parámetros, gestionarse y mantenerse en el Módulo de Control Principal.

2.1.2 Procesador de comandos

Es necesario enviar comandos a los módulos del satélite. Estos comandos deben ser generados por fuentes autorizadas. Con este objetivo se cifrarán los datos enviados usando el protocolo TEA. Deben determinarse los comandos e implementarse las acciones asociadas a los comandos recibidos por el satélite.

2.1.3 Registro y reporte de estado del satélite

Es necesario conocer el estado del satélite. A esos fines deben recolectarse periódicamente datos de todos los módulos, almacenarse en el Módulo de Control Principal, y transmitirse cuando sea solicitado por las estaciones de tierra.

2.1.4 Protocolo de comunicación Tierra – Satélite

La escasa potencia de transmisión del satélite, las variaciones en la frecuencia derivadas de los cambios de temperatura y la gran atenuación debido a la distancia, hacen que la comunicación sea muy delicada. Es necesario contar con un protocolo de datos sobre AX25 que resulte adecuado a estas condiciones.

2.1.5 Capa de enlace AX25

Durante la recepción, los módulos de comunicaciones realizan la detección de tramas AX25. Es indispensable contar con una implementación del resto de la capa de enlace revisando los campos de dirección y control además de tomarse acciones con los paquetes resultantes.

Para transmitir mensajes a tierra, es necesario construir tramas AX25 válidas a partir de los paquetes de datos generados en el satélite, definir e implementar un protocolo de interacción con el módulo de comunicación COMM2, el que se ocupa de agregar las banderas de inicio y final, realizar el “bit stuffing” y calcular el campo de verificación (FCS).

2.1.6 Servicio de repetidora para terceros (Digipeater)

Debe implementarse el servicio de repetidora definido en el protocolo AX25 y proveerse los mecanismos para habilitado y deshabilitado desde tierra.

2.1.7 Protocolo de comunicación entre módulos

El único medio de interconexión entre los módulos del satélite es el bus I²C. Un bloqueo de este bus es equivalente a un bloqueo total del satélite. Es necesario definir e implementar un protocolo de comunicación entre módulos que garantice la continuidad del servicio.

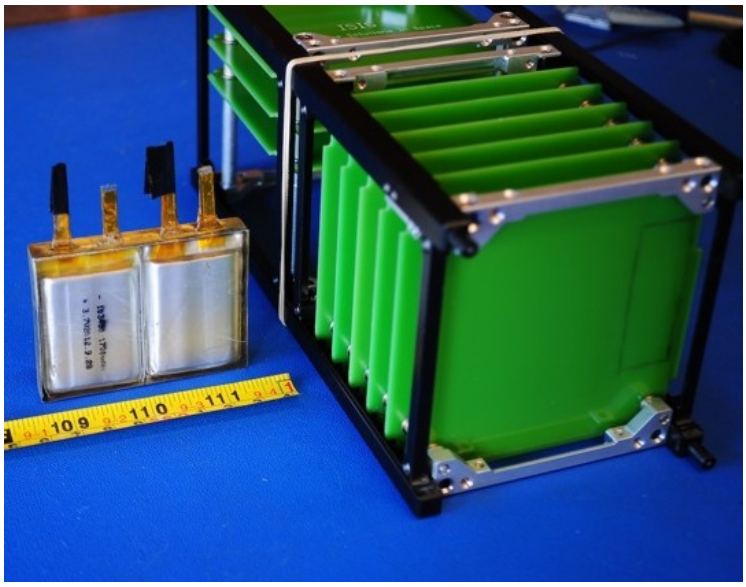

Figura 5 – Estructura del satélite

2.1.8 Persistencia ante reinicios

El Módulo de Control Principal debe mantener la configuración del satélite y la información recolectada. Las condiciones ambientales no garantizan continuidad en la operación. Ya sea por eventos de SELU o SEU como por falta de energía, se producirán reinicios o apagados temporales del módulo. Los datos esenciales deben perdurar, y ser consistentes frente a estas situaciones.

2.1.9 Mantenimiento de la hora del satélite

La hora es un parámetro esencial para el Control de Actitud y necesario para registrar los eventos ya que junto con la órbita determinan la posición del satélite. Las estaciones terrenas enviarán periódicamente una actualización de la hora. El Módulo de Control Principal debe mantenerla, ajustarla cada vez que reciba una actualización, así como reportarla a los módulos cuando estos la necesiten.

2.1.10 Bajo consumo de energía

El balance energético del satélite es muy delicado por lo que es necesario minimizar el consumo de energía.

2.1.11 Hardware y plataforma de desarrollo

La implementación del Módulo de Control Principal se realiza sobre un microcontrolador fabricado por Texas Instruments modelo MSP430F5438A. El desarrollo de software debe realizarse con la herramienta "IAR Embedded Workbench".

2.1.12 Simplicidad

El diseño debe mantenerse tan simple como sea posible. El procesamiento de los datos así como la toma de decisiones complejas deben realizarse en tierra.

2.2 Requisitos particulares

Los requisitos particulares de la misión eran desconocidos en las primeras etapas del desarrollo. Al tratarse de un trabajo de investigación, los requisitos finales fueron construyéndose con la colaboración del resto del equipo.

Capítulo 3.

Estado del arte

Contenido

3.1 Especificaciones técnicas de Antelsat

A continuación se presentan las principales características del satélite Antelsat. Posteriormente se detallan los elementos relevantes para el presente desarrollo.

3.1.1 Estructura y dimensiones

- Diseño clase CubeSat 2U
- 5 caras dedicadas a paneles solares
- Cara Nadir dedicada a sensores de carga científica y antenas de banda S

3.1.2 Sistema de gestión de energía

- Alimentación individual por módulo, incluyendo recuperación frente a latch-up, reinicio automático y desconexión de módulos con fallas.
- Inicio de operación del satélite y despliegue de antenas de acuerdo a las especificaciones de lanzamiento.
- Generación directa de baliza CW durante el modo seguro
- Operación MPPT (Maximum power point tracking) de los paneles solares.
- Gestión de baterías redundantes de Litio - polímero.

3.1.3 Comunicaciones

- Receptor VHF en banda amateur de 2 metros, protocolo AX25 1200 bps
- Enlace de carga de telecomandos
- Servicio de repetidora para comunicaciones de terceros
- Transmisor UHF en la banda amateur de 70 cm.
- Baliza CW (Morse)
- Enlace de descarga de telemetría a 1200 bps sobre protocolo AX25.
- Enlace de descarga de respaldo para imágenes de baja resolución por SSTV
- Transmisor de banda S en 2.4 GHz
- Enlace de descarga para imágenes de la carga científica.
- Enlace de respaldo para telemetría.
- Transmisores redundantes

3.1.4 Control de Actitud

- Determinación de actitud mediante magnetómetros, giróscopos y foto detectores
- Control activo en tres ejes mediante bobinas.

3.1.5 Carga científica

- Cámara infrarroja
- Cámara de espectro visible

3.2 Estructura y subsistemas del satélite

El satélite completo con sus antenas sin desplegar ocupa un espacio de 10 x 10 x 20 cm.

Una de las caras de 10 x 20 cm. –la que apunta a la tierra (nadir), aloja los sensores de dos cámaras, una en el espectro visible y otra infrarroja además de una antena direccional plana de alta frecuencia que se utiliza para transmitir las imágenes de alta resolución. Las restantes caras están cubiertas con paneles solares. Cada una de las seis caras cuenta además con un sensor de luz.

Flejes metálicos que se despliegan luego de la puesta en órbita conforman las antenas de VHF y UHF.

El satélite internamente es un conjunto de placas electrónicas que se intercomunican por un bus. Estas placas contienen los componentes que implementan los diferentes subsistemas.

El bus de interconexión contiene además de las líneas de alimentación, un bus de comunicación I^2C.

Tres de las caras del satélite -una en cada eje- tienen también giroscopios y bobinas que se utilizan para alinear el satélite respecto al campo magnético de la tierra.

3.2.1 Carga científica

La carga científica –o Payload– utiliza la cara nadir del satélite y uno de los módulos CubeSat (50% del volumen y la masa) para alojar los sensores y los sistemas de procesamiento y descarga de imágenes.

3.2.2 Subsistema de gestión de energía

Compuesto por los paneles solares, un conjunto de baterías y una placa de control, el subsistema de gestión de energía se ocupa de obtener el mejor rendimiento posible de los paneles y las baterías, encender y apagar los restantes subsistemas, protegerlos de sobre consumos, desplegar las antenas, generar la información de la baliza Morse.

3.2.3 Subsistema de comunicaciones

Dos módulos independientes de comunicaciones con receptores trabajando a frecuencias diferentes, hacen desmodulación por software de las señales recibidas en las antenas. Cada uno de los módulos tiene un trasmisor UHF. Uno trasmite la baliza y el otro los paquetes de datos.

En la recepción, los módulos detectan tramas AX25 identificando el conjunto de datos entre dos banderas y verificando una secuencia de control (FCS).

El módulo COMM2 construye tramas AX25 agregando a los paquetes de datos para transmitir recibidos a través del bus I²C, las banderas de control inicial y final, además del valor de control FCS y algunos bits para evitar que se generen banderas entre los datos.

3.2.4 Subsistema de control de actitud

La carga científica del satélite está compuesta por dos cámaras y un sistema de transmisión de datos direccional, todos elementos que requieren una actitud determinada.

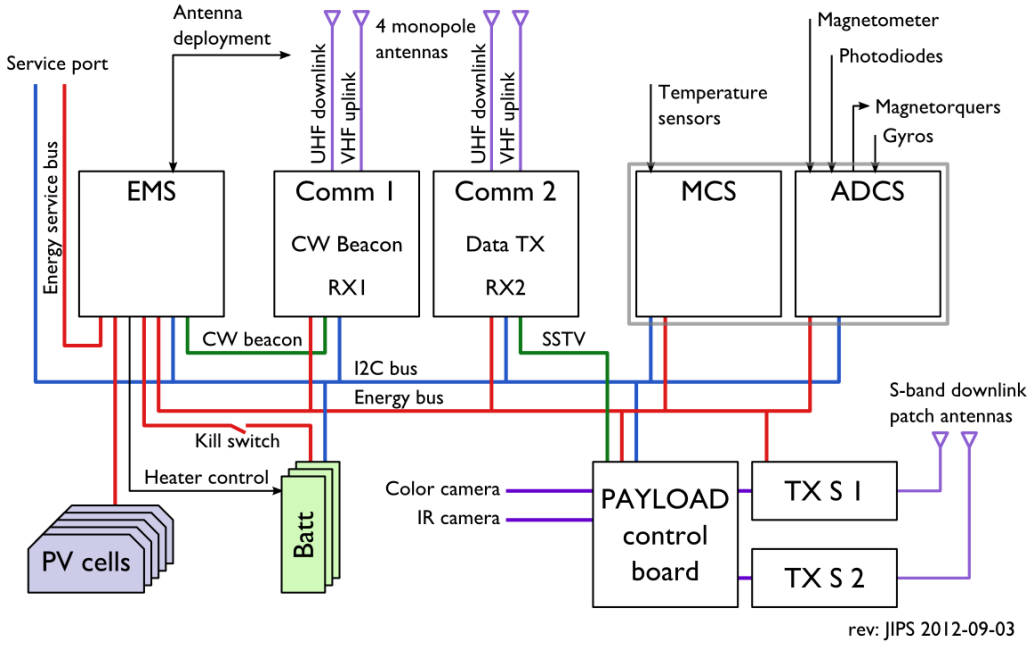

Figura 6 – Diagrama de bloques de AntelSat

Para obtener esta actitud, es necesario disponer de un sistema de control capaz de determinar la actitud instantánea, calcular el error entre la actitud actual y la deseada, y aplicar momentos para eliminar el error. El sistema de determinación de actitud es un componente clave en gran parte de los satélites, y su desempeño influye directamente en el resultado de las misiones.

Al día de hoy, se han desarrollado diversos métodos, incluyendo una gran variedad de sensores y algoritmos que permiten determinar y controlar la actitud de los vehículos espaciales con exactitud.

Utilizando las medidas de los sensores de luz ubicados en cada cara, se determina la posición real del sol. Con un magnetómetro de tres ejes se determina el campo magnético actual. A partir de la hora UTC y conociendo la trayectoria, se determina la ubicación teórica del satélite.

Usando esta ubicación, se determina la posición del sol esperada. Con la ubicación y un modelo del campo magnético de la tierra se determina la dirección esperada del campo magnético.

El módulo de detección y control de actitud acciona las bobinas de torque para rotar el satélite a la actitud deseada calculando la corrección a partir de la diferencia entre la situación actual y la situación objetivo.

3.2.5 Subsistema de control

El subsistema de Control o Módulo de Control Principal es el nodo central de comunicaciones y almacenamiento de datos del satélite.

Este realiza la interpretación y el procesamiento de los mensajes recibidos, la gestión de los parámetros de todos los módulos, el mantenimiento de la hora del sistema y el registro de eventos del satélite.

Constituido por un procesador MSP430F5438A fabricado por Texas Instruments y unas pocas piezas de hardware adicional, el módulo se integra a la placa del ADCS.

3.3 Contexto de hardware

El proyecto antecedente [2], determinó que el módulo de control principal utilizaría un procesador MSP430F5438A [25]. Se trata de un microcontrolador RISC con registros de 20 bits, de muy bajo consumo que integra gran cantidad de periféricos en el mismo chip, además de incluir 16 KByte de memoria RAM y 256 KByte de memoria FLASH.

En la tabla 1 se muestran las direcciones de los rangos de memoria del microcontrolador.

Los dispositivos de entrada salida están mapeados en las direcciones mas bajas de memoria.

Los registros y el bus de direcciones de la familia de procesadores MSP430 son de 16 bits. El MSP430F5438A pertenece a la familia MSP430X donde el bus de direcciones es de 20 bits y los registros tienen algunos 16 y otros 20 bits.

El código de MSP430 es compatible con MSP430X pero no a la inversa. El MSP430X puede utilizarse en modo MSP430, pero pierde el acceso a la memoria ubicada por encima de la dirección 0xFFFF.

En la figura 7 se muestran los periféricos integrados. El Módulo de Control Principal usa todos ellos excepto DMA, algunos puertos y uno de los temporizadores.

3.3.1 Características generales del microcontrolador

El siguiente esquema presenta las características generales del microcontrolador.

- Alimentación de 1.8 V a 3.6 V
- Muy bajo consumo de energía

- Activación desde modo “Standby” en menos de 5 µs
- 256 Kbyte de memoria Flash
- 16 Kbyte de memoria SRAM
- Arquitectura RISC de 16-Bits
- Memoria extendida de 1 MB
- Clock del sistema hasta 25 MHz
- Sistema de gestión de energía flexible.
    - LDO totalmente integrado con voltaje de alimentación del núcleo programable.
    - Supervisión de voltaje de alimentación, monitoreo y apagado,
- Sistema unificado de relojes
    - Lazo de control para estabilización de frecuencia
    - Generador de reloj interno de baja frecuencia y bajo consumo (VLO).
    - Generador interno de referencia de baja frecuencia (REFO)
    - Capacidad de usar cristales de alta frecuencia de hasta 32 MHz
- Un Temporizador de 16 bits (TA0) con 5 registros de comparación y captura
- Un Temporizador de 16 bits (TA1) con 3 registros de comparación y captura
- Un Temporizador de 16 bits (TB0) con 7 registros de comparación y captura
- Cuatro Interfaces de comunicación serial configurables como
    - UART mejorada con autodetección de velocidad
    - Codificador y decodificador IrDA
    - SPI Sincrónico
    - I^2C
- Conversor Análogo digital de 12 bits (ADC)
    - Referencia interna
    - Muestreo y retención
    - 14 canales externos y 2 canales internos
- Multiplicador por hardware para operaciones de 32 bits
- Hardware integrado para cálculo de CRC de 16 bits
- Programación serie integrada sin necesidad de voltaje externo.
- 3 canales internos de DMA
- Temporizador básico con funcionalidad para RTC

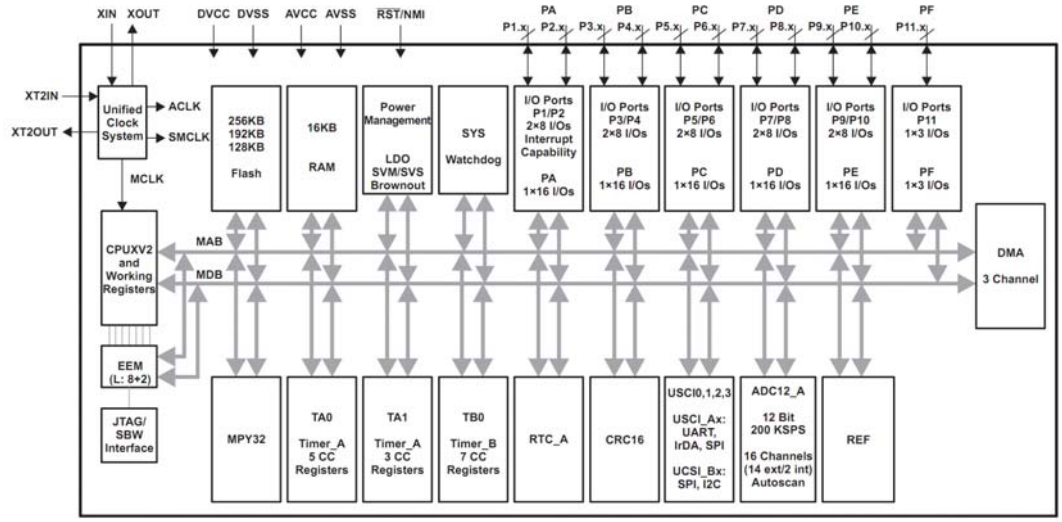

Figura 7 – Diagrama funcional del MSP430F5438A

3.3.2 Modos de operación del microcontrolador

3.3.2.1 Modo activo (AM)

- Todos los relojes del sistema activados
- Consumo: 230 µA/MHz a 8 MHz, 3.0 V, con programa ejecutando desde memoria Flash
- Consumo: 110 µA/MHz a 8 MHz, 3.0 V, con programa ejecutando desde memoria RAM

3.3.2.2 Modo "Standby" (LPM3)

- Watchdog
- Supervisor operacional
- Retención completa de RAM
- Restauración rápida
- Consumo: 1.7 µA a 2.2 V, 2.1 µA a 3.0 V usando reloj de tiempo real con cristal externo.
- Consumo: 1.2 µA a 3.0 V usando "Low-Power Oscillator" (VLO),

3.3.2.3 Modo apagado (LPM4)

- Supervisor operacional
- Retención completa de RAM
- Restauración rápida
- Consumo: 1.2 µA a 3.0 V

3.3.2.4 Modo “Shutdown” (LPM4.5)

- Consumo: 0.1 µA a 3.0 V

3.3.3 Memoria del microcontrolador

A partir de pruebas efectuadas se determinó que el procesador almacena los datos usando la estrategia little-endian.

No se encontraron referencias concretas en la documentación del fabricante del microcontrolador que aseguren que este es el formato utilizado en todos los casos.

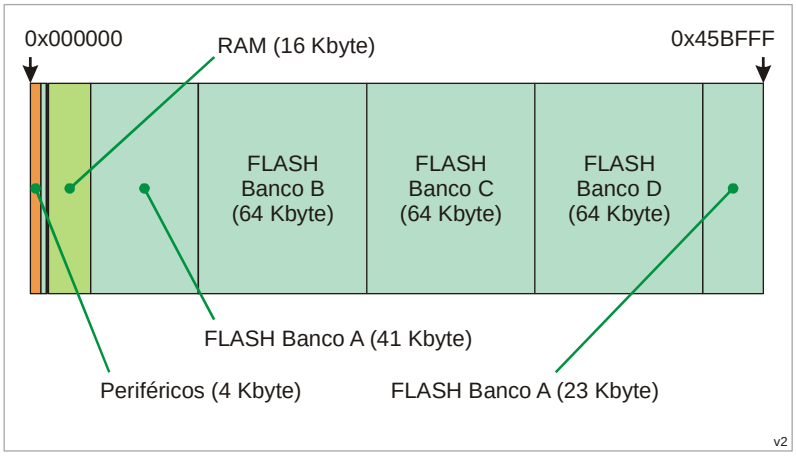

Figura 8 – Memoria del microcontrolador

La tabla 1 muestra la distribución de memoria del microcontrolador. En la figura 8 se presenta un diagrama de la distribución.

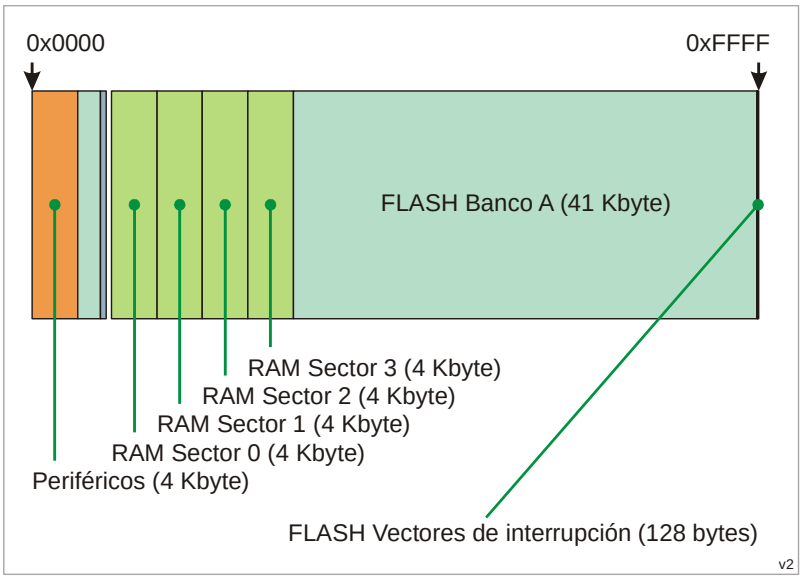

Figura 9 – Espacio de memoria de direccionamiento estándar (64 Kbyte)

Tabla 1 – Distribución de la memoria del MSP430F5438A

Tipo	Identificación	Tamaño	Desde	Hasta
Periféricos		4 KB	0x00 0000	0x00 0FFF
Bootstrap loader (BSL) (FLASH)	BSL 0	512 B	0x00 1000	0x00 11FF
	BSL 1	512 B	0x00 1200	0x00 13FF
	BSL 2	512 B	0x00 1400	0x00 15FF
	BSL 3	512 B	0x00 1600	0x00 17FF
Memoria de información (FLASH)	Info D	128 B	0x00 1800	0x00 187F
	Info C	128 B	0x00 1880	0x00 18FF
	Info B	128 B	0x00 1900	0x00 197F
	Info A	128 B	0x00 1980	0x00 19FF
Libre	Hole	512 B	0x00 1A00	0x001BFF
RAM	Sector 0	4 KB	0x00 1C00	0x00 2BFF
	Sector 1	4 KB	0x00 2C00	0x00 3BFF
	Sector 2	4 KB	0x00 3C00	0x00 4BFF
	Sector 3	4 KB	0x00 4C00	0x00 5BFF
FLASH	Banco A	41 KB	0x00 5C00	0x00 FF7F
Vectores de interrupción		128 B	0x00 FF80	0x00 FFFF
FLASH (direccionamiento extendido)	Banco B	64 KB	0x01 0000	0x01 FFFF
	Banco C	64 KB	0x02 0000	0x02 FFFF
	Banco D	64 KB	0x03 0000	0x03 FFFF
	Banco A	23 KB	0x04 0000	0x04 5BFF

3.3.4 Memoria flash

La memoria flash puede ser programada a través del puerto JTAG, usando Spy-Bi-Wire (SBW), el BSL, o por la CPU.

La CPU puede realizar escrituras en la memoria flash de un byte, una palabra, o una palabra doble (32 bits).

Características de la memoria flash incluidas en el microcontrolador:

- Existen 512 segmentos de memoria principal, cada uno de 512 bytes, y cuatro segmentos de "memoria de información" (A a D) de 128 bytes cada uno.
- Los segmentos 0 a 511 pueden ser borrados en un paso, y cada segmento puede ser borrado individualmente.
- Los bloques A, B, C y D deben ser borrados individualmente.
- El bloque A puede bloquearse por separado.

El borrado de los segmentos de memoria flash puede realizarse con código ejecutándose desde memoria RAM o desde otro banco de memoria FLASH, pero nunca desde el mismo banco.

La mínima unidad que puede borrarse es un segmento (512 bytes). Cuando un segmento se borra todos los bits pasan a valor 1. Luego, el proceso de escritura es equivalente a poner algunos bits en 0. Esta escritura no se puede revertir con otra, sólo con el proceso de borrado.

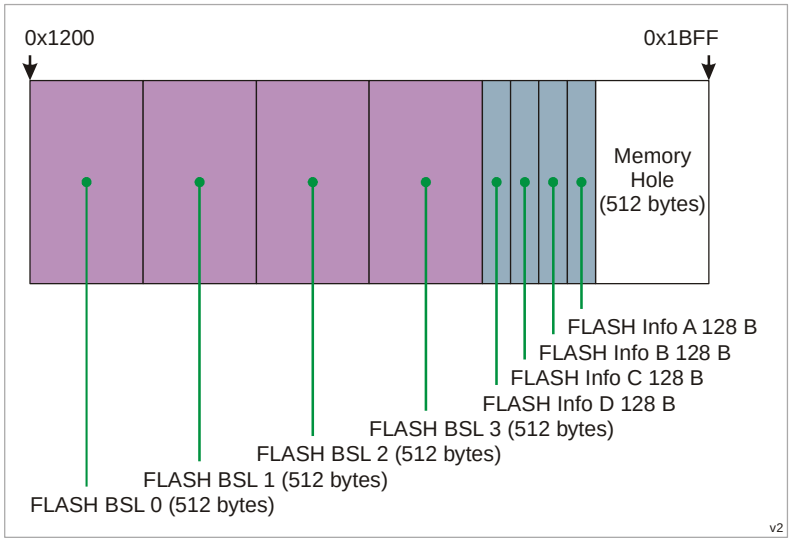

Figura 10 – Memoria de control

La cantidad de ciclos de borrado es limitada y depende de la temperatura de operación. Su cota inferior es del orden de 10.000 ciclos.

3.4 El bus I²C

El protocolo I²C –por "Inter-IC"– fue diseñado por Philips Semiconductors (ahora NXP Semiconductors).

Se trata de un protocolo bidireccional half duplex con un bus de dos hilos, reloj (SCL) y datos (SDA), ambos referidos a tierra.

Este bus serie es capaz de transferir datos en grupos de 8 bits con una velocidad de hasta 100 kbit/s en modo estándar y hasta 400 kbit/s en modo "Fast".

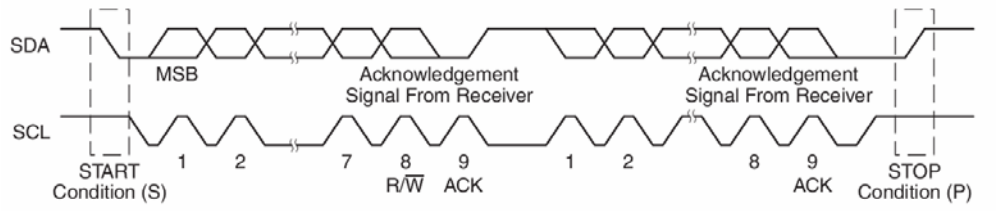

Figura 11 – La comunicación I²C

Existen versiones extendidas del protocolo que permiten velocidades de hasta 1 Mbit/s en modo “Fast-Plus”, hasta 3.4 Mbit/s en modo “High-speed” y hasta 5 Mbit/s en modo “Ultra Fast” pero no son soportadas por los dispositivos utilizados en el proyecto.

Las líneas del bus usan un “pull-up” a Vcc por lo que su estado normal es alto. Los dispositivos controlan las líneas llevándolas a tierra.

Los dispositivos acoplados al bus pueden asumir roles de maestro o esclavo. Existe una modalidad especial conocida como multi maestro, en la que más de un dispositivo puede arbitrar el bus. Esta modalidad es la utilizada en el bus principal del satélite.

3.4.1 Direccionamiento

Cada uno de los dispositivos conectados al bus tiene una dirección única. Estas direcciones pueden ser de 7 o 10 bits.

Cuando un dispositivo requiere comunicarse con otro, genera una “START CONDITION” y escribe la dirección del destinatario en el bus usando la línea SDA para los datos y la línea SCL para secuenciar seguido de la dirección de la comunicación –un bit indicando lectura o escritura–.

Todos los dispositivos en el bus “escuchan” la dirección de destino y la comparan con su propia dirección. Inmediatamente el destinatario controla la línea SDA por un ciclo de SCL para dar el reconocimiento (ACK) como se muestra en la figura 12.

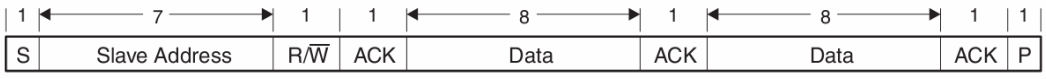

Figura 12 – Direccionamiento I²C de 7 bits

En la figura 13 se muestra cómo se realiza el direccionamiento de 10 bits. Nótese que el primer ACK, puede ser realizado simultáneamente por varios dispositivos.

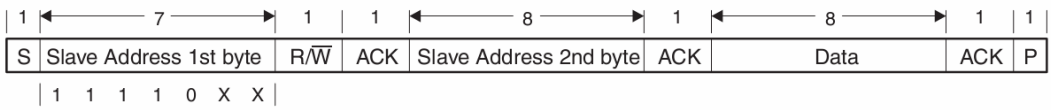

Figura 13 – Direccionamiento I²C de 10 bits

3.4.2 Arbitraje

El bus permite controlar el arbitraje. Cuando un dispositivo “escribe” en el bus, también lee el resultado. Si lo que lee no coincide con lo que escribe lo considera una pérdida de arbitraje y cancela la comunicación en curso.

Esta es una colisión típica de la modalidad multi-maestro. Si dos dispositivos intentan escribir en el bus, uno de ellos perderá el arbitraje.

3.4.3 Sincronización de relojes

El dispositivo que actúa como maestro controla la línea de reloj, pero el esclavo puede sostenerla (en cero) para frenar al maestro. Observamos este fenómeno durante el desarrollo utilizando un analizador lógico cuando se generaba una demora entre la llegada de un byte y la lectura del dato por parte de la ISR asociada. Gracias a este mecanismo, los dispositivos pueden sincronizarse aunque sus fuentes de reloj sean diferentes.

3.4.4 Lecturas y escrituras

El bit luego de la señalización indica si el maestro entregará datos o los obtendrá, es decir: determina cuál de los dispositivos controlará la línea SDA.

Luego de la transferencia de cada byte, la contraparte deberá generar el reconocimiento (ACK). En caso contrario, el dispositivo que escribe los datos generará una situación de no reconocimiento (NACK).

El protocolo permite que el maestro cambie la dirección de la comunicación y hasta eventualmente la contraparte. Esto se logra enviando una nueva señalización "START CONDITION" como se muestra en la figura 14. El cambio puede realizarse tantas veces como el maestro lo requiera.

Para finalizar la comunicación, el maestro genera una señalización "STOP CONDITION".

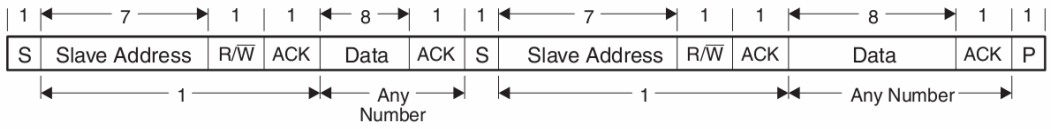

Figura 14 – Datos en una comunicación I²C

3.4.5 El periférico I²C del microcontrolador

Este periférico es capaz de generar una interrupción y proveer los elementos para que la ISR pueda detectar cuál fue la causa. Las posibles fuentes de interrupción son:

- Start condition
- Tx buffer ready
- RX data
- Arbitration lost
- Not acknowledgement
- Stop condition

La implementación de I²C en el MSP430F5438A tiene varias particularidades. Una de las más relevantes es el conjunto de fallas conocidas reportadas en el documento de Erratas del microcontrolador [26].

Estas fallas se muestran en la tabla 2 y a continuación se describe cada una de ellas.

Tabla 2 – Problemas del periférico que realiza la comunicación I²C

Errata	Revisión del microcontrolador				
	H	G	F	E	D
USCI26	✓	✓	✓	✓	✓
USCI30		✓		✓	✓
USCI31	✓	✓	✓	✓	✓
USCI35	✓	✓	✓	✓	✓

3.4.5.1 Problema USCI26

En modo multi maestro, el periférico no garantiza que se cumpla el tiempo definido en la especificación de I²C entre la señal STOP CONDITION y la siguiente START CONDITION.

3.4.5.2 Problema USCI30

Cuando el periférico I2C está configurado en modo receptor (maestro o esclavo), utiliza dos registros para almacenar los datos recibidos.

A medida que recibe los bits, estos son almacenados en un registro de desplazamiento. Cuando se completa un byte, se genera el reconocimiento en el bus, los datos se copian al registro de lectura y se genera la interrupción –si es que está configurada–.

Los siguientes bits recibidos se van almacenando en el registro de desplazamiento.

Si los datos del registro de lectura no son leídos cuando llega el séptimo bit del siguiente byte, se produce un error y el periférico puede quedar bloqueado.

3.4.5.3 Problema USCI31

Si se configura el periférico mientras este está recibiendo un byte, el error se reporta al transmisor cuando se reciba en el próximo byte.

3.4.5.4 Problema USCI35

Cuando el periférico I²C opera modo maestro con una frecuencia de reloj superior a 50KHz, no se respetan los tiempos de las señales en las "REPEATED START CONDITIONS" (cambio de sentido de la comunicación) por lo que el esclavo recibirá datos incorrectos.

3.5 Protocolo AX25

AX25 es un protocolo de capa de enlace utilizado por radioaficionados. Está en su versión 2.2 desde 1998 [3].

El protocolo propone tres tipos de tramas: supervisión (S), sin numerar (U) e información (I). Sus estructuras son las siguientes:

Frames AX25 de tipo S y U

Flag	Address	Control	Info.	FCS	Flag
01111110	112/224 Bits	8/16 Bits	N*8 Bits	16 Bits	01111110

Frames AX25 de tipo I

Flag	Address	Control	PID	Info.	FCS	Flag
01111110	112/224 Bits	8/16 Bits	8 Bits	N*8 Bits	16 Bits	01111110

Para determinar el inicio y fin de las tramas se utilizan secuencias que contienen seis unos. Esto hace que sea necesario insertar ceros en el contenido de la trama cada vez que aparece esta secuencia. Este proceso se conoce como “Bit stuffing”.

Los nodos AX25 utilizan un mecanismo de CRC para verificar errores en las tramas. El campo FCS contiene un CRC de 16 bits de los datos calculado de acuerdo a las definiciones del protocolo HDLC, del que hereda la técnica.

Los datos pueden tener un largo de hasta 256 bytes.

Las direcciones AX25 son cada una de siete bytes. Los seis primeros componen un nombre construido con letras mayúsculas y números del código ASCII. Los nombres se completan con espacios.

El séptimo byte aloja en los bits 1 a 6, un entero identificador de la estación definido como SSID. Dicho identificador es utilizado cuando existen varias estaciones asociadas al mismo nombre.

Los caracteres del nombre se representan desplazados un bit a la izquierda.

La cabecera de las tramas AX25, puede tener entre dos y cuatro direcciones.

La primera dirección es siempre la de destino de la trama, la segunda es la de origen. Las siguientes direcciones se utilizan, cuando están presentes, para determinar las estaciones repetidoras de la ruta AX25 por las que debe encaminarse la trama.

El bit de menor peso (0) del séptimo byte, se utiliza para indicar con valor 1, que la dirección es la última en la cabecera AX25. El bit de mayor peso (7) del séptimo byte, se usa para indicar que la repetidora asociada a la dirección ya encaminó el mensaje.

Estas direcciones de enrutamiento, desde la tercera a la última, se utilizan en orden.

3.6 Algoritmo de cifrado TEA

TEA –del inglés Tiny Encryption Algorithm- es un algoritmo para el cifrado de bloque de sencilla implementación. Fue diseñado por David Wheeler y Roger Needham del Cambridge Computer Laboratory, y presentado por vez primera en 1994 en el Fast Software Encryption Workshop (Wheeler y Needham, 1994) [14].

El algoritmo trabaja sobre un conjunto de 64 bits y utiliza una clave de 128 bits, pero cada una de las posibles claves es equivalente a otras tres. A continuación se presenta el código original para cifrado y descifrado.

```
/* Routine, written in the C language, for encoding with key k[0] -
k[3]. Data in v[0] and v[1]. */

void code(long* v, long* k) {
     unsigned long y=v[O] ,z=v[l], sum=O, /* set up */
     delta=OxgeS779b9, /* a key schedule constant */
     n=32 ;
     while (n-->O) { /* basic cycle start */
          sum += delta ;
          y += ((z<<4)+k[0]) " (z+sum) ^ ((z>>5)+k[1]) ;
          z += ((y<<4)+k[2]) " (y+sum) " ((y>>5)+k[3]) ;
     } /* end cycle */
     v[O]=y ; v[1]=z ; }


void decode(long* v,long* k) {
     unsigned long n=32, sum, y=v[O], z=v[l],
     delta=Oxge3779b9 ;
     sum=delta<<S ;
     /* start cycle */
     while (n-->O) {
          z-= ((y<<4)+k[2]) ~ (y+sum) ~ ((y>>S)+k[3]) ;
          y-= ((z<<4)+k[O]) ~ (z+sum) ~ ((z>>S)+k[1]) ;
          sure-:delta ; }
     /* end cycle */
     v[0]:y ; v[1]=z ;}
```

El número de rondas –el valor de “n”– es variable y hay implementaciones que usan 64 en lugar de 32.

3.7 Parámetros orbitales

Los parámetros orbitales se publican y comunican utilizando el formato TLE (NORAD Two-Line Element) [12].

Los datos para cada satélite consisten en tres líneas con el siguiente formato:

```
AAAAAAAAAAAAAAAAAAAAAAA
1 NNNNNU NNNNNAAA NNNNN.NNNNNNNN +.NNNNNNNN +NNNNN-N +NNNNN-N N
NNNNN
2 NNNNN NNN.NNNN NNN.NNNN NNNNNNN NNN.NNNN NNN.NNNN
NN.NNNNNNNNNNNNN
```

La línea 0 tiene 24 caracteres de largo y contiene el nombre del satélite. Los contenidos de las líneas 1 y 2 se detallan en la tabla 3 y la tabla 4 respectivamente.

Nótese que las ubicaciones están definidas por el número de columna que inicia en 1. Se ha mantenido esta notación que coincide con la publicación. Las posiciones se obtienen restando 1 al número de columna.

Ejemplo:

```
NOAA 14
1 23455U 94089A 97320.90946019 .00000140 00000-0 10191-3 0 2621
2 23455 99.0090 272.6745 0008546 223.1686 136.8816 14.11711747148495
```

Tabla 3 – TLE Contenido de la línea 1

Columna	Descripción
01	Número de la línea del TLE
03-07	Número del satélite
08	Clasificación (U = No clasificado)
10-11	Designación Internacional (últimos 2 dígitos del año de lanzamiento)
12-14	Designación Internacional (número de lanzamiento del año)
15-17	Designación Internacional (número de parte del lanzamiento)
19-20	Año de Epoch (últimos dos dígitos del año)
21-32	Epoch (día del año y fracción del día)
34-43	Derivada primera del Movimiento Medio
45-52	Derivada segunda del Movimiento Medio
54-61	Término BSTAR
63	Tipo de ephemeris
65-68	Número de elemento
69	Checksum (módulo 10)

Tabla 4 – TLE Contenido de la línea 2

Columna	Descripción
01	Número de la línea del TLE
03-07	Número del satélite
09-16	Inclinación (grados)
18-25	Ascensión recta del nodo ascendente (grados)
27-33	Excentricidad
35-42	Argumento de Perigeo (grados)
44-51	Anomalía Media (grados)
53-63	Movimiento Medio (Revoluciones por día)
64-68	Número de revolución en el Epoch (revs)
69	Checksum (módulo 10)

3.8 Proyectos relacionados

3.8.1 Módulo de control Principal

En la figura 15 se observa un esquema del software requerido para el Módulo de Control Principal al finalizar el desarrollo del prototipo del hardware [2] (agosto de 2012), momento en el que comienza la etapa de diseño del presente trabajo.

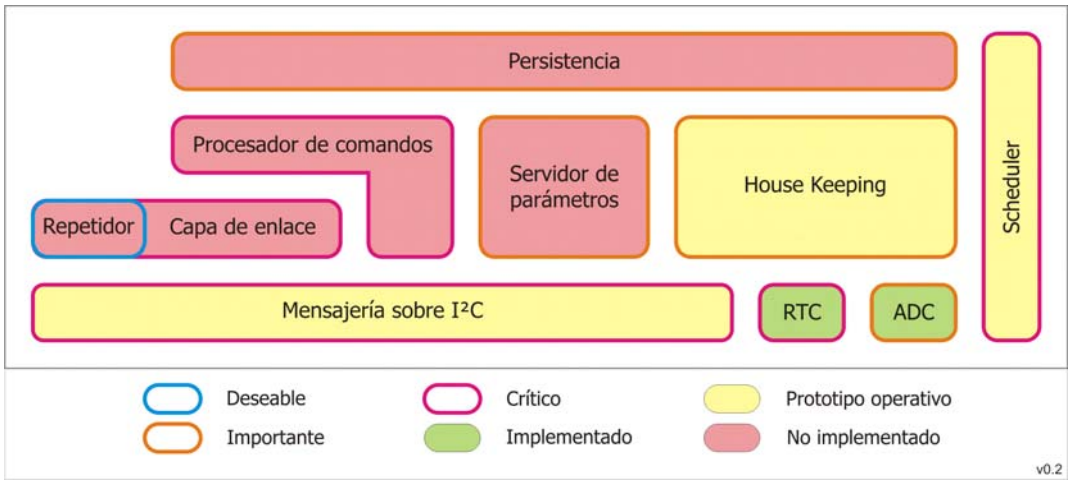

Figura 15 – Situación del MPC en agosto de 2012

Dicho prototipo incluye una placa en la que se instalarán dos microcontroladores MSP430F5438A, uno para implementar el Módulo de Control Principal y otro para implementar el módulo de Detección y Control de Actitud.

Ese diseño de hardware está acompañado por algunas piezas de software. Entre ellas se destaca un despachador que maneja una cola de tareas algunas asíncronas y otras periódicas.

Los restantes módulos implementados incluyen: la configuración de uno de los temporizadores del microcontrolador para mantener la hora, la adquisición de la temperatura del microcontrolador a través del conversor analógico - digital (ADC), un protocolo básico para transferencia de datos sobre I²C y el proceso de House Keeping.

La implementación del prototipo utiliza un modelo de datos restringido que permite acceder a los primeros 64KBytes de memoria (direccionamiento de 16 bits).

3.8.2 Otros Cubesat

El centro espacial de la Escuela Politécnica Federal de Lausanne, Suiza desarrolló desde 2005 y puso en órbita en setiembre de 2009 el satélite SwissCube, un proyecto basado en el estándar CubeSat [7].

Dicho proyecto expone gran cantidad de información con excelente nivel de detalle. Si bien el procesador utilizado es diferente, los módulos definidos son muy similares a los de Antelsat y el bus de comunicaciones interno es también I²C.

La misión tuvo problemas con el sistema de control de actitud pero sus resultados han sido aceptables. Se han logrado cuatro años de operación, comunicación exitosa y varias fotografías tomadas y enviadas.

El documento de presentación del sistema [6] y el que describe el Subsistema de Comandos y Gestión de Datos [19], equivalente al Módulo de Control Principal del Antelsat aportan cantidad de detalles que orientaron el diseño.

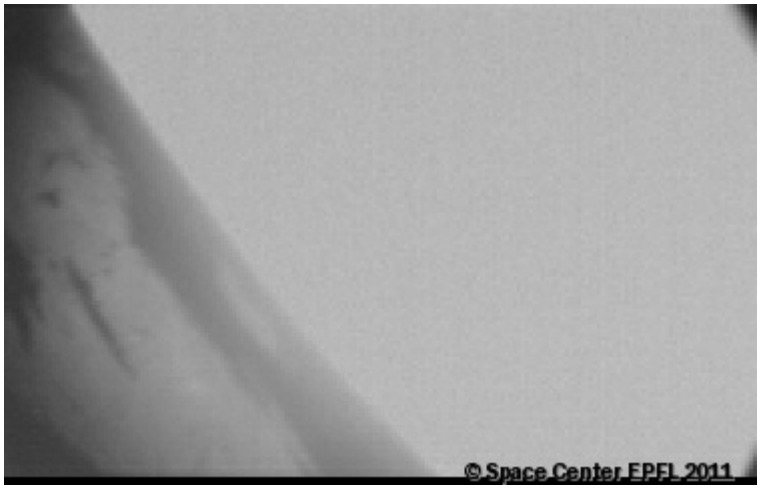

Figura 16 – Fotografía del sur de Suiza tomada por SwissCube

Entre la información de House Keeping del SwissCube, encontramos datos acerca de la temperatura entre las caras y a lo largo del tiempo en el interior del satélite [18]. Estos datos nos indican qué podemos esperar respecto a variación de frecuencia de los cristales y por lo tanto variaciones en el RTC de nuestro sistema.

En el documento de Arquitectura del Software de Vuelo [4], se presentan las generalidades de la comunicación interna sobre I^2C.

Existe muy poca información acerca de otros satélites y ningún software publicado. Esta carencia es consecuencia en gran parte de la legislación de los Estados Unidos, que considera a este tipo de dispositivos dentro de la misma clasificación que las armas.

Capítulo 4.

Diseño

Contenido

4.1 Arquitectura

El software del Módulo de Control Principal se diseñó en cuatro capas:

- Capa de aplicación
- Capa de servicios
- Capa de administración de hardware
- Capa de abstracción del hardware

Las últimas dos que inicialmente eran una se separaron para lograr portabilidad a plataformas similares.

Como se observa en la figura 17, la capa de servicios tiene más componentes que la capa aplicación. Esto se debe a que gran parte de las funcionalidades están orientadas a dar servicio a los restantes módulos.

Estos procesos y servicios se comunican utilizando los recursos provistos por el Kernel.

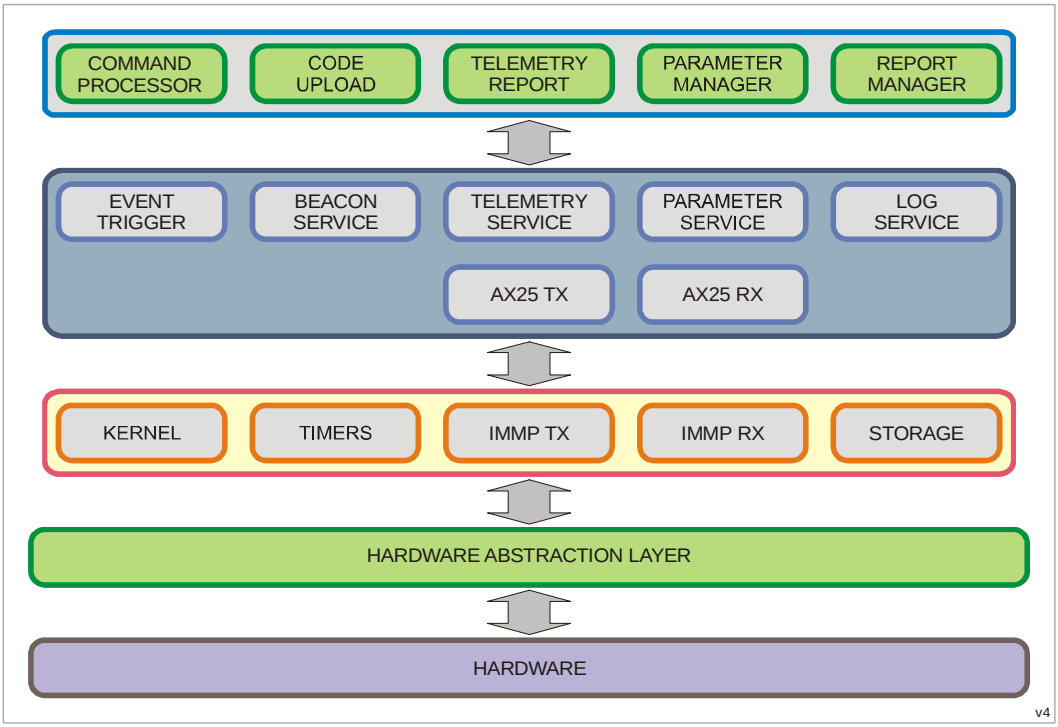

Figura 17 –Arquitectura del Módulo de Control Principal

4.1.1 Componentes de la capa Aplicación

4.1.1.1 Command processor

El procesador de comandos es una aplicación que recibe los telecomandos de tierra, los interpreta y ejecuta.

4.1.1.2 Code upload

Esta aplicación es capaz de recibir mensajes con trozos de código, para colocar en posiciones específicas de la memoria RAM.

Está diseñada para delegar el control y sus recursos a una aplicación cargada, en caso de recibir la instrucción de tierra.

4.1.1.3 Telemetry report

Esta aplicación genera un reporte periódico con los datos de telemetría recolectados de todos los módulos.

4.1.1.4 Parameter manager

Esta aplicación ejecuta los comandos de tierra relativos a los parámetros como la modificación de valores, persistencia y recuperación.

4.1.2 Componentes de la capa de Servicios

4.1.2.1 Event trigger

Este proceso se ocupa de disparar eventos de largo plazo como:

- La restauración del silencio de radio 72 horas luego de habilitado.
- La restauración del estado deseado de los módulos de comunicaciones 72 horas luego de apagados.
- La solicitud de reinicio preventivo a EMS luego de 45 días de ejecución continua
- El apagado del Módulo de Control y Detección de Actitud cuando la hora es insegura.

4.1.2.2 Beacon service

Este proceso se ocupa de reportar periódicamente al Módulo de Gestión de Energía, los datos para construir la baliza Morse.

Estos datos son:

- Los tres bits de menor peso del identificador del último mensaje procesado por el Módulo de Control Principal.
- El estado del servicio de SSTV para terceros.
- El estado del servicio de repetidora AX25 (digipeater)

4.1.2.3 Telemetry service

Este proceso se ocupa de recibir los reportes de telemetría de los restantes módulos y prepararlos para ser enviados convirtiéndolos a formato ASCII hexa-decimal.

4.1.2.4 Parameter service

Este proceso se ocupa de responder los requerimientos de parámetros de los restantes módulos así como de proveer de parámetros al Módulo de Control Principal.

4.1.2.5 Log service

Este proceso se ocupa de almacenar en el registro los reportes de eventos de todos los módulos.

4.1.2.6 AX25 TX

La transmisión AX25 está implementada con dos procesos: AX25 TX y AX25 TX SERVICE.

El proceso AX25 TX SERVICE recibe los mensajes de todos los módulos y los encapsula en tramas AX25 para enviar a tierra.

El proceso AX25 TX retiene los mensajes, y negocia la comunicación con el módulo de comunicaciones COMM2 mientras éste está ocupado.

4.1.2.7 AX25 RX

La recepción AX25 está implementada con dos procesos: AX25 RX y AX25 RX RELAY.

El proceso: AX25 RX recibe los mensajes de tierra desde los módulos de comunicaciones COMM1 y COMM2, los descifra si es necesario, recibe y redirecciona las solicitudes de SSTV, e implementa el servicio de repetidora (DIGIPEATER).

El proceso AX25 RX RELAY retiene los mensajes para otros módulos hasta que estos puedan ser entregados.

4.1.3 Componentes de la capa de Administración del hardware

4.1.3.1 Kernel

Este componente es administrador del recurso procesador. Es la pieza fundamental que posibilita la existencia de procesos y facilita la comunicación entre ellos.

4.1.3.2 Tmers

Este componente provee el acceso a los temporizadores del sistema.

4.1.3.3 IMMP TX

Este servicio se ocupa de transmitir mensajes a otros módulos utilizando el protocolo I²C.

4.1.3.4 IMMP RX

Este servicio retiene los mensajes recibidos, los entrega a los procesos destinatarios cuando éstos están disponibles y descarta los mensajes antiguos de los procesos bloqueados cuando el espacio de almacenamiento disponible está cerca de agotarse.

4.1.3.5 Storage

Este servicio se ocupa de almacenar los eventos y parámetros en memoria FLASH.

4.1.4 Componentes de la capa de Abstracción del hardware

4.1.4.1 Ajuste de Voltaje del procesador

Provee la función SetVCore para ajustar el voltaje de alimentación del núcleo del procesador. La funcionalidad incluye las funciones SetVCoreUp y Set VCore down y fueron extraidas del ejemplo pmm_hal, provisto por el fabricante del microcontrolador.

4.1.4.2 Inicialización de hardware

Realiza la configuración inicial del hardware del microcontrolador. Las operaciones son las siguientes:

- Detiene el watchdog
- Configura todos los puertos no utilizados
- Configura el oscilador basado en el cristal externo para obtener una frecuencia de 32 KHz
- Configura el oscilador interno (DCO)
- Ajusta el voltaje del procesador
- Configura el componente FLL para obtener la frecuencia de reloj de 8 MHz
- Inicializa el temporizador del sistema
  - Temporizador: A0
  - Modo: Continuo
  - Reloj: ACLK (32768 Hz)
  - Divisor de entrada: 8 (f= 4KHz)
  - Deshabilita los comparadores no utilizados del timer TA0
- Configura el temporizador del kernel (generador de tics)
  - Temporizador: A1
  - Modo: Up
  - Reloj: ACLK (32768 Hz)
  - Divisor de entrada: 1
  - Comparador utilizado: CCR0
  - Valor del comparador: 3276 (10 Hz)
  - Deshabilita los comparadores no utilizados del timer TA1
- Deshabilita el temporizador TIMERB no utilizado

4.1.4.3 Bajo consumo de energía

Provee la función low_power_mode para pasar el procesador a modo de bajo consumo de energía. Esta función es utilizada por el kernel cuando no hay ningún proceso en condiciones de recibir el procesador.

4.1.4.4 Configuración del watchdog

Provee la función paramétrica set_watchdog que configura el temporizador watchdog según un parámetro dado.

4.1.4.5 Reinicio del procesador

Provee la función reset que genera un reinicio del procesador de características similares a un reset físico (genera el evento BOR).

4.1.4.6 Temporizador del sistema

Provee las funciones system_timer_start para programar uno de los temporizadores a fin de que interrumpa en un tiempo dado y la función system_timer_stop para deshabilitar la interrupción programada.

El temporizador utiliza un reloj efectivo de 250 µs con una cuenta máxima de 2^{16}, lo que permite configurar tiempos entre 250 µs y algo más de 16 segundos.

4.1.4.7 Real Time Clock

Provee las funciones:

- set_RTC para ajustar la hora del sistema
- read_RTC para obtener la hora (estable), incluyendo el bit que indica hora insegura
- get_time para obtener la hora del sistema estable

4.1.4.8 CRC

Provee las funciones:

- data_crc16 que computa el CRC16 (base 0xFFFF) de un rango de direcciones de memoria con alineación par. Esta función es utilizada por el kernel, para controlar la integridad del stack de las aplicaciones suspendidas.
- code_crc16 que computa el CRC de un bloque de código ubicado en direcciones por debajo de los 64K. Esta función es utilizada por el kernel para realizar reemplazos de código.

4.1.4.9 Memoria FLASH

Provee las funciones:

- erase_flash_segment que borra un segmento de memoria FLASH
- flash_long_copy que copia un conjunto de datos a memoria FLASH

4.1.4.10 Vectores de interrupción

Provee las funciones asociadas a los vectores de interrupción para todas las fuentes no utilizadas específicamente por algún proceso. En particular, incluye el código que genera la temporización del kernel (tics)

4.2 Uso de la memoria

Para poder asegurar los datos críticos utilizando redundancia se maximizó el espacio disponible optimizando el uso de memoria y evitando el uso de memoria dinámica.

Todo el código y las constantes se mantuvieron en el espacio de direccionamiento MSP430 estándar –debajo de los 64 KByte–. Esto permitió que se utilizaran instrucciones más cortas y por lo tanto que se optimizara el tiempo de ejecución,

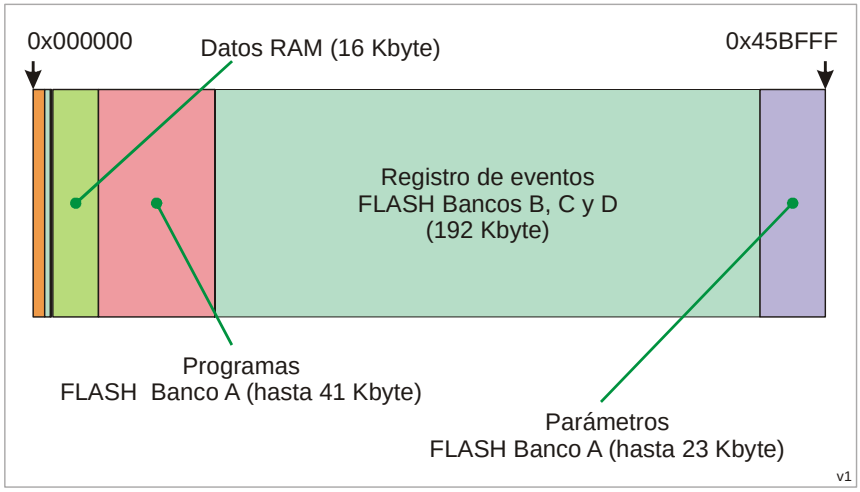

Figura 18 – Mapa de la memoria del Módulo de Control Principal

Los parámetros almacenados se colocaron en la parte alta del banco A de la memoria FLASH. Como esta memoria no puede borrarse o escribirse con instrucciones que estén en el mismo banco [23], el código asociado a estos procesos debió moverse a la RAM [27]. Este fue uno de los últimos cambios en el mapa de uso de memoria para permitir maximizar el espacio de almacenamiento para el registro de eventos.

4.3 Identificación de los módulos

Es necesario contar con una identificación de los módulos en diferentes partes del proyecto, especialmente en lo que tiene que ver con la integración.

La comunicación sobre el bus I²C requiere una dirección de destino de 7 o 10 bits, los registros de eventos requieren un identificador, las estructuras de datos comunes entre módulos requieren un nombre común.

En la tabla 5 se presentan los nombres unificados y los identificadores elegidos. Los símbolos elegidos tienen algunas particularidades:

- La distancia de Hamming entre símbolos es mayor o igual a 3. Esto permite detección y corrección de errores de un bit.
- Los valores requieren siete bits, lo que los hace válidos como dirección I²C
- Los últimos tres bits contienen valores únicos y consecutivos entre 0 y 5, lo que permite utilizarlos para indexación.

Tabla 5 – Identificación de los módulos

#	Nombre	Identificación	Valor	Binario
0	EMS	MODULE_EMS	0x00	000 0000
1	MCS	MODULE_MCS	0x19	001 1001
2	COMM1	MODULE_COMM1	0x2A	010 1010
3	COMM2	MODULE_COMM2	0x33	011 0011
4	ADCS	MODULE_ADCS	0x4C	100 1100
5	PAYLOAD	MODULE_PAYLOAD	0x55	101 0101

4.4 Persistencia ante reinicios

Cada vez que se detecta una corrupción de memoria o una aplicación se bloquea, el microcontrolador se reinicia, ya sea por acción directa del kernel o por la intervención del watchdog.

Cada vez que el Módulo de Control Principal inicia, restaura los parámetros de funcionamiento almacenados en memoria FLASH y busca la hora del último registro para dar continuidad a los eventos.

Cuando el reinicio es planificado se puede almacenar un registro con la hora para recuperarla pero cuando el reinicio es consecuencia de la detección de una corrupción de datos, esto no es posible. Para garantizar la hora se usa como respaldo al Módulo de Gestión de Energía.

En cada reinicio se pierde también el estado de los parámetros activos, pero esto no es una falla. Los parámetros no deberían hacerse permanentes hasta no tener la total seguridad de que son efectivos y no resultan peligrosos para la misión.

4.5 Bajo consumo de energía

El cuidado del balance energético fue considerado desde el inicio del proyecto. El consumo de los microcontroladores es casi despreciable respecto a otros componentes, pero su operación es permanente.

El máximo nivel de ahorro de energía alcanzable es el modo LPM3, en el que sólo la fuente de reloj ACLK permanece activa. No es posible reducir más aun el consumo porque se necesita ACLK para mantener operativo el periférico RTC, el que mantiene la hora del sistema [25].

La implementación realizada suspende hasta LPM0 cuando no hay procesos listos para ser ejecutados. El consumo de energía es algo mayor que LPM3 este modo no suspende el reloj SMCLK (Secondary Master Clock), necesario para mantener el temporizador "System Timer" y la frecuencia de transmisión de I²C.

Las modificaciones necesarias para realizar la implementación de LPM3 en el Módulo de Control Principal no son complejas, pero requieren cambios que deberían reflejarse en los restantes módulos y en las aplicaciones que utilicen una fuente de reloj distinta de ACLK (lo que incluye a las comunicaciones SPI en EMS, COMM1, COMM2 y ADCS)

Como los cambios en todo el sistema resultan bastante importantes y el beneficio obtenido es muy pequeño, se determino usar el modo LPM0 en forma generalizada.

4.6 Ambiente de desarrollo

Las plataformas disponibles para el desarrollo son:

- Code Composer Studio
- IAR Embedded Workbench

Las licencias de uso libre de estas herramientas tienen una limitación de 16 Kbyte en tamaño del código resultante.

Las licencias de IAR Embedded Workbench adquiridas por la Facultad de Ingeniería permiten la operación completa por lo que no existen opciones.

En el proceso de búsqueda de portabilidad y unicidad del entorno de desarrollo se determinó crear una máquina virtual en la que ejecutar la aplicación.

Para el entorno de virtualización se probaron:

- Virtual Box
- VMWare Player

Estas máquinas virtuales se ejecutaron sobre los siguientes sistemas operativos

- Debian 5
- Linux Mint 12
- Ubuntu 11
- Windows XP Profesional 64 bits

- Windows XP Profesional 32 bits

Sobre esas máquinas virtuales se probaron los siguientes sistemas operativos con la aplicación IAR Embedded Workbench:

- Windows 7
- Windows XP Profesional 64 bits
- Windows XP Profesional 32 bits

El que tuvo mejor desempeño fue Windows XP Profesional 32 bits funcionando sobre la máquina virtual VMWare. Este ambiente se utilizó para el desarrollo del Módulo de Control Principal. En cambio, el resto del equipo utilizaba Oracle Virtual Box.

Para construir el ambiente de integración se adoptó la herramienta de Oracle.

4.7 Lineamientos generales

El ambiente hostil y la imposibilidad de intervenir sobre el hardware o software luego de puesto en órbita hacen que se deba considerar un conjunto de medidas para enfrentar los riesgos más probables.

El problema más crítico es la falta de confianza en los dispositivos de memoria. Frente al riesgo de mutación de código, datos y estado de los dispositivos, deben tomarse las siguientes decisiones:

4.7.1 Formato de los datos

Para todas las transferencias de enteros de más de un byte, tanto entre tierra y satélite como entre módulos, se utilizará el formato little-endian.

Motiva esta decisión el que todos los módulos que se conectan al bus I^2C pertenecen a la familia MSP430 y si bien no hay una definición precisa del fabricante que asegure que toda la familia usa el mismo formato, las pruebas indican que es así.

4.7.2 Código desde FLASH

El programa se ejecutará desde memoria FLASH. No se ejecutará código desde memoria RAM salvo en los casos estrictamente necesarios.

Si bien el consumo del microprocesador es mayor cuando ejecuta el programa desde memoria FLASH que cuando lo ejecuta desde memoria RAM, esta última opción requiere construir código basado totalmente en direcciones relativas y trasladarlo desde flash a RAM previo a la ejecución.

Por otra parte, es preferible utilizar la escasa memoria RAM para contar con redundancia de datos.

4.7.3 Programación completa de los dispositivos

No deben asumirse programaciones o estados por defecto. Antes de utilizar un dispositivo, se debe programar completamente su comportamiento.

4.7.4 Duplicación de código

El código puede resultar modificado. En lo posible, el código debe almacenarse por duplicado y deben existir los mecanismos para controlarlo y reemplazarlo.

4.7.5 Seguridad de los datos

Para asegurar los datos se evaluaron distintas opciones. Entre ellas, utilizar redundancia usando controles de paridad, checksum, código Hamming y almacenamiento por triplicado y acceso por voto.

El microcontrolador no provee facilidades para implementar una decodificación Hamming ágil. Si en cambio provee un periférico para calcular CRC16.

Considerando que existe memoria Flash en exceso, las decisiones de diseño al respecto fueron: inicializar los datos por código (asumiendo que este está controlado), almacenar los datos críticos por triplicado y usar un mecanismo de acceso por voto.

4.7.6 ISR para interrupciones no esperadas.

Todas las fuentes de interrupción deben tener un manejador y atenderse aunque la situación no esté contemplada en el diseño o no sea esperada durante la ejecución.

Cuando se genera una interrupción no planificada, deberá considerarse en cada caso, restaurar el estado del dispositivo o reiniciar el sistema.

4.7.7 Interrupciones

Las rutinas de servicio a interrupción no tendrán en lo posible bucles y realizarán el mínimo procesamiento necesario. No se anidarán interrupciones, no se utilizarán NMI generadas por hardware externo, los datos compartidos entre el código principal y las rutinas de servicio a interrupción se excluirán únicamente deshabilitando interrupciones.

Se evitarán llamadas a procedimientos y el uso de funciones de biblioteca con interrupciones deshabilitadas.

4.7.8 Construcción del código

No se utilizará recurrencia, memoria alojada dinámicamente ni se utilizarán funciones de bibliotecas cuyo código sea desconocido.

Capítulo 5.

Kernel

Contenido

5.1 Motivación

En la sección Arquitectura se mencionan los procesos que implementan las aplicaciones y servicios del Módulo de Control Principal. Estos no podrían existir como tales sin un soporte que provea el aislamiento y la comunicación necesaria.

Los constructores del prototipo del módulo determinaron necesario utilizar una cola de tareas cuando los procesos estaban limitados a "house keeping" usando I2C, lectura de la temperatura interna, y RTC.

Una de las primeras actividades del proyecto fue el modelado de los procesos básicos y su interacción utilizando los mecanismos desarrollados hasta ese momento. A partir de ese modelo se determinó que la verificación resultaría imposible puesto que la gran cantidad de tareas necesarias ejecutándose en un contexto homogéneo, definían infinidad de flujos posibles de ejecución. Esto determinó la necesidad de investigar la posibilidad de usar un sistema operativo.

En la siguiente etapa del desarrollo se analizaron los sistemas operativos disponibles para el microcontrolador. Entre ellos se destacaron “Real-time operating system SYS/BIOS” desarrollado por Texas Instruments –el fabricante del microcontrolador–, “µCOS/II Kernel”, desarrollado por Micrium Inc. y “FreeRTOS operating system”, desarrollado por Real Time Engineers.

Paralelamente se investigaron las posibilidades para solucionar las posibles mutaciones del contenido de la RAM y los registros del procesador y sus periféricos. El hardware no provee herramientas para subsanar este problema, ni facilidades para usar eficientemente código de Hamming o paridad, por lo que las soluciones más equilibradas resultaron ser el control de CRC y el utilizar múltiples copias de los datos críticos.

Las características de la misión y el escaso espacio de memoria RAM disponible en el microcontrolador fueron entonces los elementos fundamentales para determinar utilizar un Kernel para administrar el recurso “procesador” en lugar de un sistema operativo completo.

5.2 Construir en lugar de adaptar

Todas las opciones disponibles requerían modificaciones importantes para contemplar los requerimientos de memoria, bajo consumo de energía y especialmente para proveer el comportamiento deseado ante alteraciones de datos.

En ese momento se hizo necesario tomar una de las grandes decisiones del proyecto. Las opciones eran, o bien estudiar a fondo los sistemas existentes para luego adaptarlos a las necesidades de la misión, o bien construir un kernel específico.

Para estimar los tiempos necesarios se construyó un pequeño prototipo de kernel basado en el núcleo de µCOS y se comenzó la investigación detallada de FreeRTOS.

De esa estimación se determinó que construir una pieza específica resultaría mucho más adecuado a las necesidades.

5.3 Un kernel colaborativo

El procesador tiene 16 registros, uno de los cuales es un generador de constantes. El contexto de ejecución es por lo tanto de 15 registros entre los que se encuentran el registro de estado, el contador de programa y el puntero del stack. Los doce registros de propósito general tienen 20 bits y almacenar cada uno de ellos requiere 4 bytes de memoria.

El microcontrolador cuenta con un periférico –watchdog– que puede generar una interrupción no enmascarable temporizada. Esta puede ser utilizada para realizar la expropiación del procesador.

El microcontrolador no provee protección de memoria ni de acceso a periféricos.

Cuando comenzó el diseño del kernel para el Módulo de Control Principal, ya se estaba considerando en el equipo de trabajo el reutilizarlo para alguno de los restantes módulos.

Era necesario decidir si utilizar un kernel expropiativo o uno colaborativo. Los requerimientos de los módulos de comunicaciones en los que en ese entonces el tiempo del procesador resultaba apenas suficiente, sumado al requisito de simplicidad, determinaron la decisión. El kernel sería colaborativo.

Por otra parte, esta configuración generaba mucho menos resistencia al cambio en los integrantes del equipo, en su mayoría con formación en electrónica y poca confianza en los procesos de software complejos.

La primera versión del kernel era capaz de ejecutar procesos independientes con el único requisito de ceder el procesador antes que expirara la temporización del watchdog y las prioridades estaban definidas en tiempo de carga. Estas condiciones prácticamente no cambiaron en las sucesivas versiones y revisiones.

La modalidad colaborativa genera escaso "overhead" porque almacena el contexto de ejecución sólo cuando el proceso se bloquea a la espera de algún recurso o evento.

Esto optimiza el uso del procesador pero determina que las operaciones críticas respecto al tiempo deban ejecutarse en procesos interruptivos.

5.4 Prioridad de los procesos

El kernel organiza los procesos en una lista. La prioridad de cada uno está definida por su posición en la misma, y esta se define en el momento de carga. Los primeros en cargarse ocupan las primeras posiciones y son los que tienen mayor prioridad. Si bien esta prioridad es fija, los procesos pueden relegarla temporalmente.

Existe un objeto llamado IDLE. Conceptualmente es un recurso que está disponible cuando ningún proceso está listo para ejecutarse. Cuando un proceso espera por el recurso IDLE, permanecerá bloqueado hasta que ningún otro proceso requiera el procesador. Esta acción es equivalente a relegar la prioridad por debajo de la menor posible. Esta espera puede ser permanente o temporizada.

Cuando más de un proceso espera por el recurso IDLE, estos se colocan en una cola. El primero en ingresar es el primero en salir salvo que la espera sea temporizada.

5.5 Temporización

5.5.1 Tiempo de espera

La unidad de temporización del kernel es el TIC. La implementación debe incrementar una variable del kernel con una frecuencia constante.

Cada proceso tiene un contador. Cuando un proceso está bloqueado a la espera de un recurso y la espera es temporizada, ese contador tiene la cantidad de TICs que restan para que la espera finalice.

Cada vez que un proceso se bloquea, el despachador del kernel toma el control y actualiza los contadores de todos los procesos bloqueados temporizados. Cuando un contador llega a cero, el proceso asociado sale del estado de espera y pasa al estado "READY" y el resultado de la operación se define como TIMEOUT.

Para la misión, el período de temporización del kernel se fijó en 100 ms. Este valor resultó adecuado pero vale aclarar que es muy diferente a los valores habitualmente utilizados en RTOS que suelen estar en el orden de 1 ms.

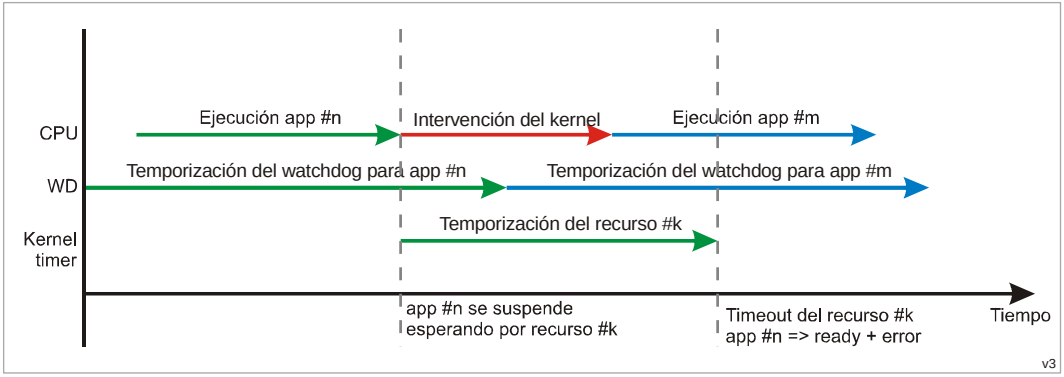

Figura 19 – Tiempos de ejecución

5.5.2 Tiempo de ejecución

Para que el sistema pueda operar correctamente en un contexto colaborativo, los procesos no pueden retener permanentemente el procesador.

En la operación de carga de cada proceso se define la configuración del temporizador watchdog que se utilizará cada vez que se entregue el procesador al proceso. Esa configuración debe ser adecuada al tiempo máximo de ejecución requerido en el peor caso puesto que, si la cuenta expira, se considerará como una situación anómala y se generará un reinicio del microcontrolador.

La intervención del kernel, aunque resulta muy breve, está incluida dentro del tiempo de la ejecución. Una vez que el kernel determina a cuál de los procesos asignará el procesador, configura el watchdog usando el nuevo valor.

El tiempo requerido para ejecutar los servicios de las interrupciones también debe ser contemplado en la configuración.

5.6 Comunicación y sincronización de procesos

Además de la funcionalidad básica de administración del procesador, este kernel provee los mecanismos necesarios para comunicar y sincronizar a los procesos.

El kernel administra entidades abstractas llamadas recursos. Los procesos pueden tomar posesión de estos recursos convirtiéndose en propietarios de estos. El proceso propietario de un recurso puede establecer un espacio de transferencia asociado al mismo y recibir datos a través de él, bloquearse hasta que se le entreguen datos o hasta obtener una señalización o liberarlo.

Todos los procesos pueden, a su vez, entregar datos a un recurso y bloquearse hasta que un recurso este libre tanto para entregar datos como para tomar propiedad y preguntar por el estado de un recurso.

Estos recursos pueden usarse para administrar el acceso al hardware, comunicar procesos o señalizar. Las operaciones de bloqueo, lectura y escritura son bloqueantes y pueden cancelarse por tiempo. Se provee además una función especial para que los servicios de interrupciones entreguen datos y/o liberen procesos bloqueados.

5.7 Bajo consumo de energía

Cuando el despachador no encuentra ningún proceso en condiciones de ser ejecutado, pone al procesador en modo de bajo consumo de energía. La interrupción que actualiza el contador de TICs del kernel debe sacar al procesador de este estado así como cualquier otra interrupción que requiera liberar a un proceso bloqueado.

El cambio del estado del procesador se realiza en una función externa al kernel para hacerlo independiente de la arquitectura.

Inmediatamente antes de pasar a modo de bajo consumo de energía, el kernel configura el temporizador watchdog con un valor apropiado. Este valor es obtenido de la configuración y debe ser adecuado al intervalo TIC.

5.8 Arquitecturas soportadas

Cuando se construyó la primera versión del kernel, los únicos módulos cuyo software estaba avanzado eran los de comunicaciones. Estos módulos están implementados sobre un procesador diferente al Módulo de Control Principal y tienen requerimientos diferentes.

La primera versión del kernel soportaba únicamente al procesador y modo de ejecución del Módulo de Control Principal.

La segunda versión soportaba sólo el modo de direccionamiento extendido –utilizado únicamente por el Módulo de Control Principal– pero resultaba compatible con el procesador utilizado en los módulos de comunicaciones.

A partir de la versión 5, el kernel es compatible con todos los procesadores y modos de direccionamiento utilizados en la misión y está construido de forma tal que le permite adaptarse a otros procesadores de la familia MSP430.

5.9 Otros detalles

Los estados de los procesos, el algoritmo de planificación, los recursos, la interfaz con los programas, el uso de la pila y la configuración se describen en el Manual del Usuario del kernel.

Capítulo 6.

Fecha y hora

Contenido

6.1 Mantenimiento de la fecha y la hora

Algunos subsistemas del satélite requieren conocer la fecha y la hora. El módulo de Control de Actitud la necesita para extrapolar la última órbita conocida cada vez que debe estimar la posición. El módulo de Control Principal la requiere para registrar los eventos. La carga científica la requiere para agregarla a las imágenes tomadas para precisar la ubicación.

El microcontrolador tiene un temporizador llamado RTC –"Real Time Clock"– capaz de mantener la hora en distintos formatos. Este temporizador necesita un oscilador muy estable y alimentación permanente. Las opciones de hardware van desde un oscilador interno hasta alimentación externa y un cristal independiente. El diseño electrónico determinó utilizar en el módulo de control principal un cristal de 32.768 Hz para propósito general. A partir de este cristal se genera la señal que mantiene al RTC.

Otra decisión de diseño relevante es la de no incluir alimentación externa para el RTC. Esto implica que cuando el microcontrolador se reinicia, se perderá la hora del sistema. Para solucionar este inconveniente, se agregó al Módulo de Gestión de energía, un cristal de las mismas características a fin de tener un temporizador de respaldo y se diseñó un protocolo para determinar la hora luego de un reinicio.

Cuando el Módulo de Control Principal inicia, intenta obtener la hora de entre los registros de eventos. Si esto no es posible, establecerá la hora por defecto. Luego solicita la hora a EMS y decide cuál es la más precisa. De ser necesario, informa a EMS de la nueva hora que debe adoptar.

6.2 Desviación de la hora

Las temperaturas en el ambiente del satélite hacen que la frecuencia obtenida del cristal tenga desviaciones y por lo tanto, que la hora pierda precisión. Para solucionar este problema, las estaciones terrenas deben enviar en forma periódica actualizaciones de la hora al satélite.

Una excesiva desviación en la hora puede provocar que la posición extrapolada por el algoritmo de control de actitud sea incorrecta, por lo que, si la hora no es actualizada en un período de tiempo predefinido, se considerará insegura.

Este período de tiempo tiene un valor inicial de siete días. Este valor fue determinado por el proyecto de Detección de Actitud [1] y puede ser corregido desde tierra en función de las desviaciones obtenidas cada vez que se actualiza la hora.

6.3 Representación de la hora

La necesidad de revisar el formato de la hora aparece durante el diseño de los registros a realizar en memoria flash. El espacio de almacenamiento no es escaso, pero el uso descontrolado genera limitaciones innecesarias.

Los eventos astronómicos requieren una medida de tiempo unificada y utilizan habitualmente la fecha Juliana: JDN (Julian Day Number). El día Juliano número 0 se asigna al iniciado al mediodía, hora de Greenwich, del 1 de enero de 4713 AC del calendario Juliano Proléptico. La hora se expresa como fracciones de día [9].

Los algoritmos que utiliza el módulo de Control de Actitud requieren la medida de tiempo en este formato. La precisión requerida por estos algoritmos es del orden de un segundo.

La fecha juliana actual es de aproximadamente 2,5 x 10e6 y los segundos están en el orden de 1 x 10e-5. Una variable de punto flotante de 32 bits no tiene precisión suficiente para representar los datos requeridos, una variable de 64 bits permite representar una fecha juliana con 1 milisegundo de precisión (según datos extraídos de IEEE 754-2008 [8]).

Tabla 6 – Precisión de las representaciones de punto flotante

Precisión	Tamaño (bits)	Signo (bits)	Exponente (bits)	Mantisa (bits)	Precisión decimal
Single	32	1	8	23	6-9
Doble	64	1	11	52	15-17

El MCP necesita las marcas de tiempo para el registro de eventos que se almacenan en memoria FLASH.

Cada proceso de escritura con código ejecutado desde memoria FLASH está limitado a 32 bits [27], por lo que sería recomendable que los registros tuvieran este tamaño, para a la vez, minimizar el espacio de almacenamiento.

La representación de la hora UNIX (Unix time), es un entero que cuenta la cantidad de segundos transcurridos desde las 0 horas del 1 de enero de 1970. Este formato tiene precisión suficiente para satisfacer las necesidades y la conversión desde y a JDN son muy sencillas. Ver en tabla 7.

Un entero con signo de 32 bits permite, utilizando este formato, representar marcas de tiempo hasta el año 2038 con precisión de un segundo. A la marca de tiempo se debe agregar el concepto de inseguridad, lo que se puede hacer con el signo.

Tabla 7 – Conversiones entre JDN y UT

Objetivo	Fórmula
UT	(JD − 2440587.5) × 86400
JD	2440587.5 + (UT ÷ 86400)

Por otra parte, el periférico RTC del microcontrolador puede operar tanto en modo calendario como en modo contador de 32 bits.

La hora se representa en todo el sistema con un entero de 32 bits, con el valor de UT y con el bit de mayor peso en 1 cuando su valor sea inseguro.

6.4 Determinación de la hora

Cuando el Módulo de Control Principal (MCS) inicia busca la fecha y hora del registro más reciente entre los eventos y fija esa hora como actual. Además marca su hora como insegura. Si no encuentra registros utiliza el valor de hora por defecto.

Posteriormente solicita la hora al Módulo de Gestión de Energía (EMS). Si recibe una respuesta evalúa:

- Si la hora de EMS no es insegura (EMS no se reinició), ajusta su hora al valor de EMS.
- Si la hora de EMS es insegura y su valor es mayor al valor de MCS, ajusta la hora al valor de EMS.
- Si la hora de EMS es insegura y su valor es menor o igual al valor de MCS, comunica su hora a EMS.

El Módulo de Gestión de Energía sigue la siguiente lógica:

- Cuando inicia establece la hora por defecto.
- Si es interrogado, responde con su hora.
- Si se le indica actualizar la hora, lo hace inmediatamente.

6.5 Actualización de la hora

Cuando un comando de tierra indica actualizar la hora, ésta se comunica inmediatamente al Módulo de Gestión de Energía, y de estar encendido, al modulo de Detección y Control de Actitud.

6.6 Servicio de hora

Luego de negociar la hora con el Módulo de Gestión de Energía, el Módulo de Control Principal queda a la espera de solicitudes de hora de los restantes módulos.

Cuando la hora alcanza la máxima deriva tolerada, el Módulo de Control Principal fuerza el estado deseado del Módulo de Detección y Control de Actitud a apagado.

Capítulo 7.

Parámetros

Contenido

7.1 Introducción

Definimos parámetros como el conjunto de datos que determina el comportamiento del satélite cuyo valor puede ser modificado desde tierra y no cambia por el funcionamiento autónomo del software.

Durante el proceso de desarrollo, se identificaron los parámetros de funcionamiento de todos los módulos. Se detectaron varios conjuntos de valores para esos parámetros.

7.2 Parámetros por defecto

Los valores enviados desde tierra se almacenan en el Módulo de Control Principal por lo que los restantes módulos requieren comunicarse con el primero para obtenerlos.

Para minimizar la dependencia del correcto funcionamiento del Módulo de Control Principal, se definió un primer conjunto de parámetros que cada uno de los módulos carga desde su propia memoria cada vez que se inicia. Definimos a este conjunto de valores como Parámetros por defecto. Estos parámetros son especialmente relevantes en el caso del Módulo de Gestión de Energía, ya que es el primero que se enciende y es probable que pase un tiempo prolongado antes de encender al Módulo de Control Principal.

Estos valores se definen en tiempo de compilación y no cambian durante la misión.

7.3 Parámetros de fábrica

Los parámetros alojados en el Módulo de Control Principal pueden ser modificados durante la misión utilizando comandos de tierra. Existe la necesidad de restaurar los parámetros al estado inicial de la misión.

A este conjunto de valores alojados en el Módulo de Control Principal le llamamos "Parámetros de fábrica". Los valores resultaron ser en varios casos diferentes a los parámetros por defecto.

7.4 Parámetros activos

Los valores de los parámetros se van modificando a lo largo de la misión. Para prevenir fallas permanentes derivadas de valores inadecuados de los parámetros, los cambios no se almacenan en memoria FLASH hasta que no hay una solicitud expresa de tierra. Los valores modificados se mantienen en la memoria RAM del Módulo de Control Principal. A estos valores, los llamamos parámetros activos

7.5 Parámetros almacenados

Cuando un comando de tierra así lo solicita, los parámetros activos son almacenados en la memoria FLASH. Cada vez que el Módulo de Control Principal inicia, carga en memoria RAM estos valores, los que son entregados a los módulos cuando lo solicitan.

7.6 Aseguramiento de los parámetros de fábrica

Los parámetros de fábrica se almacenan en la memoria FLASH del Módulo de Control Principal. Una alteración de estos datos puede tener consecuencias fatales para la misión.

Para asegurar sus valores, los parámetros de fábrica se almacenan por triplicado y su valor se obtiene utilizando la función mayoría.

7.7 Notificación de cambios

Cuando el Módulo de Control Principal recibe una actualización de parámetros de tierra, actualiza sus datos locales pero no comunica los cambios a los módulos involucrados debido a que cambios en los parámetros individuales durante la ejecución pueden llevar a resultados inesperados.

Una vez modificados todos los parámetros deseados es posible notificar a un módulo utilizando un comando.

7.8 Módulos con parámetros

Los módulos de "Gestión de Energía" y "Detección y Control de Actitud" ejecutan la siguiente secuencia de acciones al iniciar:

- Establecer valores por defecto
- Enviar un mensaje a MCS solicitando sus parámetros
- Esperar un mensaje de MCS con una respuesta por un tiempo limitado
- Si no se obtiene respuesta, continuar con los parámetros por defecto
- Quedar a la espera de parámetros enviados por el Módulo de Control Principal.

7.9 Módulos sin parámetros

Si bien el diseño se realizó para todos los módulos, el módulo de comunicaciones COMM1 resultó no tener parámetros. Este módulo inicialmente contaba con un receptor y el transmisor Morse. Al finalizar el desarrollo, el transmisor Morse se integró al Módulo de Gestión de energía y COMM1 terminó siendo únicamente un receptor.

Por otra parte, el módulo de comunicaciones COMM2 resultó tener como único parámetro el "Modo de transmisión" que puede tomar valores AFSK o FSK. Para simplificar la implementación y reducir la cantidad de procesos, considerando que todos los mensajes enviados a tierra son solicitados por el Módulo de Control Principal, el parámetro se movió a este último módulo y se agregó a cada solicitud el modo de transmisión.

El Módulo Payload se diseñó con un sistema de almacenamiento propio por lo que no utiliza este servicio.

7.10 Persistencia y recuperación de los parámetros almacenados

Cada vez que es requerido desde tierra, los parámetros son almacenados por triplicado en la memoria FLASH, específicamente en la parte alta del Banco A. Antes de escribir cada copia, todo el espacio de almacenamiento necesario se borra y luego de almacenados todos los datos, se etiquetan con la fecha y hora del inicio del comando.

De esta forma, cuando se desea recuperarlos, si las tres fechas coinciden, los parámetros se recuperan utilizando la función mayoría, si no coinciden, se utiliza un algoritmo para recuperar la copia más segura.

Capítulo 8.

Comandos

Contenido

8.1 Introducción

Los comandos enviados al satélite deben enviarse cifrados. Los detalles se presentan en la sección “Comunicación Tierra – Satélite”.

El procesador de comandos del satélite puede recibir en una comunicación una secuencia de comandos tan larga como el tamaño del mensaje lo permita.

Todos los comandos comienzan con un identificador de un byte y pueden ser de largo fijo o variable. A continuación se detallan los comandos implementados.

Los comandos se ejecutan secuencialmente. Si el procesador de comandos encuentra el identificador **END** o un identificador desconocido, detiene la ejecución aunque existan datos sin procesar.

8.2 Actualización de la hora del satélite

El comando **SET_TIME** permite actualizar la hora del satélite. Tiene cinco bytes de largo y está compuesto por el identificador seguido de un entero sin signo de 32 bits en notación little endian.

El entero debe contener la hora UTC expresada en formato UNIX.

Su ejecución modifica inmediatamente la hora del sistema, genera un evento **MCS_TIME_UPDATED** y dispara la notificación de la nueva hora al Módulo de Gestión de Energía y al Módulo de Detección y Control de Actitud si este está encendido.

8.3 Actualización de los parámetros orbitales

Existen dos formas de actualizar los parámetros orbitales:

El comando **SET_TLE** permite actualizar los parámetros orbitales. Tiene 139 bytes de largo y está compuesto por el identificador seguido de 138 bytes, los primeros 69 caracteres de cada una de las líneas del TLE que se desea entregar.

Su ejecución modifica inmediatamente el registro de TLE en RAM, genera un evento **MCS_TLE_UPDATED** y dispara la notificación al Módulo de Detección y Control de Actitud si este está encendido.

Alternativamente, el comando **SET_COMP_TLE** permite actualizar los parámetros orbitales. Tiene 68 bytes de largo y está compuesto por el identificador seguido de 67 bytes que contienen cada uno en los cuatro bits de menor peso, la codificación de los caracteres 2 a 68 de la línea 1 del TLE y en los cuatro bits de mayor peso, la codificación de los caracteres 2 a 68 de la línea 2 del TLE. La tabla 8 muestra la codificación utilizada.

El siguiente ejemplo ilustra la codificación.

TLE sin comprimir:

```
1 23455U 94089A 97320.90946019 .00000140 00000-0 10191-3 0 2621
2 23455 99.0090 272.6745 0008546 223.1686 136.8816 14.11711747148495
```

TLE codificado:

```
0x22, 0x33, 0x44, 0x55, 0x55, 0xAE, 0xAA, 0x99,. . . 0x46, 0x92,
0x51
```

Esta compresión permite entregar un TLE con un mensaje corto y fue diseñada para ser utilizada en caso de que existan problemas de comunicación con los mensajes largos.

La ejecución de este comando modifica inmediatamente el registro de TLE en RAM, genera un evento **MCS_TLE_UPDATED** y dispara la notificación al Módulo de Detección y Control de Actitud si este está encendido.

Tabla 8 – Codificación de los parámetros orbitales

Caracter	Codificación	Descripción
0..9	0x0..0x9	Números
‘ ’	0xA	Espacio
‘+’	0xB	Signo de suma
‘-‘	0xC	Signo de resta
‘.’	0xD	Punto
‘U’	0xE	Letra U
‘A’	0xF	Letra A

8.4 Actualización de los parámetros de los módulos

El comando **SET_PARAMETER** permite actualizar un parámetro. Tiene 5 bytes de largo y está compuesto por el identificador del comando seguido por el identificador del módulo del que se desean modificar los parámetros –1 byte–, la posición del parámetro –un entero de 8 bits en notación little endian–, y el nuevo valor para el parámetro –un entero de 16 bits en notación little endian–.

Los identificadores de módulo válidos para esta operación son:

- **MODULE_MCS**
- **MODULE_EMS**
- **MODULE_ADCS**

Los parámetros de los módulos pueden tener diferentes estructuras y formatos. Proveer mecanismos para modificar los datos considerando sus características resultaba demasiado complejo por lo que se determinó considerar a los parámetros de cada módulo como un arreglo de palabras y proveer los comandos para modificar los valores de estas.

De esta forma, es posible modificar por ejemplo el valor de un parámetro de tipo float (4 bytes) ejecutando dos comandos.

Los tipos int, long, float y double tienen tamaño múltiplo de 2. Los datos de tamaño impar generan desalineaciones en la estructura por lo que fueron colocados al final.

La modificación de parámetros de tamaño impar debe realizarse con precaución ya que el cambio afecta a una cantidad par de bytes en la estructura de parámetros.

La ejecución de este comando modifica inmediatamente el valor de los parámetros activos. La modificación no se persiste ni se informa al módulo propietario de los parámetros modificados.

8.5 Persistencia de los parámetros activos

El comando **SAVE_PARAMETERS** permite almacenar los parámetros activos en la memoria flash del Módulo de Control Principal. Tiene un byte de largo para alojar únicamente al identificador del comando.

La ejecución de este comando modifica el valor de los parámetros almacenados y genera un evento **MCS_PARAMETERS_SAVED**.

8.6 Recuperación de los parámetros almacenados

El comando **RELOAD_PARAMETERS** permite recuperar los parámetros almacenados en la memoria flash del Módulo de Control Principal. Tiene un byte de largo para alojar únicamente al identificador del comando.

Su ejecución modifica inmediatamente los parámetros activos. Si no hay parámetros almacenados o los datos almacenados no son consistentes, los parámetros activos no se modifican.

8.7 Recuperación de los parámetros de fábrica

El comando **FACTORY_PARAMETERS** permite recuperar los parámetros de fábrica. Tiene un byte de largo para alojar únicamente al identificador del comando.

Su ejecución modifica inmediatamente los parámetros activos, no altera el valor de los parámetros almacenados ni notifica a los restantes módulos.

8.8 Encendido y apagado de módulos

El comando **MODULE_CONTROL** permite encender y apagar módulos. Tiene 3 bytes de largo y está compuesto por el identificador del comando seguido por el identificador del módulo que se desea afectar –1 byte– y el estado deseado para el módulo –1 byte–.

Los identificadores de módulo válidos para esta operación son:

- **MODULE_COMM1**
- **MODULE_COMM2**
- **MODULE_ADCS**
- **MODULE_PAYLOAD**

Los valores válidos para el estado deseado son:

- **0x01: Encendido**
- **0x00: Apagado**

Establecer el estado deseado de COMM1 en apagado fuerza el estado deseado de COMM2 al valor encendido y planifica la restauración de COMM1 en 72 horas.

Establecer estado deseado de COMM2 en apagado fuerza el estado deseado de COMM1 al valor encendido y planifica la restauración de COMM2 en 72 horas.

La ejecución de este comando modifica los valores de los estados deseados y notifica al Módulo de Gestión de Energía.

8.9 Encendido y apagado de servicios

El comando **SERVICE_CONTROL** permite encender y apagar servicios. Tiene 3 bytes de largo y está compuesto por el identificador del comando seguido por el identificador del servicio que se desea afectar –1 byte– y el estado deseado para el servicio –1 byte–.

Los identificadores de servicio válidos para esta operación son:

- **SERVICE_SSTV**
- **SERVICE_RADIO**
- **SERVICE_DIGIPEATER**

Los valores válidos para el estado deseado son:

- 0x01: Encendido
- 0x00: Apagado

Establecer el estado deseado de **SERVICE_RADIO** planifica la restauración de COMM1 en 72 horas.

Modificar el estado deseado de **SERVICE_RADIO** genera una notificación inmediata al Módulo de Gestión de Energía.

La ejecución de este comando modifica inmediatamente los valores de los estados deseados.

8.10 Reinicio del satélite

El comando **SYSTEM_REBOOT** permite reiniciar todo el sistema. Tiene un byte de largo para alojar únicamente al identificador del comando.

La ejecución del comando genera un evento **MCS_REBOOT_COMMAND**, una notificación al Módulo de Gestión de Energía y una espera de 5 segundos. Si en ese tiempo el Módulo de Gestión de Energía no reinició al Módulo de Control Principal, este se reinicia.

8.11 Asignación del mensaje de usuario

El comando **SET_USER_MSG** permite definir el mensaje de usuario que se agrega a la baliza Morse. Tiene un tamaño mínimo de 4 bytes y un máximo de 4 más el valor de la constante **USER_MESSAGE_MAX_LENGHT**. Está compuesto por el identificador de comando seguido por el tiempo que se emitirá el mensaje expresado en segundos –un entero de 16 bits en notación little endian–, la cantidad de caracteres del mensaje –un entero sin signo de 8 bits– y tantos caracteres como indica esa cantidad.

Los caracteres del mensaje están limitados al Alfabeto Morse Internacional. Lo caracteres válidos son:

- Números: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Símbolos: ?, /, Ñ, @, espacio, punto, coma
- Letras: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

La ejecución de este comando genera una notificación de tipo **SET_USER_MESSAGE** al procesador de comandos del Módulo de Gestión de Energía. Esta notificación incluye que incluye el tiempo y el mensaje recibido.

8.12 Descarga de eventos

El comando **DOWNLOAD_LOG** permite controlar la descarga del registro de eventos. Tiene un largo de 6 bytes y está compuesto por el identificador del comando seguido por la cantidad de bloques que se desea descargar –un entero sin signo de 8 bits – y una marca de tiempo UNIX.

El comando dispara un proceso que busca el primer bloque con fecha mayor o igual a la fecha indicada y envía cuatro mensajes AX25 por bloque hasta agotar los datos, completar el número de bloques solicitados o recibir otra solicitud.

Los mensajes AX25 son dirigidos a la estación que envió el telecomando, son de tipo "Unnumbered Information" (Control = 0x03), sin protocolo definido (Protocol = 0xF0) y contienen el identificador del mensaje seguido por la marca de tiempo inicial del bloque y 128 bytes de datos.

El identificador del mensaje toma para cada parte del bloque respectivamente los valores:

- **DOWNLOAD_LOG_PART1**
- **DOWNLOAD_LOG_PART2**
- **DOWNLOAD_LOG_PART3**
- **DOWNLOAD_LOG_PART4**

8.13 Volcado de memoria del Módulo de Control Principal

El comando **DUMP_MEMORY** permite obtener el contenido de la memoria del Módulo de Control Principal. Tiene un largo de 3 bytes y está compuesto por el identificador del comando seguido de los 16 bits más significativos de la primera dirección de memoria extendida (20 bits) que se desea explorar.

Su ejecución genera el envío de un mensaje AX25 dirigido a la estación que envió el telecomando. El mensaje es de tipo "Unnumbered Information" (Control = 0x03), sin protocolo definido (Protocol = 0xF0) y contiene el identificador **MCS_MEMORY_DUMP** seguido por los 16 bits más significativos de la dirección de memoria inicial y 128 bytes de datos extraídos a partir de esa dirección.

8.14 Redireccionar un mensaje

El comando **DIRECT_MSG** permite reenviar un mensaje a una aplicación de cualquier módulo del satélite incluyendo el mismo Módulo de Control Principal. Tiene un largo variable que va entre 4 bytes y el máximo admisible por el protocolo de capa inferior. Está compuesto por el identificador del comando seguido del largo de datos –un entero sin signo de 8 bits– y tantos caracteres como indique el largo.

Los datos de este comando se entregan como parámetro a la función "send" –interfaz del protocolo IMMP–.

Este es tal vez el comando más potente ya que permite entregar cualquier tipo de mensaje a cualquier aplicación en cualquier módulo, con la única limitación de que los mensajes tienen como origen al Módulo de Control Principal.

8.15 Demorar la ejecución

El comando **DELAY** permite demorar la ejecución del resto de los comandos. Tiene un byte de largo para alojar únicamente al identificador del comando.

Su ejecución introduce una espera de un tiempo definido en tics en la variable **DELAY_COUNT**.

Este comando es útil por ejemplo para reiniciar módulos ejecutando la secuencia:

- Apagado usando **MODULE_CONTROL**
- Pausa usando **DELAY**
- Encendido usando **MODULE_CONTROL**

8.16 Forzar la entrega de parámetros

El comando **DELIVER_PARAMETERS** permite forzar la entrega de parámetros a un módulo. Tiene un largo de 2 bytes y está compuesto por el identificador del comando seguido por el identificador del módulo al que se desea entregar los parámetros.

Los identificadores de módulo válidos para esta operación son:

- **MODULE_EMS**
- **MODULE_ADCS**

Su ejecución fuerza la entrega de los parámetros del módulo indicado al procesador de comandos del mismo.

8.17 Detención condicional del proceso de comandos

El comando **STOP_IF_DATE_GT** permite detener el proceso de comandos si la hora del sistema supera un valor dado. Tiene un largo de 5 bytes y está compuesto por el identificador del comando seguido de una marca de tiempo UNIX –un entero sin signo de 32 bits–.

Este comando está diseñado para proteger al sistema. En algunos casos puede ser necesario solicitar a terceros que envíen mensajes al satélite. Si bien el satélite recuerda los últimos mensajes recibidos para no procesar varias veces el mismo, un tercero podría repetir el mensaje diferido en el tiempo y provocar un comportamiento inadecuado. Para evitar esto basta con colocar este comando al principio de la secuencia con una marca de tiempo cercana a la actual. De esta forma el mensaje puede ser entregado nuevamente pero el procesador de comandos ignorará el resto del mensaje.

Capítulo 9.

Eventos y registros

Contenido

9.1 Eventos

Durante la vida útil del satélite existen eventos relevantes que interesa conocer. La ubicación del satélite respecto a las estaciones terrenas no permite reportar inmediatamente los datos por lo que estos deben almacenarse hasta que la comunicación sea posible.

Durante el inicio del presente trabajo, el registro se definía como "House keeping" – una recolección periódica de datos de los módulos almacenada en la memoria flash del Módulo de Control Principal. Con la evolución del proyecto, el concepto se transformó. Los módulos pasaron a reportar sus datos en lugar de ser interrogados y la estructura de datos pasó de ser rígida a totalmente flexible. Esto permitió que se reportaran eventos puntuales como el despliegue de antenas del Módulo de Gestión de Energía o un cambio en el estado del sistema de Detección y Control de Actitud, y a la vez, permitió que se optimizara el uso del espacio de almacenamiento.

Definimos "registros" como el conjunto de datos que resultan del funcionamiento del satélite que se desea conocer en tierra y de cuyos valores no depende la ejecución del software.

9.2 Almacenamiento

El Módulo de Control Principal cuenta con un espacio de memoria FLASH como medio de almacenamiento. No se trata de un dispositivo independiente sino del mismo espacio en el que se almacenan los programas. El compilador coloca el código y las constantes en las direcciones más bajas. La mayor parte del espacio disponible se encuentra en las direcciones por encima de 0xFFFF.

Para maximizar el espacio de almacenamiento se determinó utilizar los bancos B, C y D para los registros –192 KByte entre las direcciones 0x10000 y 0x3FFFF–. La parte alta del banco A se reservó para almacenar las múltiples copias de los parámetros –23 KByte entre las direcciones 0x040000 y 0x045BFF–.

Las características del borrado determinan el tamaño del segmento sea la unidad básica de almacenamiento.

Es probable que suceda un reinicio durante un proceso de escritura o borrado y cada vez que el procesador inicia debe detectar cuál es la posición del último registro. Se analizaron varias alternativas para asegurar el almacenamiento. La que resultó más robusta fue el utilizar este espacio como buffer circular de segmentos, cada uno etiquetado con la hora del primer registro almacenado. En la figura 20 se muestra un esquema al respecto.

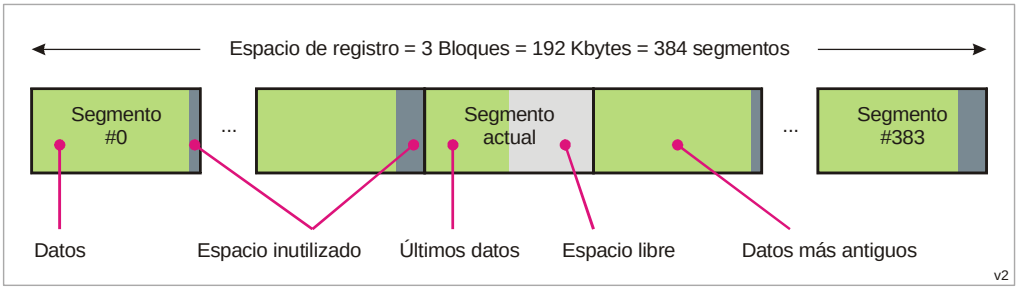

Figura 20 – Estrategia de almacenamiento de eventos

Uno de los elementos cruciales en este sistema de almacenamiento es la hora, al punto de que su representación se reformuló para adaptarse a este registro.

La lectura de esta memoria no tiene diferencias con la lectura de la RAM. La escritura requiere seguir un protocolo con el periférico. Si bien existen varias opciones, se optó por utilizar solamente la escritura de palabras dobles (32 bits).

Para poder utilizar la parte alta del banco A –el mismo en el que se aloja el código– se hizo necesario alojar las rutinas de borrado y escritura de la FLASH en memoria RAM. Esto se logró usando directivas del compilador.

El usar los segmentos como unidad de borrado determinó que los registros no pudieran exceder los 512 bytes, lo que no resultó ser un problema ya que la cota del mensaje entre módulos está bastante por debajo de ese valor. Considerando esta cota y con la intención de optimizar el uso de memoria, el tamaño del registro se limitó a 256 bytes.

9.2.1 Formato de registro

Los datos se almacenan en conjuntos de 4 bytes que llamamos elementos. Cada elemento es el resultado de una operación de escritura. Cada registro tiene como mínimo dos elementos con la siguiente estructura:

- Fecha y hora (4 bytes)
- Cantidad de elementos adicionales (1 byte)
- Módulo que genera el registro (1 byte)
- Identificador del evento (1 byte)
- Datos (1 byte)

Los registros pueden tener hasta 248 bytes de datos adicionales (62 elementos). En la figura 21 se muestra un esquema. Por lo tanto, los registros tienen entre 8 y 256 bytes, lo que permite que cada bloque aloje entre 2 y 64 registros.

Cuando un registro no puede ser alojado en el segmento en uso, el espacio remanente se deja en blanco y el siguiente segmento del buffer se borra para almacenar los datos.

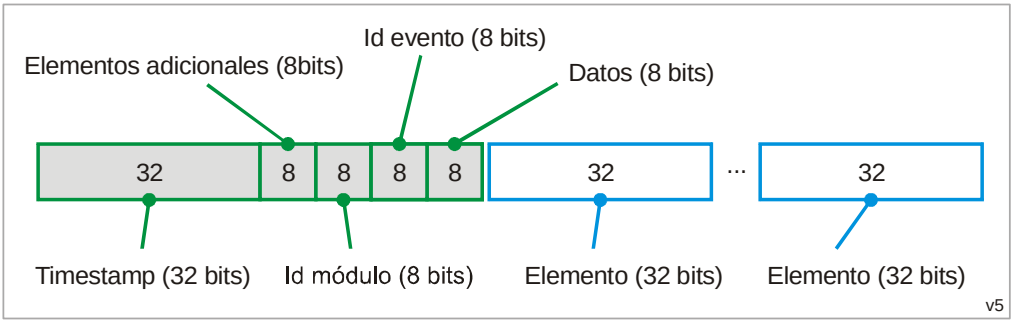

Figura 21 – Formato de los registros

9.3 Servicio de registro

Los diferentes módulos necesitan realizar sus registros, ya sea para almacenar datos en forma periódica o para reportar situaciones puntuales. Para facilitar este proceso el Módulo de Control Principal provee un servicio a través del cual los distintos procesos del satélite dentro o fuera del módulo pueden enviar un mensaje a la aplicación MCS_APP_LOG_SERVICE con el identificador del evento que desean registrar y los datos necesarios para el caso. El servicio de registro se encarga de agregar la hora y almacenar los datos.

Capítulo 10.

Protocolos de comunicación

Contenido

10.1 Comunicación entre módulos

Uno de los grandes desafíos del proyecto fue construir un protocolo de comunicación entre los módulos que satisficiera las necesidades de la misión.

En etapas tempranas del diseño del hardware se determinó utilizar un bus I²C para interconectar las piezas. En el prototipo [2] el bus I²C resultaba controlado únicamente por el Módulo de Control Principal.

Una de las decisiones más importantes en la definición del protocolo fue la de restringir los mensajes a comunicaciones unidireccionales de escritura. Varios fueron los motivos de esta decisión, entre ellos la gran cantidad de errores conocidos en la implementación del periférico y la necesidad de simplificar la comunicación a bajo nivel. Esta decisión permitió desplazar la complejidad a las capas superiores. De cualquier manera, para implementar la comunicación fue necesario crear dos aplicaciones y una máquina de estado que involucra a dos ISR.

Para lograr una implementación robusta fue necesario considerar todos los comportamientos posibles de los dispositivos involucrados, incluyendo los imprevistos. Además, se agregó temporización a todas las esperas por eventos.

10.1.1 Objetos de la comunicación

Desde etapas tempranas del desarrollo, estuvo definido que el protocolo debía transportar mensajes. En ningún momento se detectó la necesidad de transportar flujos de datos.

La mayor parte de las comunicaciones necesarias resultaron ser de unos pocos bytes salvo los mensajes AX25 entre los módulos de comunicaciones y el Módulo de Control Principal. Estos mensajes se convirtieron en la cota superior de las necesidades definiendo el tamaño máximo en 300 bytes.

10.1.2 Protocolo de transporte

Las primeras etapas de diseño tuvieron como objetivo lograr la comunicación entre módulos utilizando I²C. A medida que avanzó el proyecto el objetivo pasó a ser el lograr la comunicación entre aplicaciones.

El protocolo resultante que llamamos IMMP –por Inter Module Message Protocol–, es capaz de canalizar los mensajes hacia un conjunto de recursos predefinidos del uKernel. De esta forma, los procesos registrados pueden recibir mensajes de aplicaciones del mismo u otros módulos en forma totalmente transparente.

10.1.3 Estructura de los mensajes

La aplicación que genera un mensaje debe especificar a qué módulo y aplicación deben entregarse los datos.

La aplicación que recibe los datos, necesita saber desde qué módulo proviene el mensaje y, opcionalmente, a qué aplicación debe dirigir las respuestas.

Al primer byte de datos le llamamos “identificador del mensaje” pero el protocolo no controla su valor. En general este dato suele ser un comando o identificador de tipo de mensaje. Su presencia es obligatoria.

Con la configuración utilizada, los mensajes IMMP pueden transportar hasta 297 bytes adicionales.

Una de las particularidades del protocolo es que la cabecera de los mensajes salientes es diferente a la de los mensajes entrantes. Si bien los datos omitidos no son necesarios, una futura versión debería unificar estas cabeceras.

En las figuras a continuación se muestra la estructura de los mensajes IMMP enviado y recibido.

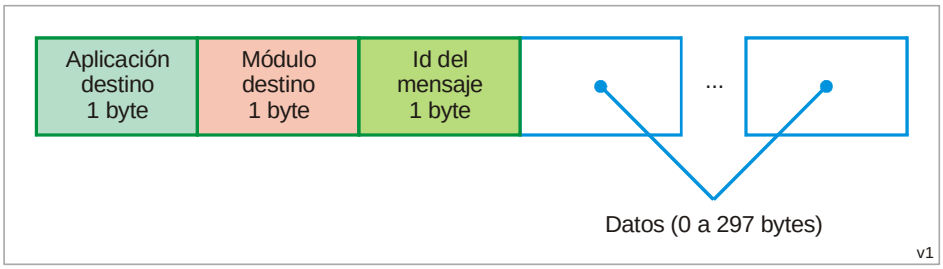

Figura 22 – Mensaje saliente IMMP

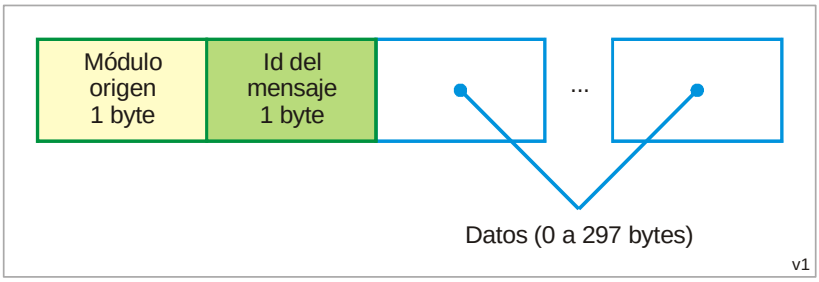

Figura 23 – Mensaje entrante IMMP

10.1.4 Componentes de la implementación

La implementación del protocolo está constituida por tres piezas. El proceso transmisor, el control de transmisión y recepción, y el distribuidor de mensajes recibidos.

Esta implementación utiliza los recursos del kernel para comunicar y sincronizar las piezas.

10.1.4.1 Transmisor

El proceso transmisor recibe los datos a enviar a través de un recurso IMMP_REQUEST y reporta el resultado a través del recurso IMMP_REPLY.

Para enviar datos es necesario bloquear el recurso IMMP_REPLY, establecer un espacio de transferencia para recibir el resultado de la operación, entregar los datos al recurso IMMP_REQUEST, leer el resultado obtenido a través de IMMP_REPLY y liberarlo para que otros procesos puedan realizar envíos.

Se provee una función “send” que encapsula todas estas operaciones pero no necesariamente debe utilizarse. Si ninguno de los procesos utiliza el resultado del envío, es suficiente con entregar los datos al recurso IMMP_REQUEST.

El proceso transmisor configura al proceso de control y procesa el resultado realizando varios intentos en caso de colisión o falta de respuesta o falla de la comunicación.

10.1.4.2 Control

El control de transmisión y recepción está implementado utilizando el servicio de dos interrupciones. La ISR del periférico asociado al bus I2C y la ISR del temporizador seleccionado implementan una máquina de estado que gestiona todos los casos posibles.

La comunicación tiene tres estados posibles: libre, transmitiendo y recibiendo. Estos estados tienen un enfoque interno. El estado libre implica que no se está transmitiendo ni recibiendo pero no necesariamente que el bus esté libre.

Todas las esperas por eventos son temporizadas y la comunicación es cancelada en caso de falla.

El sistema interruptivo coloca los mensajes recibidos en un buffer y señaliza al proceso distribuidor a través del recurso IMMP_RX.

10.1.4.3 Distribuidor

El proceso distribuidor analiza los mensajes en el buffer de recepción y utiliza una tabla de traslación de aplicación a recurso para determinar dónde entregar cada mensaje.

Cuando el recurso destino está disponible el distribuidor entrega el mensaje, de lo contrario retiene los datos en el buffer.

Cuando hay poco espacio disponible en el buffer, el distribuidor descarta los mensajes antiguos que no se pudieron entregar.

10.1.5 Envío y recepción

Para enviar un mensaje basta con anteceder el mismo con dos bytes indicando la aplicación y módulo de destino y utilizar la función “send” indicando la dirección de los datos y el largo de los mismos.

La función “send” permite además entregar mensajes en el mismo módulo de origen en forma totalmente transparente para los procesos que intervienen en la comunicación.

Para recibir mensajes es necesario haber definido en el archivo de configuración de IMMP un recurso asociado a la aplicación destino. Cumplido esto basta con utilizar la función “read” del kernel sobre el recurso.

10.1.6 Configuración

El protocolo se configura con definiciones en un archivo de nombre “immp_config.h”.

Es posible seleccionar el periférico al que está conectado el bus I2C. Las opciones programadas son I2C_USCI_B0 y I2C_USCI_B1. Para configurarlo basta incluir la definición correspondiente. Por ejemplo:

```
#define I2C_USCI_B0
```

Es posible seleccionar el temporizador utilizado. Las opciones programadas son I2C_TIMER_A0 y I2C_TIMER_B0. Para configurarlo basta incluir la definición correspondiente. Por ejemplo:

```
#define I2C_TIMER_A0
```

Es posible configurar la fuente de reloj del dispositivo y el divisor de entrada. Las opciones programadas son UCSSEL_1 para usar ACLK y UCSSEL_2 para usar SMCLK.

El divisor de frecuencia debe dar como resultado un valor aproximado a 100 KHz puesto que la frecuencia se define como:

```
f = I2CCLK / I2CFDIV
```

La configuración utilizada en el Módulo de Control Principal usa SMCLK (Secondary Master Clock) con una frecuencia de 2 MHz y un divisor de 20. La configuración es la siguiente:

```
#define I2CCLK UCSSEL_2
#define I2CFDIV_msb 0/**< bits mas significativos */
#define I2CFDIV_lsb 20     /**< bits menos significativos */

/* Timeout relativo a system_timer hasta el próximo evento i2c ~ 20
ms */
#define I2C_TIMEOUT 80

/*
     Tamaño maximo de mensaje a transmitir (bytes)
     max ax25 = 286 = 256 data + 30 header
     max immp = 289 = 3 i2c header + ax.25
*/
#define IMMP_TX_BUFFER_LENGTH 300

/* Tamaño del buffer de recepción (bytes) */
#define IMMP_RX_BUFFER_LENGTH 1024

/* Espacio minimo requerido en buffer i2c_rx */
#define MIN_FREE_SPACE 200

/* Largo máximo de los mensajes salientes */
#define MAX_I2C_MESSAGE_LENGTH IMMP_TX_BUFFER_LENGTH

/* Largo mínimo de los mensajes entrantes (ap destino, modulo
origen) */
#define MIN_I2C_MESSAGE_LENGTH 2

/* Tiempo de espera en tics si hay mensajes en el buffer */
#define TIMEOUT_IF_MESSAGE 2

/* Tiempo de espera en tics si no hay mensajes en el buffer */
#define TIMEOUT_IFNOT_MESSAGE 0

/* Tiempo máximo en tics para que la ISR envíe el mensaje */
#define TIMEOUT_I2CISR_SEND 10

/* Tiempo máximo en tics para entregar un mensaje local */
```

```
#define IMMP_LOCAL_SEND_TIMEOUT 10

/* Cantidad de reintentos de la transmisión I2C */
#define I2C_TX_RETRY 5

#define TIMER_I2C 0

/* Direccion Propia */
#define LOCAL_MODULE MODULE_MCS

/* Cantidad de aplicaciones que esperan mensajes por el bus i2c */
#define NUM_OF_APPS_IMMP 8

const resource_t immp_resource_table[NUM_OF_APPS_IMMP] =
{
     RESOURCE_AX25_RX_INPUT,                    /**< MCS_APP_AX25_RX
=0 */
     RESOURCE_AX25_TX_REPLY,                    /**<
MCS_APP_TX_REPLY =1 */
     RESOURCE_TX_SERVICE_INPUT,          /**< MCS_APP_TX_SERVICE =2
*/
     RESOURCE_TIME_SERVICE_INPUT, /**< MCS_APP_TIME_SERVICE =3*/
     RESORCE_LOG_SERVICE_INPUT,          /**< MCS_APP_LOG_SERVICE =4
*/
     RESOURCE_PAR_SERVICE_INPUT,         /**< MCS_APP_PAR_SERVICE
=5*/
     RESOURCE_TELEMETRY_INPUT,           /**< MCS_APP_TELEMETRY =6
*/
     RESOURCE_CODE_UPLOAD_INPUT          /**< MCS_APP_CODE_UPLOAD =7
*/
};
```

10.1.7 Revisiones

Una vez implementado, el protocolo se verificó exhaustivamente. A partir de las pruebas de stress se detectó un comportamiento inesperado del periférico que provocó una revisión completa.

La verificación posterior delató ocasionales problemas de alineación de palabras en memoria dependientes de la compilación. Esto provocó una segunda revisión exhaustiva.

10.2 Comunicación Tierra – Satélite

10.2.1 Recepción AX25

El protocolo AX25 tiene capacidad de realizar comunicaciones orientadas a conexión. A pesar de esto, y basando la decisión en la experiencia generada por otros fabricantes de satélites que optaron por este protocolo de comunicación, se utilizarán sólo tramas sin numerar (un equivalente al protocolo UDP en el stack TCP/IP).

Los módulos de comunicaciones detectan las tramas identificando las banderas de inicio y fin, y controlando el FCS. Las tramas detectadas se envían al módulo de control principal sin incluir las banderas ni el campo de control.

El módulo de control principal tiene un proceso que está a la espera de estas tramas y clasifica cada una de las recibidas.

Los contenidos posibles son:

- Tramas que contienen comandos de tierra para controlar el satélite.
- Tramas que requieren ser repetidas por el satélite
- Tramas que contienen solicitudes de terceros (SSTV)

Nótese que la dirección de origen no se verifica en ningún caso, salvo el LSB del séptimo byte para determinar si hay más direcciones en la cabecera. Esta dirección se entrega junto con los datos a la aplicación correspondiente para que la respuesta, si corresponde, se dirija al mismo origen que la solicitó.

10.2.1.1 Dirección del satélite

La dirección del satélite se definió como CX1SAT con ssid = 0. La figura 24 muestra los valores resultantes de utilizar la representación definida por el protocolo AX25.

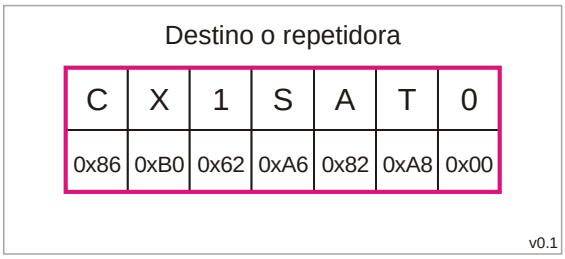

Figura 24 – Dirección AX25 del satélite

10.2.2 Telecomando

Las tramas de telecomando deben estar dirigidas al satélite (Destino = CX1SAT ssid = 0), ser de tipo "Unnumbered Information" (Control = 0x03), sin protocolo definido de capa 3 (Protocol = 0xF0), tener un identificador de origen que indique que proviene o bien de FING o bien de ANTEL, un número de paquete que no se encuentre entre los 10 últimos procesados y un conjunto de datos cifrados de tamaño múltiplo de 8 de entre 8 y 248 bytes cuyo descifrado tenga un contenido consistente.

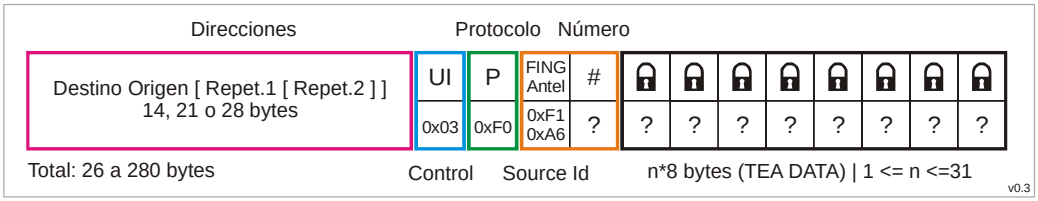

Figura 25 – Contenido de la trama AX25 para Telecomando

10.2.2.1 Consistencia de los datos

Para determinar si un mensaje recibido es consistente, luego de descifrado se verifica que el primer carácter coincida con el identificador de origen y el segundo con el número de paquete.

10.2.2.2 Cifrado TEA

El algoritmo de cifrado y descifrado se ha probado fuera de la plataforma con un conjunto importante de casos de prueba y resultados correctos.

Fuera de la debilidad del algoritmo no parece haber mayores inconvenientes.

Las claves de cifrado serán únicas para toda la misión y no se modificarán ni proveerán mecanismos para modificarla por lo que para asegurarlas se almacenan por triplicado y se recuperan aplicando el algoritmo mayoría.

10.2.3 Servicio de SSTV

El servicio de SSTV permite a terceros solicitar que se tome una fotografía y se envíe a tierra utilizando el método Slow Scan TV.

10.2.3.1 Requerimientos para SSTV

El servicio de SSTV no está siempre operativo. Se necesita habilitación desde tierra para que las solicitudes se envíen al PAYLOAD.

Por otra parte, cada vez que se hace efectiva una solicitud, el servicio se suspende por un tiempo de 5 minutos.

10.2.3.2 Solicitudes de SSTV

Las solicitudes de SSTV deben estar dirigidas al satélite (Destino = CX1SAT ssid = 0), ser de tipo “Unnumbered Information” (Control = 0x03), sin protocolo definido de capa 3 (Protocol = 0xF0), y tener dos bytes de datos con el valor 0x53 (código ASCII de la letra S mayúscula) seguido del una ‘V’ (0x56) para SSTV visible, una ‘I’ (0x49) para SSTV infrarrojo, o una ‘T’ para SSTV de verificación (test).

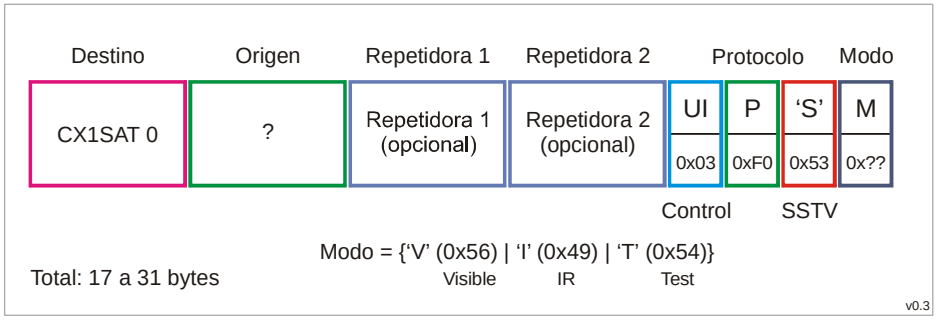

Figura 26 – Contenido de la trama AX25 para Solicitud de SSTV

Estos mensajes son aceptados cuando el servicio de SSTV está habilitado, lo que se reporta en la baliza morse.

10.2.4 Servicio de repetidora (DIGIPEATER)

10.2.4.1 Servicio de repetidora (recepción de DIGIPEATER)

Los mensajes a repetir pueden tener hasta cuatro direcciones en la cabecera AX25 y la dirección del satélite debe estar presente como primera repetidora no utilizada y no debe ser la dirección de destino. La figura 27 y la figura 28 muestran las estructuras aceptadas.

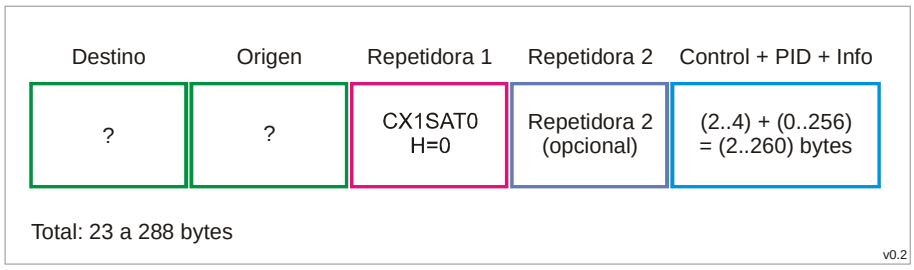

Figura 27 – Contenido de la trama AX25 a repetir (primera repetidora)

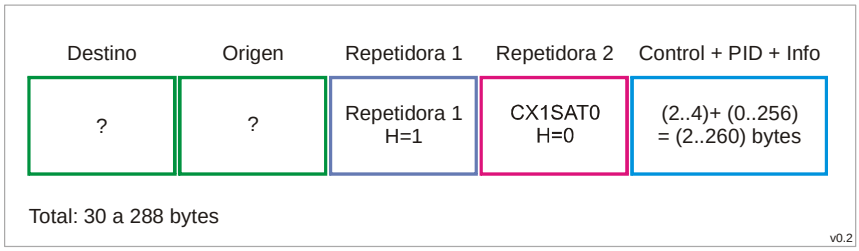

Figura 28 – Contenido de la trama AX25 a repetir (segunda repetidora)

Estos mensajes son aceptados cuando el servicio de DIGIPEATER está habilitado lo que se reporta en la baliza morse.

10.2.5 Transmisión AX25

La transmisión AX25 se realiza utilizando mensajes de tipo "Unnumbered Information" (Control = 0x03), sin protocolo definido de capa 3 (Protocol = 0xF0).

El servicio de transmisión construye el encabezado con una dirección de destino recibida como parámetro, la dirección del satélite como origen, sin usar repetidoras y con un conjunto de datos de largo de entre 0 y 255 caracteres.

El servicio de transmisión no asegura la entrega del mensaje, todos los mensajes son de tipo "Final" de acuerdo con la definición del campo de control del protocolo AX25.

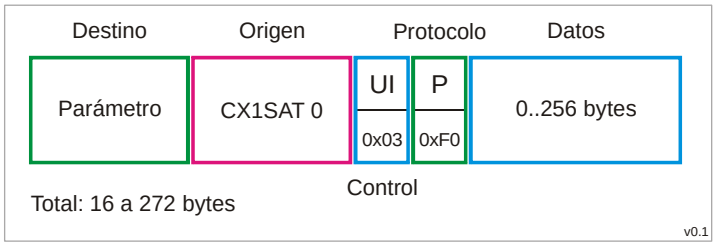

Figura 29 – Contenido de la trama AX25 enviadas a tierra.

Existen varios tipos de mensaje. El contenido depende de la aplicación que los envía.

Capítulo 11.

Verificación e integración

Contenido

11.1 Verificación

El desarrollo realizado tiene dos grandes partes. Por una lado está el kernel y por otro las aplicaciones. Para realizar la verificación se utilizaron dos metodologías, una para cada parte de acuerdo con la naturaleza de cada una.

11.1.1 Verificación del Kernel

El kernel desarrollado gestiona el procesador y los recursos.

La parte que gestiona el procesador está escrita en Assembler. Las actividades realizadas tienen que ver con la preparación del contexto, el almacenamiento para suspender un proceso y su restauración.

Para verificar el comportamiento de esta parte se diseñaron casos de prueba para conjuntos de entre uno y cinco procesos interactuando. Este comportamiento se verificó primero en un depurador analizando los valores de los registros, usando un simulador y luego sobre el procesador.

La parte que gestiona los recursos está escrita en lenguaje C. Las actividades realizadas tienen que ver con la gestión de las colas de espera por un recurso y suspensión temporizada a la espera de un evento.

Para verificar el comportamiento de esta parte se diseñaron casos de prueba con procesos intercambiando datos y compitiendo por los recursos. El comportamiento se verificó exhaustivamente con un simulador y utilizando herramientas de depuración sobre el microcontrolador en la placa de desarrollo.

Estos casos de prueba evolucionaron con las versiones del kernel.

11.1.2 Verificación de las aplicaciones y protocolos

Una vez asegurado el comportamiento del kernel las aplicaciones pudieron verificarse como piezas independientes.

La mayor parte de las aplicaciones son reactivas a mensajes, es decir, se ejecutan hasta bloquearse a la espera de un evento cuya notificación y/o datos le son entregados a través de un recurso.

Esto provee una gran ventaja en el momento de la verificación porque los mensajes pueden ser enviados por una aplicación sustituta.

Por ejemplo: La aplicación que recibe los mensajes de los módulos de comunicaciones implementa la capa de enlace AX25.

Generar estos mensajes y enviarlos al satélite resulta bastante complejo. En su lugar, para las pruebas iniciales se utilizó una aplicación sustituta lo que facilitó ampliamente la verificación.

La implementación de los protocolos a bajo nivel usa también recursos del kernel para comunicarse. De igual modo, la verificación se valió de esta ventaja para emular el comportamiento de una ISR.

Para todos los casos la herramienta de simulación incluida en el entorno de desarrollo resultó muy útil. La única desventaja que tiene es que no emula los periféricos incluidos en el microcontrolador y en los casos donde puede hacerlo, no emula los errores conocidos de la implementación de hardware.

11.2 Integración

El software del Módulo de Control Principal estuvo listo antes de que comenzaran a implementarse el Módulo de Gestión de Energía y el Módulo de Detección y Control de Actitud. Gran parte de los procesos interactúan con los restantes módulos por lo que para validar la implementación fue necesario simular el comportamiento de los procesos interlocutores.

11.2.1 Integración con el Módulo de Gestión de Energía

Cuando el hardware de los módulos comenzó a estar disponible y los requerimientos de los módulos de gestión de energía y actitud comenzaron a definirse, se inició la construcción de una nueva versión del Módulo de Control Principal.

Paralelamente, el equipo de Simón González y Pablo Yaniero construyó el prototipo del software del Módulo de Gestión de energía. Basado en este prototipo se realizaron las pruebas del hardware y ajuste del software asociado (a cargo de Andrés Touya).

A medida que se fueron ajustando esos detalles se fueron construyendo las piezas del software de vuelo del Módulo de Gestor de Energía que se describen a continuación:

Figura 30 – Pruebas de integración en laboratorio[1]

11.2.1.1 Gestor de parámetros

Este proceso establece los parámetros por defecto y cuando el Módulo de Control Principal esta encendido le solicita los valores de los parámetros actualizados. Posteriormente queda a la espera de actualizaciones.

Junto con el gestor de parámetros se implementa otro proceso que se ocupa de restaurar el estado deseado de los módulos de comunicaciones, deshabilitar el silencio de radio y proveer de intentos adicionales a los módulos que fueron deshabilitados por fallar repetidamente.

11.2.1.2 Gestor de energía

Este proceso determina el estado de energía del satélite y controla el estado de los restantes módulos y servicios.

Los elementos controlados son:

- Transmisor de la baliza Morse.
- Bus de comunicaciones principal (I2C)

[1] Fotografía gentileza de Andrés Touya.

- Módulo de Control Principal
- Módulo de Comunicaciones COMM1
- Módulo de Comunicaciones COMM2
- Módulo de Detección y Control de Actitud
- Payload
- Transmisor de banda S N° 1
- Transmisor de banda S N° 2

Existen tres estados posibles.

- PPOD
- RECOVERY
- SAFE

Mientras no se desplieguen las antenas –lo que se realiza utilizando mecanismos de hardware– el satélite se mantiene en estado PPOD. En este estado todos los elementos controlados están apagados.

En estado RECOVERY, sólo el transmisor Morse estará encendido (salvo que desde tierra se haya solicitado apagarlo)

En estado SAFE, el bus I2C y el Módulo de Control Principal estarán encendidos y los restantes elementos se encenderán o apagarán en función de las solicitudes de tierra.

11.2.1.3 Procesador de comandos

Este proceso recibe los siguientes mensajes del Módulo de Control Principal:

- Solicitud de reinicio
- Mensaje de usuario, el texto que se agrega a la baliza Morse a solicitud de tierra.
- Habilitación o no de las transmisiones Morse (silencio de radio)
- Estado deseado de los módulos COMM1, COMM2, ADCS y PAYLOAD.

11.2.1.4 Reporte de telemetría

Este proceso realiza periódicamente un conjunto de mediciones y las envía al Módulo de Control Principal para que este construya los informes de telemetría del satélite.

El reporte contiene más de treinta lecturas de valores analógicos obtenidos mediante el conversor analógico digital del procesador. Estos valores representan tensiones, corrientes y temperaturas.

11.2.1.5 Mantenimiento de la hora

Este proceso participa en la negociación de la hora con el Módulo de Control Principal. Es capaz de recibir y responder solicitudes de hora, además de recibir y ejecutar comandos de ajuste de hora.

11.2.1.6 Configuración de IMMP y Kernel

Otro de los aspectos necesarios para realizar la integración del Módulo de Control Principal con el Módulo de Gestión de Energía es la configuración del protocolo IMMP y el Kernel.

El protocolo IMMP requiere que las aplicaciones que reciben mensajes estén declaradas y exista una tabla de traslación para determinar los recursos a los que la implementación del protocolo debe entregar los mensajes recibidos.

El Kernel requiere que se determinen los controles que realizará, se declaren los procesos y los recursos.

La configuración realizada alcanza los procesos de aplicaciones y recursos mencionados en esta sección.

11.2.2 Integración con el Módulo de Detección y Control de Actitud

Para este módulo también se desarrollaron los procesos básicos para interactuar con el Módulo de Control Principal. Este trabajo incluyó la configuración de IMMP y del Kernel, el proceso de mantenimiento de la hora, el manejador de parámetros y el reporte de telemetría.

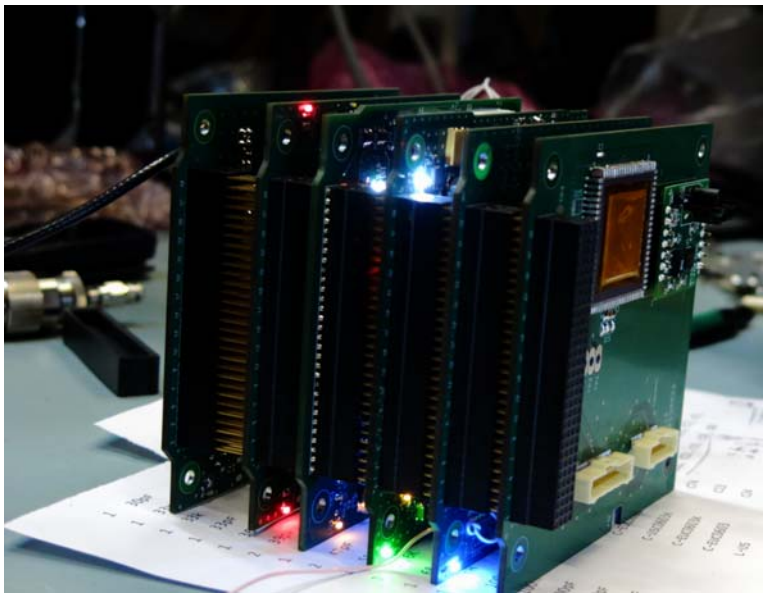

Figura 31 – Módulos de vuelo conectados

11.2.3 Integración con el Payload

No fue posible realizar la integración del Módulo de Control Principal con el hardware del Payload debido a que este último no estuvo listo durante el período de trabajo. En su lugar se utilizó una placa de desarrollo con un prototipo del software.

La integración incluyó la comunicación de solicitudes de SSTV, la transferencia de telecomandos descifrados y la recepción de mensajes para enviar a tierra, su envío y confirmación.

Capítulo 12.

Gestión del proyecto

Contenido

12.1 Organización del proyecto Antelsat

Durante el desarrollo del presente trabajo el proyecto Antelsat se organizó en varios equipos. Estos desarrollaron las siguientes tareas.

- Diseño y fabricación del hardware del módulo de gestión de energía (Gonzalo Sotta, Ignacio de León y Javier Ramos),
- Diseño y fabricación del hardware de los módulos de comunicaciones (Gonzalo Gutiérrez).
- Diseño y fabricación del hardware para los módulos de control principal y Detección y Control de Actitud. Desarrollo de los algoritmos de detección y control de actitud. (Matías Tassano).
- Diseño e implantación del software de demodulación, modulación e interfaz con el hardware para los Módulos de Comunicaciones, (Andrés Touya).
- Diseño de la primera versión del software de Módulo de Gestión de Energía y la versión de vuelo del módulo de Detección y Control de Actitud (Simón González y Pablo Yaniero).

El presente trabajo fue desarrollado en conjunto con estos equipos y en especial con la colaboración de:

- Matías Tassano quien brindó la orientación e instrucción inicial y determinó gran parte de los lineamientos del trabajo.
- Andrés Touya quien implementó y verificó la interfaz con el hardware del Módulo de Control Principal, y el Módulo de Gestión de Energía.

Andrés Touya es además coautor del protocolo de comunicación entre módulos (IMMP).

El proyecto estuvo dirigido por Juan Pechiar.

12.1.1 Modelo de desarrollo utilizado

A partir de la falta de precisión en los requerimientos detectados inicialmente, se decidió utilizar un modelo que permitiera al software crecer con la evolución del proyecto.

El modelo de desarrollo evolutivo no resultaba muy conveniente por la cantidad de grupos de trabajo y la complejidad, tampoco un modelo basado en componentes ya que al menos al inicio los requerimientos y funcionalidad resultaban muy dispares. Además, el prototipo del Módulo de Control Principal presentaba un alto grado de acoplamiento con el resto de los módulos.

No menos relevante es el hecho de que el desarrollo se trate de un sistema embebido, carente totalmente de un ambiente que permitiera la separación de procesos.

Se determinó por lo tanto utilizar un sistema operativo que permitiera separar las actividades en procesos independientes y construir estos últimos utilizando un modelo basado en componentes que posteriormente fueron efectivamente reutilizados en los restantes módulos.

Una vez que se determinó no utilizar un sistema operativo comercial sino construir uno acorde a las necesidades del proyecto, este se desarrolló utilizando un modelo evolutivo con prototipos descartables.

12.1.2 Iteraciones

Este trabajo comenzó a desarrollarse en setiembre de 2012 cuando el proyecto antecedente finalizó su trabajo culminando con la entrega del prototipo de hardware del Módulo de Control Principal.

El desarrollo se planificó con tres iteraciones y un plazo total de nueve meses, tres meses por iteración.

La primera iteración que culminó a finales de 2012 no pudo ser integrada con los restantes módulos porque estos no estaban construidos.

La segunda iteración, que debía refinar los protocolos de comunicación entre módulos, no pudo llevarse a cabo hasta casi seis meses después y estuvo limitada a la interacción con el Módulo de Comunicaciones COMM2.

Las piezas definitivas de hardware terminaron de construirse en noviembre de 2013.

La fabricación del kernel tuvo seis iteraciones y el desarrollo de los procesos del Módulo de Control Principal dos. La segunda iteración pudo comenzar a integrarse con las restantes piezas de hardware, pero la mayor parte del trabajo se realizó utilizando placas de desarrollo y simulación.

12.1.3 Plan de trabajo

El plan de trabajo tuvo muchas variaciones a lo largo del proyecto. En la figura 32 se muestra el plan de trabajo inicial.

Fueron tres los elementos que generaron las distorsiones más importantes respecto a la planificación inicial.

El primero fue la falta de integrantes. Inicialmente el trabajo iba a ser realizado por un estudiante de maestría secundado por el autor del presente trabajo. Ese estudiante abandonó el trabajo durante la primera etapa. Posteriormente hubo un segundo candidato que colaboró en el diseño pero que no participó del resto del desarrollo.

Fecha		Iteración	Hitos
2012	Setiembre	Iteración 1	
	Octubre		
	Noviembre		Prototipo 1
	Diciembre	Iteración 2	
2013	Enero		Libre
	Febrero		
	Marzo		Liberación 1
	Abril	Iteración 3	
	Mayo		
	Junio		Liberación final

Figura 32 – Plan de trabajo inicial

El segundo elemento y no menos relevante fue la demora en la construcción de las piezas de hardware definitivas. El hardware de los restantes módulos estuvo efectivamente operativo en el mes de noviembre de 2013. La integración con el Payload se realizó con una placa de desarrollo ya que el hardware de ese módulo no estuvo disponible ni siquiera en versión de prototipo.

El tercer elemento fue la falta de personal dedicado al desarrollo de software de gran parte de los restantes módulos. Recién a mediados de 2013, un equipo de estudiantes en el ámbito de su proyecto de grado comenzó a generar el software para el Módulo de Gestión de Energía.

En ese contexto, se hizo necesario construir el software para los restantes módulos para poder integrar y validar el software del Módulo de Control Principal.

A continuación se muestra la distribución del trabajo resultante. Los plazos son aproximados.

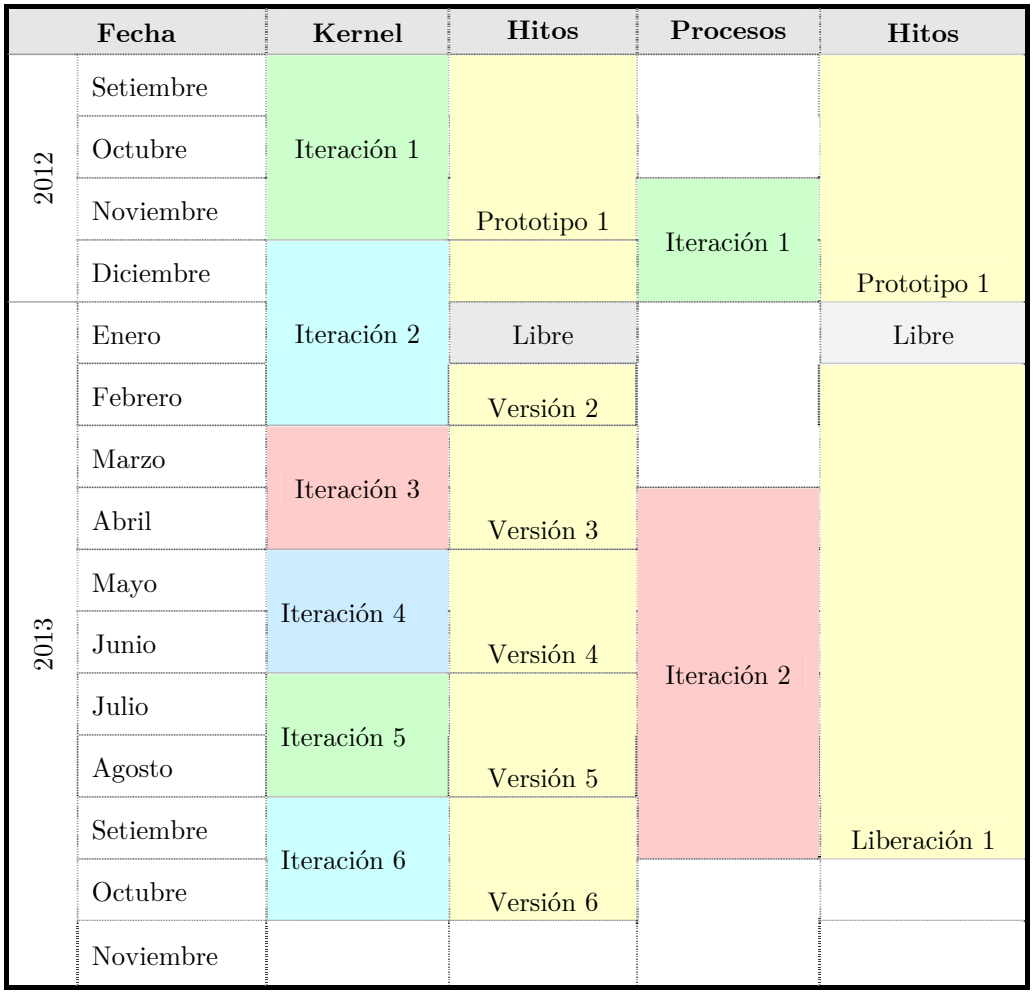

Figura 33 – Distribución aproximada del tiempo por actividad

12.2 Trabajo realizado

En esta sección se describen las actividades realizadas a lo largo del proyecto y se menciona a los integrantes del equipo que participaron.

12.2.1 Ambiente de desarrollo

- Definición del ambiente de desarrollo. Pruebas del sistema operativo base y máquina virtual de soporte. Instalación y configuración del entorno de desarrollo.

12.2.2 Arquitectura

- Definición de la arquitectura del software del Módulo de Control Principal.
- Definición de los lineamientos generales de la arquitectura de los módulos de Gestión de Energía y de Detección y Control de Actitud.

12.2.3 Kernel

12.2.3.1 Diseño del Kernel

- Diseño para el microcontrolador del Módulo de Control Principal operado con direccionamiento extendido para permitir acceder a la totalidad de la memoria FLASH.

12.2.3.2 Adaptación del Kernel para otras plataformas

- Adaptación para lograr independencia de la arquitectura.
- Adaptación para ser utilizado con direccionamiento MSP430 estándar (16 bits).

12.2.3.3 Implementación

- Implementación del prototipo y las distintas versiones del Kernel.

12.2.3.4 Verificación

- Diseño e implementación de los casos de prueba para todas las versiones.
- Verificación de todas las versiones.

12.2.3.5 Documentación

- Confección del manual y ejemplos de uso.
- Revisiones realizadas por Julio Pérez, Cecilia Rodríguez y Joaquín Born.

12.2.3.6 Entrenamiento

- Asistencia a estudiantes de Proyecto de fin de carrera del Instituto de Ingeniería Eléctrica para el uso y la inyección de fallas sobre una emulación del microcontrolador sobre FPGA (grupo a cargo del docente Julio Pérez).
- Asistencia a estudiantes de Proyecto de Grado del Instituto de Computación para la configuración y uso del Kernel.

12.2.4 Detección y corrección de errores

- Definición de los mecanismos de detección y corrección de errores
- Pruebas de implementación sobre MSP430 de código Hamming.
- Pruebas de implementación sobre MSP430 de paridad bidireccional.
- Implementación de controles basados en CRC16 y función mayoría.

12.2.5 Configuración de relojes y temporizadores

- Configuración basada en el oscilador interno del microcontrolador para utilizar las placas de desarrollo a la frecuencia de trabajo.

Trabajo realizado en conjunto con Matias Tassano para satisfacer los requisitos de los módulos MCS, ADCS y EMS:

- Definición y configuración de frecuencia de Master Clock (MCLK)
- Definición y configuración de frecuencia de Secondary Master Clock. (SMCLK)
- Definición y configuración de frecuencia de Auxiliary Clock (ACLK)
- Definición y configuración de frecuencia de I2C, SPI, y temporizador del sistema.

12.2.6 Hora

- Diseño de la representación de la hora con la colaboración de Andrés Touya en el acceso al hardware y Matías Tassano en la definición de precisión y necesidades de los distintos módulos.
- Definición del protocolo para recibir, registrar, reportar y establecer la hora recibida de tierra en MCP.
- Diseño del protocolo de negociación de la hora con la colaboración de Ignacio de León para asegurar el soporte del hardware en el módulo de gestión de energía.
- Implementación del servicio de hora para MCP en conjunto con Andrés Touya en la configuración de hardware.

12.2.7 Persistencia en FLASH

- Diseño de la metodología de persistencia de eventos y su recuperación.
- Diseño de la metodología de persistencia de los parámetros y su recuperación.

- Diseño e implementación del almacenamiento y borrado de FLASH con la asistencia de Andrés Touya en el acceso al hardware.

12.2.8 Reinicio del procesador

- Diseño de la configuración del watchdog y su incorporación al kernel
- Diseño de la metodología de reinicio del Módulo de Control Principal.

12.2.9 Protocolo IMMP

Trabajo realizado con Andrés Touya.

- Diseño e implementación del protocolo de mensajería entre módulos sobre I^2C

12.2.10 Protocolo AX25

- Implementación de la capa de enlace del protocolo con la colaboración de Juan Pechiar.
- Implementación del servicio de repetidora AX25 (DIGIPEATER) incluyendo habilitación desde tierra y anuncio en la baliza Morse.
- Verificación realizada en conjunto con Andrés Touya.

12.2.11 Cifrado y descifrado TEA

- Adaptación del algoritmo para MSP430.
- Verificación realizada en conjunto con Andrés Touya.

12.2.12 Upload en tiempo de ejecución.

- Diseño e implementación de la aplicación y mecanismos de comunicación para cargar código en los módulos luego del lanzamiento.

12.2.13 Comandos de tierra

- Definición de los comandos de tierra al satélite.
- Implementación del procesador de comandos del Módulo de Control Principal.
- Implementación de los comandos.

12.2.14 Parámetros

- Diseño de la metodología para gestión de parámetros incluyendo carga, almacenamiento, recuperación, restauración y modificación en tiempo de ejecución.

12.2.15 Eventos

- Diseño e implementación de la metodología para registrar, almacenar y recuperar los registros de eventos y datos generados por los módulos del satélite.

12.2.16 Telemetría

- Diseño e implementación del sistema para recolectar datos, emitir telemetría y suspenderla a demanda.

12.2.17 SSTV

- Diseño e implementación del protocolo de servicio de SSTV para terceros incluyendo su habilitación desde tierra y anuncio en la baliza Morse.

12.2.18 Procesos de EMS

Diseño e implementación de los procesos:

- Gestor de parámetros
- Gestor de energía
- Encendido y apagado de módulos
- Procesador de comandos
- Reporte de telemetría
- Mantenimiento de la hora
- MPPT (Maximum power point tracking)

Trabajo en conjunto con Andrés Touya, Ignacio de León y Gonzalo Sotta basado en el prototipo realizado por Simón González y Pablo Yaneiro.

12.2.19 Procesos de ADCS

Diseño e implementación de los procesos:

- Gestor de parámetros
- Procesador de comandos
- Reporte de telemetría
- Cliente de la hora

12.2.20 Ambiente de integración

- Definición del ambiente de integración y condiciones de construcción de los módulos de software.

12.2.21 Integración con el Payload

- Comunicación de solicitudes de SSTV
- Transferencia de telecomandos descifrados
- Recepción de mensajes para enviar a tierra, su envío y confirmación.

Trabajo realizado con Juan Odriozola (Antel) y Andrés Touya.

12.2.22 SASE 2013

- Participación en el Simposio Argentino de Sistemas Embebidos 2013 que aportó los detalles necesarios para el diseño de la sexta versión del Kernel.

12.2.23 Arquitectura de Software para la Estación Terrena

- Coautor del trabajo Arquitectura de Software para la Estación Terrena del proyecto Antelsat junto con Javier Alsina, Mauricio Bouza, Diego González y Agustín Arizti.
- Responsable de la revisión 1.

Este trabajo describe los lineamientos para transformar el prototipo de la estación terrena en el sistema de gestión de tierra que la misión necesita. Los temas abordados fueron: la arquitectura del software, los servicios que debe consumir la estación terrena para funcionar correctamente, las tecnologías a utilizar, el origen de los datos necesarios para la operación y los métodos de compresión de audio. El análisis se enfocó en la interacción entre la estación terrena y la base de datos, contemplando la existencia de un portal web público con capacidad de realizar consultas sobre los datos en la base.

12.2.24 Documentación del proyecto

Construcción del presente documento.

12.3 Materiales utilizados

12.3.1 Placa de desarrollo

La mayor parte del desarrollo se realizó utilizando una placa de prueba MSPFET430U5X100 con un procesador MSP430F5438A.

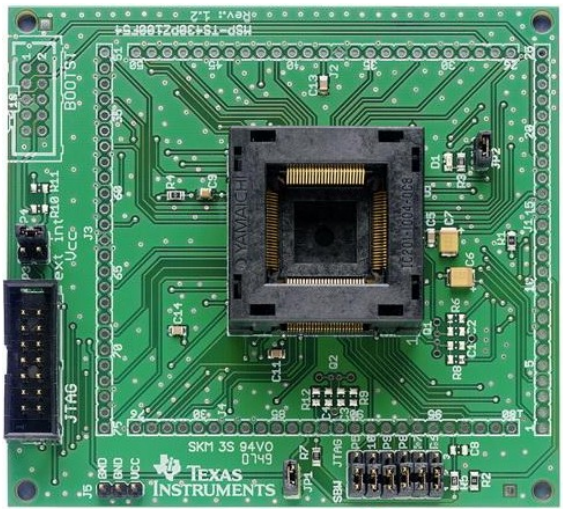

Figura 34 – Placa de desarrollo

Esta placa tiene todo lo que necesita el Módulo de Control Principal excepto el cristal de 32768 Hz necesario para generar la frecuencia de trabajo y mantener la hora del sistema.

Para utilizarla fue necesario realizar una configuración especial que permitiera emplear un oscilador interno menos preciso pero adecuado a las necesidades de desarrollo.

12.3.2 Interfaz de programación

La interfaz de programación MSP-FET430UIF permite conectar un PC con el microcontrolador. Soporta los protocolos de depuración JTAG y Spy-Bi-Wire (JTAG de 2 hilos).

Esta herramienta permite depurar los programas conectándose tanto a la placa de desarrollo como al modelo de vuelo.

Figura 35 – Interfaz de programación MSP-FET430UIF

12.3.3 Entorno de desarrollo

Todo el desarrollo se realizó utilizando el entorno de desarrollo IAR Embedded Workbench evolucionando en diferentes versiones culminando con la 5.52.

Este entorno de desarrollo incluye un gestor de proyectos, un editor y un depurador.

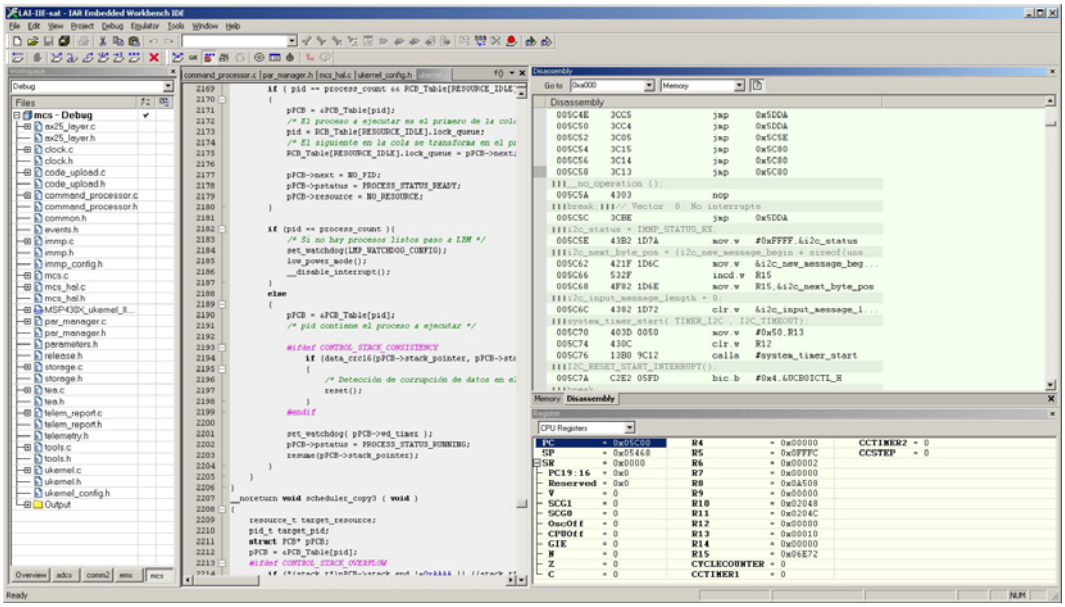

Figura 36 – Entorno de desarrollo

12.3.4 Software de virtualización

El entorno de desarrollo trabaja sobre el sistema operativo Windows por lo que fue necesario utilizar una máquina virtual para soportarlo.

Durante las primeras etapas del proceso se utilizó VMWare Player, entre las versiones 3 y 6. Posteriormente se unificaron las características de todas las estaciones de desarrollo y se pasó a utilizar Virtual Box de Oracle Corporation "Oracle VM VirtualBox" Versión 4.2.18.

12.3.5 Estación de trabajo

El proceso de desarrollo no tiene mayores requerimientos sobre el equipo de soporte. La capacidad necesaria para lograr un buen funcionamiento de la máquina virtual fue suficiente para soportar al entorno de desarrollo.

12.3.6 Licencias

El trabajo requirió una licencia para el sistema operativo Windows que fue provista por la Unidad de Recursos Informáticos y una licencia para el entorno de desarrollo que fue provista por el Instituto de Ingeniería Eléctrica.

12.4 Tiempo dedicado

El trabajo requirió un total de alrededor de 1000 horas incluyendo las actividades relacionadas con el proyecto no incluidas en este trabajo.

Tabla 9 – Tiempo dedicado incluyendo actividades relacionadas

Actividad	Horas	Porcentaje
Investigación	164	17%
Requisitos	113.5	12%
Diseño	200.5	21%
Implementación	77	8%
Verificación	116.75	12%
Integración	93	10%
Otros	116.75	12%
Documentación	95.25	10%
Total	976.75	

El tiempo se distribuyó como se muestra en la tabla 9. Se destaca el trabajo de análisis y diseño que tuvo una incidencia importante no sólo desde el punto de vista del tiempo dedicado sino de los resultados de los procesos posteriores.

Tabla 10 – Tiempo dedicado directo

Actividad	Horas	Porcentaje
Investigación	164	19%
Requisitos	113.5	13%
Diseño	200.5	23%
Implementación	77	9%
Verificación	116.75	14%
Integración	93	11%
Documentación	95.25	11%
Total	860	

El la figura 37 se muestra la distribución del tiempo a lo largo de todo el proyecto hasta noviembre de 2013 (sin incluir las actividades relacionadas).

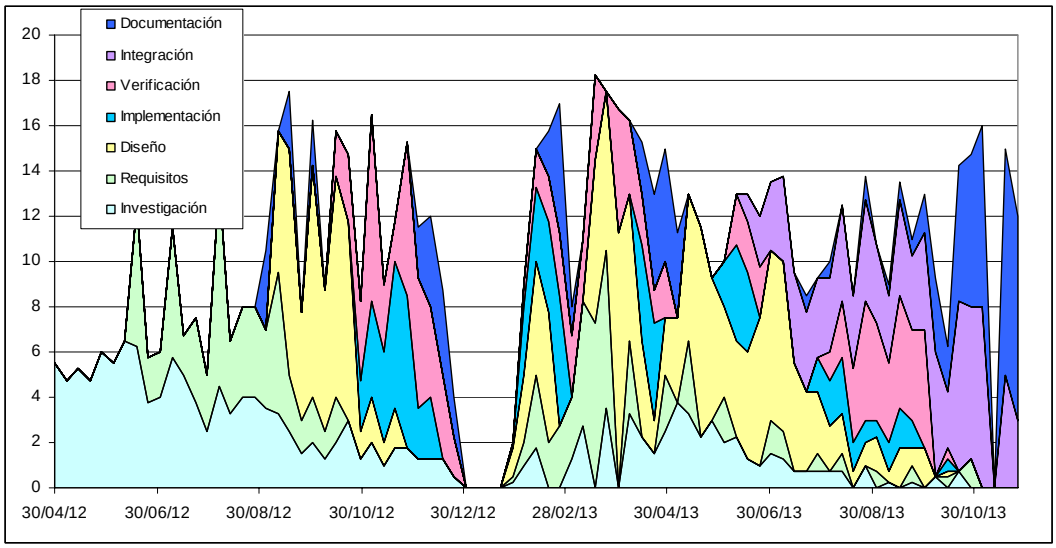

Figura 37 – Distribución del tiempo

Capítulo 13.

Conclusiones

Contenido

13.1 Conclusiones generales

El trabajo resultó muy satisfactorio. El equipo de trabajo colaboró ampliamente durante todo el desarrollo aportando requerimientos, inquietudes, analizando casos y realizando pruebas. A pesar de tratarse de un emprendimiento novedoso no resultó suficientemente atractivo como para que se sumaran más integrantes especializados en el desarrollo de software. Este riesgo fue detectado inicialmente, no obstante las medidas tomadas no fueron suficientes para evitarlo.

Para compensarlo se extendió el período de trabajo y se pospuso la construcción de este documento.

A cambio se logró contar con:

- Una plataforma unificada de desarrollo para todos los módulos.
- Una arquitectura de procesos simples y altamente desacoplados, sencillos de actualizar y verificar.
- Una implementación robusta del Módulo de Control Principal.
- La implementación de los componentes clave de los módulos de gestión de energía y detección y control de actitud.

13.2 Objetivos alcanzados

En una de las primeras reuniones, el director del proyecto señalaba refiriéndose al software para el Módulo de Control Principal: “Necesitamos construir la versión de vuelo, no tenemos tiempo para otro prototipo”. Ese objetivo se ha alcanzado y el desarrollo está siendo reutilizado para los restantes módulos.

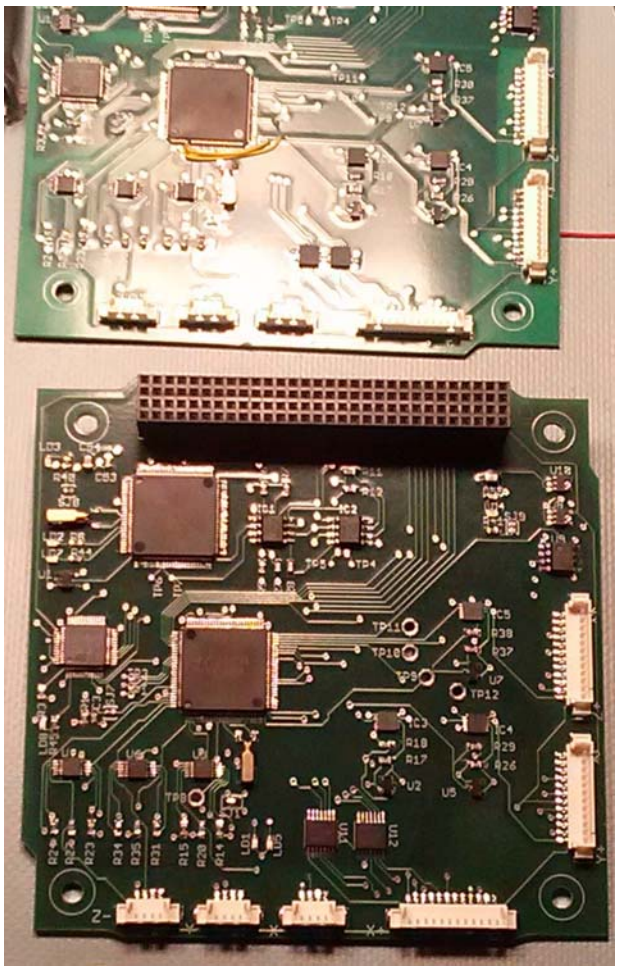

Figura 38 – Últimas dos versiones de la placa de MCS y ADCS

13.3 Colaboración con el proyecto

Más allá del desarrollo realizado, el aspecto más relevante en lo que respecta a colaboración con el proyecto parece ser el aporte de conceptos de ingeniería de software.

El resto del equipo –un conjunto de profesionales en electrónica– ha desarrollado software inclusive desde antes de que comenzara este trabajo. Es notable el cambio en su metodología al respecto, especialmente en la empleada por los que colaboraron más estrechamente.

13.4 Evaluación de la planificación

13.4.1 Distribución de tiempos

La planificación inicial estimaba necesario el trabajo de tres estudiantes con una dedicación de 500 horas cada uno.

El tiempo planificado para cada actividad difiere ampliamente del utilizado como se muestra en la figura 39. El motivo es claramente la falta de integrantes.

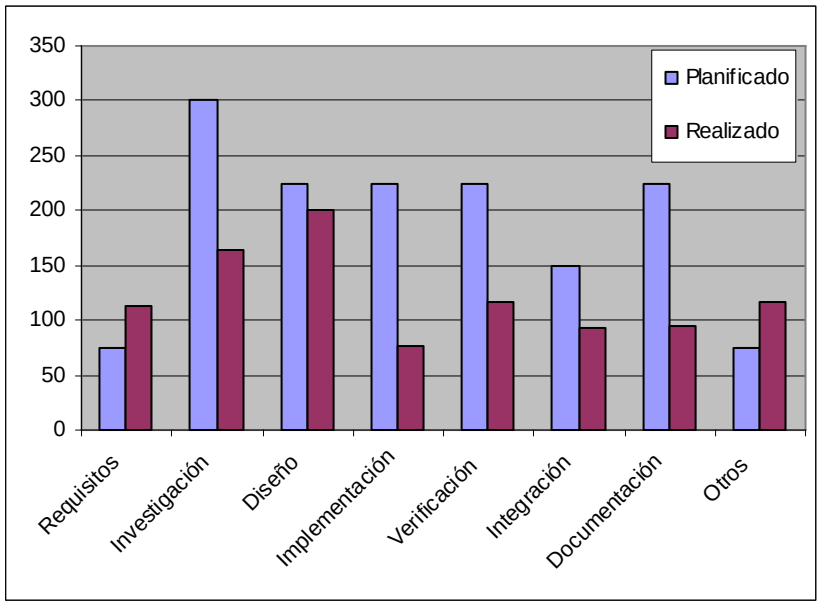

Figura 39 – Comparación entre el tiempo planificado y el utilizado (horas)

La distribución del tiempo, en cambio, fue mas precisa. En el gráfico de la figura 40 se comparan porcentualmente el tiempo estimado y el dedicado.

La definición de requisitos, el diseño y las actividades adicionales consumieron más tiempo del previsto.

En sentido contrario se destaca la implementación la cual requirió mucho menos de lo planificado.

Por último, es importante destacar que el tiempo dedicado a documentación se limitó para ganar tiempo para cubrir las faltas de personal del resto de los desarrollos de la misión, lo que lamentablemente hace que este documento no sea tan rico como debiera.

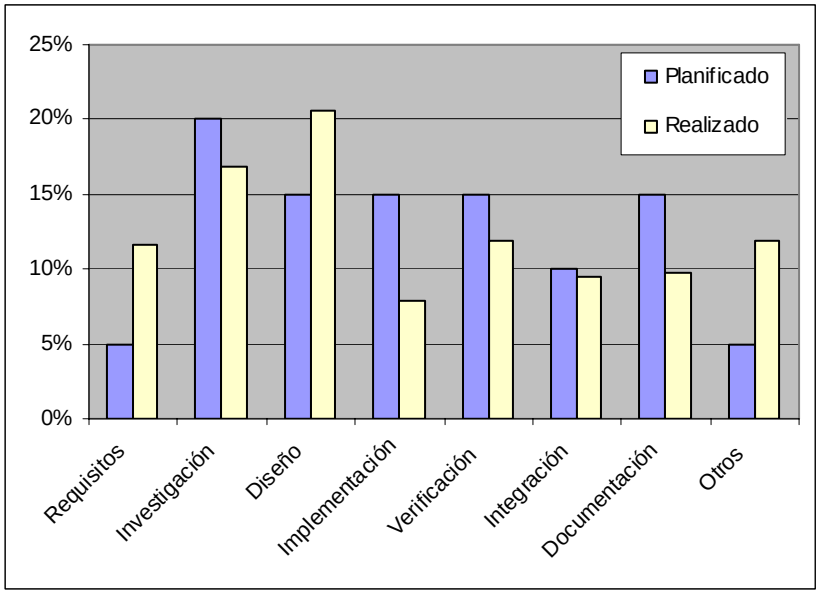

Figura 40 – Comparación entre el tiempo planificado y el utilizado (porcentual)

13.4.2 Extensión en el tiempo

Formalmente el trabajo se inició durante el primer semestre de 2012. El equipo responsable del proyecto antecedente estaba muy atrasado con el trabajo y no estuvo en condiciones de integrar a más personas. Las actividades se retrasaron debido a que ese proyecto culminó en los primeros días de setiembre de 2012.

Por causa de ese retraso en el inicio, el final del trabajo se proyectó para el mes de mayo de 2013. Sin embargo, por diferentes razones el trabajo no pudo culminarse sino hasta el mes de setiembre.

De todos modos, aunque fue necesario esperar a que finalizara la construcción del hardware y, a pesar de que la integración con el Payload no se ha completado, los resultados obtenidos generan una satisfacción personal difícil de igualar.

13.5 Lecciones aprendidas

13.5.1 Escasos recursos para el desarrollo de software

Uno de los problemas más importantes de la misión es la baja cantidad de recursos humanos dedicados al diseño y desarrollo de software.

Durante el desarrollo de este trabajo hubo una media de 1.5 estudiantes del área de software trabajando en el software en el ámbito de su proyecto de grado.

La planificación del proyecto no contempla el desarrollo del software como una actividad esencial. Los hitos están en su mayoría asociados al hardware.

Los desarrolladores del hardware, sin ninguna especialización en software, debieron construir los prototipos. Esto resultó en piezas de muy alto acoplamiento y escasa documentación e hizo necesario rediseñar y seguramente volver a tomar decisiones que tal vez ya habían sido tomadas.

Un desarrollo de estas características debería haber considerado los pormenores del software de la misma forma que consideró el desarrollo del hardware.

No obstante, algunos cambios se produjeron a partir de las necesidades del software. Entre ellos se destacan el agregado de un cristal en el Módulo de Gestión de Energía para poder mantener la hora cuando el Módulo de Control Principal esta apagado, y el agregado de un procesador ARM conectado al microcontrolador del Módulo de Detección y Control de Actitud para realizar las complejas operaciones de punto flotante.

13.5.2 Documentación temprana

Durante las primeras etapas del trabajo se determinó como muy importante la documentación. En función de esto, y en paralelo con el desarrollo, fueron construyéndose documentos y esquemas con información detallada.

La mayor parte de esta documentación no resultó útil porque los resultados de la investigación generaron cambios muy importantes entre los primeros conceptos y el resultado final.

Esto no implica que haya sido contraproducente, pero sí que frente a un escenario de tan escasos recursos hubiera sido más eficiente documentar sólo las decisiones de diseño.

13.5.3 Decisiones de diseño junto con el código

Los documentos utilizados para el diseño se ampliaron con los seudo códigos y a partir de ellos se construyeron los prototipos.

La definición de los casos de uso, las notas acerca de los casos de prueba y sus posteriores resultados, se acumularon como comentarios y evolucionaron con la implementación.

A la hora de construir este documento la fuente más importante de datos resultó estar en el propio código de la implementación de los prototipos.

13.6 Trabajo futuro

El trabajo no ha finalizado. La integración con hardware no se ha podido completar porque las piezas definitivas aun están siendo construidas y, al momento de finalización de este trabajo, el satélite no está ensamblado.

Una de las piezas fundamentales faltantes es la carga científica. El módulo conocido como PAYLOAD fue suplantado en el proceso de integración por una placa de desarrollo que emula su comportamiento.

13.6.1 Verificación del sistema completo

Una vez ensamblado el satélite es indispensable verificar el sistema completo. Esta verificación debe realizarse evaluando todas y cada una de las funcionalidades implementadas.

Es recomendable realizar una verificación usando primero herramientas de depuración sobre la totalidad del código y luego una verificación del comportamiento enviando mensajes, observando las respuestas y descargando y analizando los registros generados.

13.6.2 Mejoras sobre el desarrollo realizado

13.6.2.1 Ampliar la documentación relativa a los procesos

Existen muchas decisiones de diseño que no están presentes en este documento. Si bien gran parte de las mismas está incorporada en los comentarios del código esto no es suficiente.

Cada una de esas definiciones requirió trabajo del equipo discutiendo el diseño, analizando los casos y realizando pruebas de hardware y software.

El trabajo fue muy amplio y este documento no presenta un enfoque detallado como para abarcar todos los detalles. El camino que parece más razonable para subsanar esto es construir pequeños documentos específicos.

13.6.2.2 Agregar eventos

El desarrollo realizado relativo a este tema se enfocó en proveer los mecanismos y maximizar el espacio de almacenamiento para registrar eventos, tanto del Módulo de Control Principal, como del resto de los módulos, pero son pocos los eventos registrados.

No menos importante es agregar eventos a los restantes módulos en los que el reporte es por el momento casi nulo.

13.6.2.3 Unificar las cabeceras de IMMP

Sería conveniente unificar las cabeceras del protocolo para simplificar los procesos y eliminar problemas de alineación.

Cuando el protocolo se diseñó, uno de los objetivos era minimizar la cantidad de información transferida para no degradar la operación de procesos de altos requerimientos de tiempo del procesador. Ese requerimiento resultó ser excesivo.

Para realizar este cambio hace falta modificar todas las partes de todos los programas que utilizan comunicación entre módulos y modificar la mayor parte de las pruebas de regresión puesto que estas están construidas en su mayoría, inyectando mensajes IMMP.

13.6.2.4 Asegurar los datos del kernel

El kernel tiene la capacidad de asegurar los datos locales de los procesos pero no controla sus propios datos.

Los datos de las aplicaciones se modifican mientras estas se ejecutan y se mantienen estables mientras están suspendidas. Los datos del kernel en cambio, están permanentemente modificándose. Con cada interacción entre procesos las estructuras de control se actualizan.

Las pruebas determinaron que en más de la mitad de los casos, la corrupción de un dato individual provoca directa o indirectamente un bloqueo que termina en un reinicio.

Una de las características del procesador es que si intenta cargar el Program Counter con instrucciones desde una dirección inválida –ya sea en el espacio de direccionamiento de entrada/salida como en una dirección donde no hay memoria–, la operación obtiene el código de la instrucción “salto relativo con offset cero”. Esto provoca un bloqueo que el watchdog finaliza con un reinicio.

De todos modos, sería conveniente modificar el kernel para que controle de forma eficiente sus propios datos.

13.6.3 Investigación

13.6.3.1 Alteraciones en memoria RAM

Una de las funcionalidades implementadas permite investigar el contenido de la memoria RAM del Módulo de Control Principal.

Esta operación se realiza enviando comandos desde tierra y observando los contenidos transmitidos por el satélite como respuesta.

13.6.3.2 Alteraciones en memoria FLASH

Una de las funcionalidades implementadas genera reportes acerca del CRC del espacio de memoria FLASH utilizada para programas y constantes del Módulo de Control Principal.

Para estudiar las alteraciones se deben investigar los reportes enviados por el satélite acerca de la memoria. En caso de detectarse una alteración es posible investigar el contenido en forma detallada.

13.6.3.3 Operar sensor experimental

El Módulo de Control principal tiene conectado un sensor de temperatura experimental que se desea verificar en órbita.

Este sensor requiere un protocolo de operación que no está soportado a nivel de hardware porque su incorporación se realizó cuando el diseño de las placas estaba finalizando.

Es posible que pueda implementarse la operación totalmente controlada por software. Un equipo de trabajo está investigando las posibilidades. Una vez finalizado ese trabajo debe estudiarse el impacto sobre el resto de las piezas del Módulo de Control Principal y de ser posible, debe implementarse el control del experimento y la recolección de resultados.

13.6.4 Reutilización del desarrollo

Resultaría muy apropiado adaptar el kernel para uso general. Una de las funcionalidades es la de crear y terminar procesos en tiempo de ejecución.

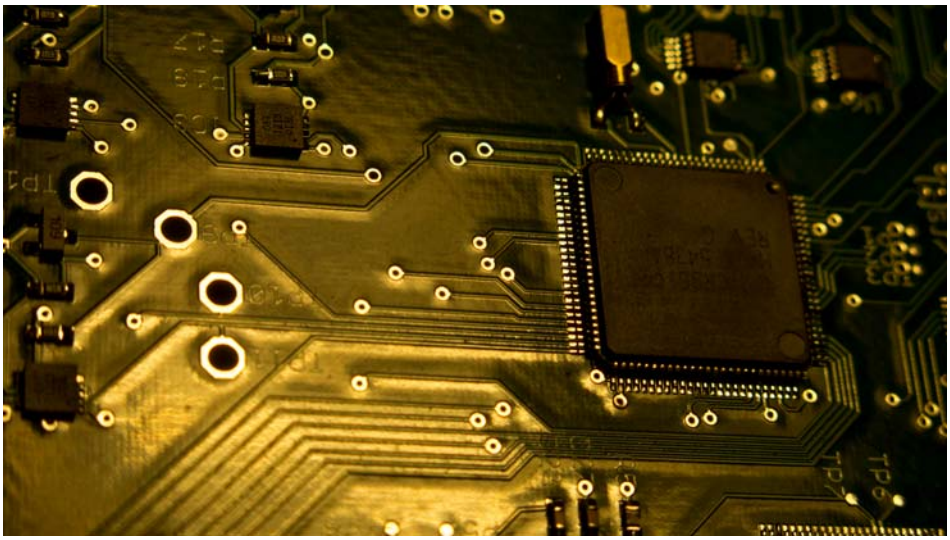

Figura 41 – El microcontrolador de MCS en la placa de vuelo

Glosario

Actitud Se denomina actitud a la orientación angular de un vehículo espacial respecto a un sistema de referencia fijo.

AX25 Protocolo de capa de enlace derivado del protocolo X25 y diseñado para ser utilizado por los operadores de radio aficionados. AX25 ocupa las dos primeras capas del modelo OSI y es responsable de la transferencia de datos (encapsular en paquetes) entre los nodos y detección de errores introducidos por el canal de comunicaciones.

Banda S Es un rango de frecuencias que va desde 1.5 a 5.2 GHz, parte de la banda de microondas del espectro electromagnético utilizada por radares meteorológicos y algunos satélites de comunicaciones.

Endian Modalidad en que se almacenan los datos en memoria. En la modalidad little-endian los elementos de menor peso se almacenan en direcciones más bajas de memoria. En la modalidad big-endian es a la inversa.

I²C Bus de comunicaciones serial diseñado por Philips. El nombre deriva de "Inter-Integrated Circuit". La velocidad puede ser de 100 o 400 Kbps. Es un bus muy usado en la industria, principalmente para comunicar microcontroladores y sus periféricos en sistemas integrados.

LDO Un regulador de baja caída de tensión o LDO es un regulador lineal de tensión continua que puede operar con una diferencia de potencial entrada-salida muy pequeña. Las ventajas de una baja caída de tensión incluyen: un voltaje mínimo de operación más bajo, mayor eficiencia de operación y menor disipación de calor.

Nadir Denominación de la intersección entre la vertical del observador y la esfera celeste en sentido contrario al cenit.

NORAD Comando Norteamericano de Defensa Aeroespacial por "North American Aerospace Defense Command"

RTOS Sistema operativo de tiempo real.

SSTV "Slow Scan TV" es un método de transmisión de imágenes, utilizado principalmente por radioaficionados, para transmitir y recibir imágenes estáticas a través de enlaces de radio.

TLE Two-Line Element es una estructura utilizada para comunicar el conjunto de datos que describen la órbita de un satélites. El TLE es un formato especificado por NORAD que consiste en una línea de título seguido de dos líneas de texto.

Bibliografía

[1] Alonsoperez V; Castro A; Zito S. “Sistema de Determinación de Actitud”, Proyecto de fin de carrera del IIE, UDELAR, abril de 2011.

[2] Ardao Martín; Fernandez Mariángel; Mato Mercedes. “Módulo de Control Principal para Cubesat”. Proyecto de fin de carrera del IIE, UDELAR, agosto de 2012.

[3] Beech, William A; Nielsen, Douglas E; Taylor, Jack (1998) “AX.25 Link Access Protocol for Amateur Packet Radio Version 2.2”, Publicado por Tucson Amateur Packet Radio Corporation. Disponible en http://www.tapr.org/pdf/AX25.2.2.pdf Visitado en noviembre de 2013

[4] Benoit, C.; Choueiri, T.; Florian, G. (2007) “SwissCube Flight Software Architecture”. Disponible en “http://ctsgepc7.epfl.ch/05%20-%20Flight%20 software/S3-BC-SE-1-0-Flight_Software_Architecture.pdf” Visitado en noviembre de 2013.

[5] California State Polytechnic University. (2009) "CubeSat Design Specification Rev. 12". Disponible en http://www.cubesat.org/images/ developers/cds_rev12.pdf Visitado en noviembre de 2013.

[6] Choueiri T.; Cosandier, B.; George, F.; Jordan, F.; Krpoun, R.; Noca, M.; Roethlisberger, G.; Scheidegger, N. (2008) “SwissCube Mission and System Overview”, Disponible en “http://ctsgepc7.epfl.ch/01%20-%20Systems%20and%20 mission%20 documents/S3-A-SET-1-0-PRR%20Mission_System_Overview.pdf”, Visitado en noviembre de 2013

[7] École Polytechnique Fédérale De Lausanne Space Center. “SwissCube” Suiza. Disponible en “http://swisscube.epfl.ch”. Visitado en noviembre de 2013.

[8] IEEE, (2008) “IEEE 754: Standard for Binary Floating-Point Arithmetic” Disponible en “http://grouper.ieee.org/groups/754”. Visitado en noviembre de 2013

[9] International Astronomical Union, (1997) “Resolution B1 on The use of Julian Dates” de “The XXIIIrd International Astronomical Union General Assembly”, Disponible en “http://www.iers.org/nn_10910/IERS/EN/Science/ Recommendations/resolutionB1.html”. Visitado en noviembre de 2013.

[10] Kelso, T. S. “NORAD Two-Line Element Sets Current Data”, Disponible en “http://celestrak.com/NORAD/elements”, Visitado en noviembre de 2013.

[11] Micrium, Inc. “µCOS/II Kernel”, Disponible en http://micrium.com/page/ products/rtos/os-ii, visitado en febrero de 2013.

[12] National Aeronautics and Space Administration, (2011) “Definition of Two-line Element Set Coordinate System”, Disponible en “http://spaceflight.nasa.gov/ realdata/sightings/SSapplications/Post/JavaSSOP/SSOP_Help/tle_def.html”, Visitado en noviembre de 2013.

[13] National Aeronautics and Space Administration. (2002) “Anomalía del Atlántico Sur”. Informe de la misión ROSAT (1990-1999) Disponible en “http://heasarc.gsfc.nasa.gov/docs/rosat/gallery/ misc_saad.html” Visitado el novembre de 2013.

[14] Needham, R.; Wheeler, D, (1994) “TEA, a Tiny Encryption Algorithm”, Computer Laboratory Cambridge University, England, Disponible en “http://link.springer.com/chapter/10.1007%2F3-540-60590-8_29”. Visitado en noviembre de 2013.

[15] NXP Semiconductors (2012) “UM10204 I²C-bus specification and user manual Rev. 5”, Disponible en “http://www.nxp.com/documents/user_manual/ UM10204.pdf”

[16] Philips Semiconductors (2003) “AN10216-01 I²C MANUAL”, Disponible en: “http://www.nxp.com/documents/application_note/AN10216.pdf”

[17] Real Time Engineers, “FreeRTOS operating system”, Disponible en http://www.freertos.org, Visitado en noviembre de 2013.

[18] SwissCube mission, “Flight housekeeping data”, Disponible en “http://ctsgepc7.epfl.ch/13%20-%20Flight%20housekeeping%20data/S3-E-Sat%20Temp%20flight%20data%20up%20to%2024-01-11.xlsx”, Visitado en noviembre de 2013.

[19] Taparle, Pierre-André, (2006) “SwissCube Command & Data Management System”, Disponible en “http://ctsgepc7.epfl.ch/04%20-%20Command%20and%20data%20management/S3-A-CDMS-1-0-CDMS.pdf”, Disponible en “http://ctsgepc7.epfl.ch/04%20-%20Command%20and%20data%20management/ S3-B-CDMS-1-0-CDMS.pdf”. Visitado en noviembre de 2013.

[20] Texas Instruments Incorporated (2002) “Mixing C and Assembler with the MSP430” Disponible en “http://www.ti.com/lit/an/slaa140/slaa140.pdf” Visitado en noviembre de 2013.

[21] Texas Instruments Incorporated (2004) “CRC Implementation with MSP430” Disponible en “http://www.ti.com/lit/an/slaa221/slaa221.pdf” Vistado en novembre de 2013.

[22] Texas Instruments Incorporated (2006) “MSP430 Software Coding Techniques” Rev. A. Disponible en “http://www.ti.com/lit/an/slaa294a/slaa294a.pdf” Visitado en noviembre de 2013.

[23] Texas Instruments Incorporated (2008) “MSP430 Flash Memory Characteristics” Publicado como slaa334a, Disponible en “http://www.ti.com/lit/an/slaa334a/ slaa334a.pdf” Vistado en noviembre de 2013

[24] Texas Instruments Incorporated (2012) “MSP430F5438A Device Erratasheet” Disponible en” http://www.ti.com/lit/er/slaz290/slaz290.pdf” Visitado en noviembre de 2013.

[25] Texas Instruments Incorporated (2012) “MSP430F543xA, MSP430F541xA Mixed Signal Microcontroller (Rev. C)” Publicado como “slas655C”. Disponible en “http://www.ti.com/lit/gpn/msp430f5438a” Visitado en noviembre de 2013.

[26] Texas Instruments Incorporated (2013) “MSP430F5438A Device Erratasheet (Rev. D)” Publicado como “slaz290d” Disponible en “http://www.ti.com/litv/pdf/slaz290d” Visitado en noviembre de 2013.

[27] Texas Instruments Incorporated (2013) “MSP430x5xx and MSP430x6xx Family User's Guide (Rev. L)” Publicado como “slau208l”. Disponible en “http://www.ti.com/litv/pdf/slau208l” Visitado en enero de 2013.

[28] Texas Instruments Incorporated “Real-time operating system SYS/BIOS”, Disponible en http://processors.wiki.ti.com/index.php/Category:SYSBIOS, Visitado en noviembre de 2013.