Universidad de la República
Facultad de Ingeniería

Proyecto de grado

Desarrollo de software para CubeSat
Módulo de Gestión de Energía
Módulo de Determinación y Control de Actitud

Autores:
Simón González
Pablo Yaniero

Tutores:
Juan Pechiar - Instituto de Ingeniería Eléctrica
Ariel Sabiguero - Instituto de Computación

13 de Enero de 2015

Agradecimientos

A Juan Pechiar, por darnos la oportunidad de trabajar en este proyecto y confiar en nosotros.

A Ariel Sabiguero, por la tutoría de nuestro proyecto.

A nuestras familias y amigos que creyeron en nosotros, especialmente Arianne Palau y Liber Dovat por su colaboración en la revisión de este texto.

A todos los integrantes del equipo de AntelSat de Facultad de Ingeniería, que estuvieron con nosotros a lo largo de todo el proceso: Gustavo De Martino, Matías Tassano, Ignacio de León, Gonzalo Sotta, Gonzalo Gutiérrez, Andrés Touyá, Javier Ramos y Ricardo Rezzano.

Y por supuesto a ANTEL y la Facultad de Ingeniería sin las cuales nada de esto habría sido posible.

Resumen

Este documento describe el trabajo realizado por nuestro grupo en el proyecto AntelSat, en el contexto del proyecto de fin de carrera de Ingeniería en Computación. En primer lugar se trabajó en el desarrollo del software del Módulo de Gestión de Energía, incluyendo análisis de requerimientos, implementación y verificación de un prototipo. En segundo lugar se trabajó en el desarrollo del software del Módulo de Determinación y Control de Actitud, incluyendo análisis de requerimientos, implementación del software del coprocesador, implementación de un prototipo del software del procesador principal, implementación del protocolo de comunicaciones entre ambos procesadores y verificación de todos los componentes anteriores. Los capítulos subsiguientes detallan el trabajo realizado y los resultados obtenidos.

Contenido

Índice de figuras

Índice de cuadros

Introducción

"El espacio... la frontera final."

Star Trek

El proyecto AntelSat[13][4] es un esfuerzo conjunto de la Facultad de Ingeniería de la Universidad de la República y la empresa de telecomunicaciones Antel. Su objetivo es construir y operar un satélite de clase CubeSat. Ésta es la primera experiencia en desarrollo de sistemas espaciales en el país. Es entonces un proceso de aprendizaje, generación de nuevas capacidades, y colaboración academia-industria. Se trata de la continuación de trabajos de investigación y desarrollos realizados en el Instituto de Ingeniería Eléctrica a partir de diciembre de 2006.

3.1. Motivación

La construcción del satélite del proyecto AntelSat ha implicado el trabajo de decenas de personas pertenecientes a Facultad de Ingeniería, Antel y Escuela Universitaria Centro de Diseño, desempeñando tareas de definición, diseño y construcción del hardware y software asociado. Dentro de Facultad de Ingeniería el proyecto ha sido llevado a cabo por el Instituto de Ingeniería Eléctrica (IIE). El equipo del IIE conformado por Ingenieros Eléctricos con especialización en Electrónica concentró sus esfuerzos en las tareas de desarrollo del hardware, siendo su formación idónea para tal trabajo. Para el software sin embargo, se solicitó la colaboración del Instituto de Computación mediante la coordinación de proyectos de grado. El primer proyecto en este contexto fue la tesis del Ing. Gustavo De Martino [9]. Nuestro trabajo es el segundo proyecto de grado del Instituto de Computación, siendo el principal objetivo continuar aportando en el desarrollo del software requerido por el satélite.

3.2. Objetivo

ANTELSAT está dividido en dos partes:

- *Aviónica*: el vehículo. Desarrollado por Facultad de Ingeniería.
- *Carga útil* o *payload*: el pasajero. Desarrollado por ANTEL.

La construcción de un satélite tipo CubeSat es una tarea compleja que requiere la solución de múltiples problemas. Dado que el satélite es un dispositivo que debe funcionar en forma autónoma, se necesita controlar el funcionamiento de los paneles solares a efectos de lograr la máxima extracción de energía posible. Las hostiles condiciones del espacio pueden causar fallas en el hardware como cortocircuitos que dejen fuera de servicio un componente durante un tiempo. Se necesita poder detectar estas situaciones e intentar poner en funcionamiento los componentes en falla. Los anteriores problemas forman parte de lo que se entiende como *Gestión de Energía* y es una de las funciones más críticas del satélite, de la cual depende todo el resto.

Otros problemas son los asociados al movimiento del satélite. En el ambiente del espacio el mismo gira libremente. Sin embargo parte del instrumental contenido en la carga útil requiere que el satélite esté orientado en una cierta posición (particularmente las cámaras). Lo anterior implica controlar la orientación del satélite de forma de colocarlo con las cámaras mirando hacia la Tierra (esta orientación

de denomina la *actitud* del satélite). Los problemas asociados son lo que se denomina *Determinación y Control de Actitud*, función necesaria para el correcto funcionamiento del instrumental científico presente en la carga útil.

El objetivo de nuestro proyecto es colaborar en la construcción de los componentes de la aviónica vinculados a Gestión de Energía y Determinación y Control de Actitud a través de la construcción del software requerido.

Teniendo en consideración el tamaño del problema y las limitaciones de tiempo existentes, el alcance del proyecto fue modificado a lo largo de su desarrollo. Los resultados esperados finales son los siguientes:

1. Desarrollo de un prototipo del software de gestión de energía.
2. Integración del software de cálculo para control de orientación.

El punto 2 implica también el desarrollo de protocolos de comunicación que permitan ejecutar el software de cálculo.

3.3. Organización del documento

En el capítulo 4 se presenta el Estado del Arte, donde se brinda una introducción al proyecto AntelSat explicando sus características y los trabajos que se han realizado dentro del mismo. Este capítulo desarrolla en mayor profundidad los conceptos esbozados en los párrafos anteriores como ser Gestión de Energía y Determinación y Control de Actitud. A su vez, se provee una síntesis de los aspectos más relevantes abordados en cada uno de los proyectos de grado precedentes, los cuales forman la base del desarrollo aquí presentado. El texto de este capítulo está preparado para poder ser entendido en su mayor parte por un lector sin formación en Ciencias de la Computación.

El resto de este texto es de naturaleza más específica y está orientado a un lector con formación técnica o de Ingeniería. El capítulo 5 describe el trabajo realizado en el Módulo de Gestión de Energía, detallando cada una de las etapas del desarrollo, desde la definición de los objetivos, pasando por el diseño, la solución desarrollada y las pruebas realizadas.

Análogamente, el capítulo 6 describe lo desarrollado para el Módulo de Determinación y Control de Actitud, destacando el diseño de los protocolos de comunicación y el trabajo de verificación realizado.

El capítulo 7 da un panorama del proceso de desarrollo, describiendo la metodología de trabajo, las herramientas utilizadas y brindando información estadística.

El capítulo 8 ofrece un resumen del contenido de los capítulos anteriores.

Finalmente en los capítulos restantes se presentan las conclusiones del trabajo realizado.

Estado del Arte

En esta sección, previo a la introducción del trabajo del presente proyecto, se realiza una introducción a los proyectos CubeSat en general, para luego brindar una reseña de los proyectos precedentes asociados a AntelSat.

4.1. Cubesat

El proyecto Cubesat es llevado a cabo por la Universidad Politécnica de California (Cal Poly) y la Universidad de Stanford desde 1999. Su propósito es proveer un estándar para diseñar pequeños satélites, reduciendo costos y tiempo de desarrollo, dando acceso al espacio a pequeñas cargas científicas.

Este estándar, llamado "CubeSat Design Specification"[6], define un módulo CubeSat como un cubo de 10 cm con una masa de hasta 1.33 Kg. que debe cumplir una serie de especificaciones relativas a la estructura, materiales y comportamiento.

La puesta en órbita de un CubeSat se realiza incluyendo el mismo como carga secundaria en lanzamientos de otros sistemas. La universidad de California desarrolló un dispositivo para el despliegue de los Cubesat en la etapa final de la puesta en órbita, denominado P-POD (*Poly-PicoSatellite Orbital Deployer*). Éste puede alojar hasta tres módulos CubeSat. Se trata de un sistema mecánico que incluye una puerta y un mecanismo de resortes donde se alojan los módulos para su transporte.

4.2. Proyectos anteriores

Los trabajos para desarrollar un satélite comenzaron a finales de 2006 con el proyecto LAÍ [13].

Figura 4.1. Cubesat 2U. Autor de la imagen: ISIS - Innovative Solutions In Space B.V.

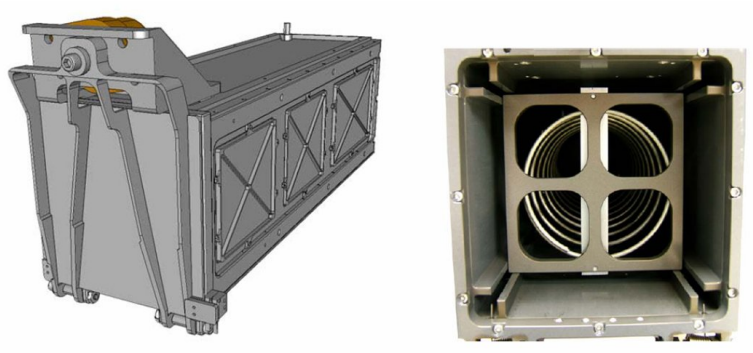

Figura 4.2. P-POD. Autor de la imagen: Cubesat.org

Los primeros desarrollos estuvieron vinculados a Globos Satelitales, sistemas autónomos suspendidos de un globo de tipo meteorológico que permiten realizar ensayos a gran altura. El trabajo "Telemetría para Globosats"[27] llevado a cabo entre 2007 y 2008 alcanzó el objetivo de desarrollar el sistema de telemetría para Globos Satelitales. En este proyecto se desarrolló el 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.

El proyecto "Sistema de gestión de energía para un satélite" [8] se desarrolló entre 2009 y 2010 con el objetivo de garantizar el suministro energético del satélite durante su vida útil.

De 2010 a 2011 se desarrollaron dos proyectos: "Módulo de determinación de actitud" [2], que determina los instrumentos a utilizar e implementa el prototipo de hardware y software y el proyecto "Sistema de telemetría, telecomando y control" [16].

En el período entre 2011 y 2012 se desarrolló el proyecto "Módulo de control principal y control de actitud" [5], durante el que se construyó el prototipo del software y la placa electrónica que implementaría los módulos ADCS (Detección y control de actitud) y MCS (Control principal). Paralelamente se llevó a cabo el proyecto "Estación terrena" [3].

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 de desarrollo nacional: AntelSat. Para el desarrollo del proyecto, Antel aporta capital y se ocupa de la carga útil del satélite y parte de las comunicaciones.

4.3. Estructura y subsistemas de AntelSat

El satélite consiste en un Cubesat clase 2U, con cinco caras dedicadas a paneles solares. La estructura completa con las antenas sin desplegar ocupa un espacio de 10 x 10 x 20 cm. La cara restante contiene dos cámaras, una en el espectro de la luz visible y otra infrarroja, junto con una antena direccional plana de alta frecuencia que se utiliza para transmitir las imágenes de alta resolución. Existen además antenas para la recepción de VHF y UHF que se despliegan luego de la puesta en órbita. Cada cara cuenta con un sensor de luz o fotodiodo. Tres de las caras (una por cada eje) disponen de giróscopos capaces de medir la velocidad angular y magnetorquers que hacen posible realizar correcciones de actitud.

El conjunto de placas electrónicas que conforman el satélite se encuentran conectadas por un bus I^2C de transmisión de datos.

La carga científica es el componente que contiene los sensores y sistemas de procesamiento de imágenes. Ocupa la mitad del volumen del satélite y es desarrollado por Antel.

El Módulo de Gestión de Energía (o EMS por sus siglas en inglés) contiene los paneles solares, las baterías y una placa de control. Sus cometidos son administrar la alimentación a los demás módulos, protegerlos de sobre consumos, obtener el mayor rendimiento posible de los paneles solares, enviar la baliza y desplegar las antenas.

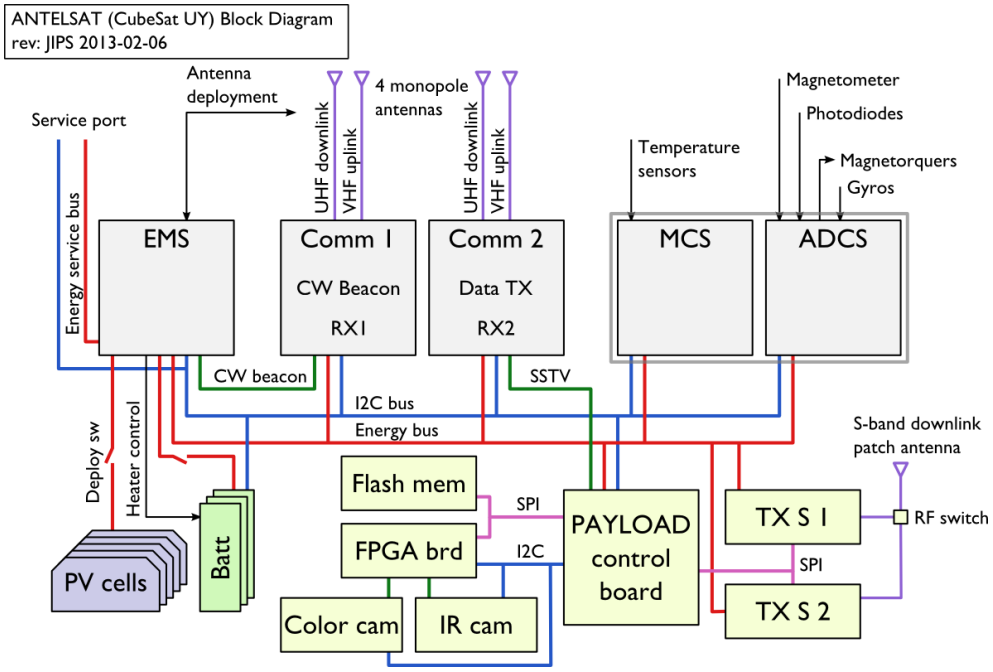

Figura 4.3. Diagrama de bloques de ANTELSAT. Autor de la imagen: Juan Pechiar

Existen dos módulos de comunicaciones independientes, COMM1 y COMM2, con receptores que trabajan a frecuencias diferentes. Los mismos realizan demodulación por software de las señales recibidas por las antenas. La recepción de datos se realiza detectando tramas AX25.

El Módulo de Determinación y Control de Actitud (o ADCS por sus siglas en inglés) contiene el sistema de control que se encarga de estabilizar el giro del satélite y posicionar la cámara de la carga científica apuntando hacia el Nadir.

El Módulo de Control Principal (o MCS por sus siglas en inglés) se encarga de interpretar y procesar los mensajes recibidos por el satélite, gestionar los parámetros de todos los módulos, mantener la hora del sistema, registrar los eventos y el estado de salud del satélite.

En el resto de este documento se nombrarán los módulos por sus siglas en inglés que son de manejo usual en el contexto de los Cubesat.

4.4. Determinación de actitud

En Abril del 2011, se finalizó el proyecto de fin de carrera "Sistema de Determinación de Actitud"[2]. Este proyecto tuvo como objetivo implementar y probar un subsistema que pudiera determinar la actitud de un CubeSat con un error menor a 10°en cada eje. En esta sección se describen las tareas realizadas en dicho trabajo.

4.4.1. Definición del problema

El movimiento de un cuerpo rígido en el espacio presenta seis grados de libertad. La posición de un punto del cuerpo se describe con tres coordenadas. La orientación se describe por tres coordenadas adicionales. En general, la traslación y la actitud (orientación) de un satélite de órbita baja pueden ser analizadas en forma independiente. Se puede asumir que la ley horaria que describe la traslación del satélite es conocida, y puede ser utilizada como un dato en el sistema de determinación de actitud. La determinación de la actitud del satélite se refiere al cálculo de su orientación con respecto a un sistema de referencia inercial o un objeto fijo como, por ejemplo, la Tierra o el Sol.

4.4.2. Sensores utilizados

La elección de los sensores usados para resolver este problema estuvo sujeta a restricciones de tamaño, consumo, masa y precisión, entre otras. Se evaluaron distintos tipos de sensores como alternativas, descartando algunos por potencia consumida o masa. Se estudiaron proyectos CubeSat de otras universidades, observando que la gran mayoría usa una combinación de magnetómetro y sensor de radiación solar. La principal ventaja de este sistema es su reducido consumo y masa. Se decidió usar sensores de radiación solar y magnetómetros para la determinación de actitud. Los sensores de sol utilizados en el AntelSat son seis fotodiodos colocados en las caras del satélite. Un fotodiodo es un dispositivo sensible a la luz que entrega una corriente de salida que varía con el ángulo de incidencia de la luz sobre su superficie.

4.4.3. Sistemas de referencia

Durante la determinación de actitud se procesan las posiciones, direcciones y velocidades de varios elementos como, por ejempo, el Sol, el campo mágnetico o el propio satélite. Este proceso requiere la conversión de las distintas medidas entre varios sistemas de referencia a fin de poder aplicar los algoritmos necesarios. Estos sistemas se pueden agrupar en dos: terrestres y del satélite.

Sistemas de referencia terrestres

Los sistemas de referencia terrestres tienen su centro en el centro de la Tierra.

- Sistema ECI (*Earth-Centered Inertial*): sistema inercial con el eje x apuntando hacia el Equinoccio Vernal (punto de corte del Plano Ecuatorial con la Eclíptica), el eje z apuntando hacia el Polo Norte geográfico y el eje restante determinado por la regla de la mano derecha. La Figura 4.4 ilustra el sistema ECI.
- Sistema ECEF (*Earth-Centered, Earth-Fixed*): sistema de referencia fijo con respecto a la Tierra (rota junto con ella). El eje x apunta hacia el punto de coordenadas (0,0), es decir, hacia el punto donde el Meridiano de Greenwich se intersecta con el Plano Ecuatorial. El eje z se encuentra junto con el eje de rotación de la Tierra y el eje y se obtiene con la regla de la mano derecha.

Nota sobre el sistema ECI: dado que el eje de rotación de la Tierra no es fijo, sino que varía sutilmente su orientación (movimiento de *precesión*), cabe hacer una aclaración. El sistema ECI usado como referencia en este trabajo suele llamarse también *true equator, mean equinox* (TEME). Esto significa que el eje z está alineado con el polo norte verdadero (instantáneo) y el eje x está alineado con la posición promedio del Equinoccio Vernal. Este es el criterio seguido por el Comando Norteamericano de Defensa Aeroespacial (NORAD) en el formato TLE (*Two Line Element*) [7] y es en el cual se basa el algoritmo SGP4 de propagación de órbita (ver sección 4.4.7).

Sistemas de referencia del satélite

Estos sistemas tienen su centro de coordenadas en un punto del satélite.

- SCB (*Spacecraft Body*): su centro se sitúa en el centro de masa del satélite. Para definir los ejes del sistema se enumeraron las caras del satélite y se tomó como eje x al eje que apunta hacia la cara 1, como eje y al eje que apunta hacia la cara 2 y como eje z al eje que apunta hacia la cara 3 (ver Figura 4.6). Los ejes están fijos con respecto al satélite.
- NED (*North East Down*): sistema no inercial con su origen en el centro de gravedad del satélite. Sus ejes se orientan según las direcciones geodésicas definidas por la superficie terrestre. El eje x apunta al Norte en dirección polar, el eje y apunta al Este siguiendo la curva de latitud y el eje z apunta hacia el centro de la Tierra. Ver Figura 4.7.
- Orbital: este sistema de referencia es usado para los cálculos de actitud. Consta de un eje que apunta hacia el centro de la Tierra, un eje que es tangencial a la órbita (paralelo a la velocidad) y un tercero que es el producto vectorial de los dos anteriores.

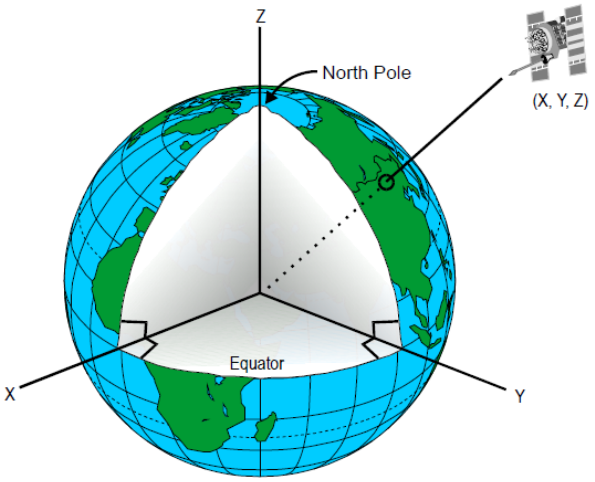

Figura 4.4. Sistema de referencia ECI. Autor de la imagen: *U.S. Department of Transportation Federal Aviation Administration - Airway Facilities Division*

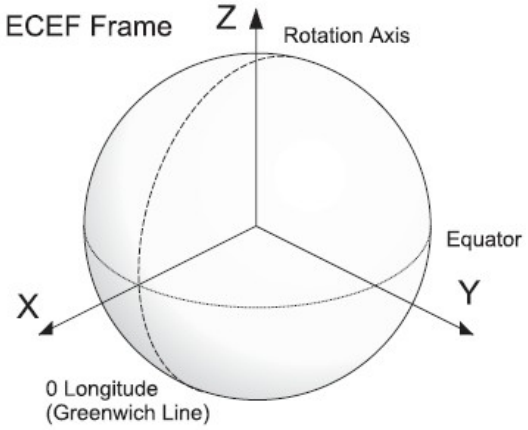

Figura 4.5. Sistema de referencia ECEF. Imagen de dominio público.

4.4.4. Representación de la actitud

La actitud de un cuerpo rígido se representa mediante la transformación de rotación que lleva del sistema SCB al sistema de referencia Orbital. Existen varias maneras de representar esta transformación.

Matriz de cosenos directores: Es la matriz asociada a la transformación de rotación. Cada una de sus entradas es el ángulo formado por uno de los ejes del sistema de referencia del cuerpo con uno de los ejes del sistema de referencia fijo.

Ángulos de Euler: Describen una rotación como la composición de tres rotaciones sucesivas realizadas en un orden dado. Cada ángulo corresponde a la rotación sobre un eje. Los ángulos de Euler son frecuentemente llamados *Roll*, *Pitch* y *Yaw*.

4.4.5. Elección del método de determinación de actitud

De acuerdo a estos términos, determinar la actitud del satélite es equivalente a encontrar la matriz de rotación que transforma el sistema de referencia solidario al satélite en el sistema ECI. Los sensores del AntelSat permiten obtener el vector de dirección del Sol y el vector de campo magnético en el

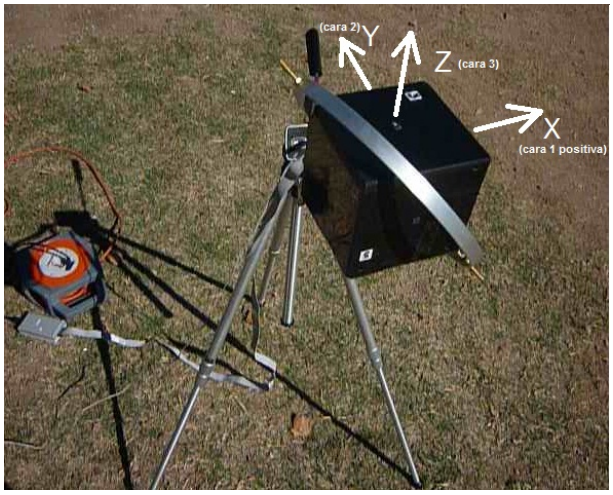

Figura 4.6. Orientación de los ejes del sistema SCB en el ANTELSAT. Autores de imagen: Victoria Alonsoperez, Adriana Castro, Santiago Zito

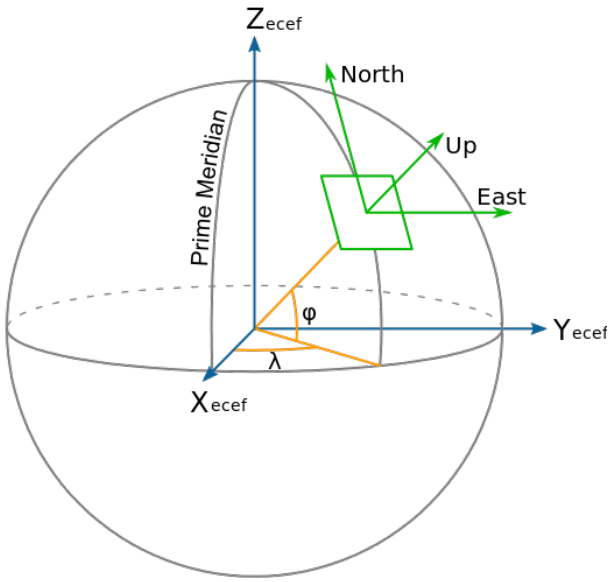

Figura 4.7. Sistema de referencia NED. Imagen de dominio público.

sistema SCB. Asimismo se pueden obtener las representaciones de estas magnitudes en el sistema ECI. El problema consiste entonces en encontrar la matriz de rotación a partir de estos cuatro vectores.

Se usó el método TRIAD [38] para la determinación de actitud. Los motivos para esta decisión fueron la sencillez de implementación y prueba. Este método consta de las siguientes etapas:

- Medición del campo magnético: se usa el magnetómetro para obtener el vector de campo magnético en el sistema SCB.
- Medición del vector de Sol: los valores de los seis fotodiodos se procesan para obtener el vector de dirección del Sol en el sistema SCB.
- Estimación de la posición: se calcula una aproximación de la posición del satélite en base a la fecha juliana. Se denomina fecha juliana a la cantidad de días transcurridos desde el primero de Enero de 4713 AC, sumada a una parte fraccional que indica la proporción del día transcurrido.
- Cálculo del vector de campo magnético: a partir de un modelo del campo magnético de la Tierra y la posición calculada anteriormente se calcula el vector de campo magnético en el sistema ECI.
- Cálculo del vector de sol: a partir de la fecha juliana se calcula el vector de Sol en el sistema ECI.

- Determinación de actitud: en base a los dos vectores expresados en el sistema SCB y los dos vectores en el sistema ECI, se calcula la orientación de un sistema con respecto al otro por el método TRIAD.

  La descripción detallada del método TRIAD puede verse en [38].

4.4.6. Elección del microcontrolador

La elección del microcontrolador utilizado en el ADCS se realizó tomando en cuenta las siguientes restricciones:

Comunicación: el hardware debe soportar comunicación I^2C en modo multimaestro.

Interacción con sensores: los fotodiodos tienen salida analógica por lo que se debe usar un ADC para convertir la señal a formato digital. Es por lo tanto conveniente que el microcontrolador tenga un ADC interno. Asimismo el hardware debe soportar comunicación por bus SPI ya que esta es la interfaz con el magnetómetro.

Watchdog: un watchdog es un dispositivo que consta de un contador que se incrementa continuamente, y reinicia el sistema cuando alcanza un valor determinado. El software debe reiniciar este contador periódicamente como parte de su funcionamiento normal. Si un error de programación causa que la aplicación quede bloqueada, el watchdog reinicia el sistema, lo cual provee un mecanismo automático de recuperación ante fallas. Se buscó que el microcontrolador contara con un watchdog.

Costo: para cumplir con el presupuesto del proyecto se busca que el microcontrolador tenga el costo más reducido posible dentro de las alternativas que cumplen las demás restricciones.

Depuración: se debe contar con una interfaz de depuración como por ejemplo UART, o JTAG.

Alimentación: el hardware debe poder alimentarse con una tensión de 3.3V.

Temperatura: se espera que el satélite trabaje en un rango de temperatura de -30°C a 50°C. El hardware debe soportar esto.

Se evaluaron distintas alternativas disponibles en el mercado teniendo en cuenta los factores anteriores. Se decidió usar el microcontrolador MSP430F5438 de Texas Instruments [36]. Se trata de un chip diseñado específicamente para aplicaciones de bajo consumo, que cuenta con un procesador de 16 bits de arquitectura RISC.

4.4.7. Determinación de la posición del satélite

Un dato esencial en el algoritmo de determinación de actitud es la posición actual del satélite con respecto a un sistema de referencia inercial. Este cálculo se realiza por medio de un modelo orbital que permite determinar la posición del satélite en un instante dado a partir de una descripción de la órbita. La misma se describe por medio de un conjunto de parámetros denominados elementos orbitales, que son transmitidos desde la estación terrena en un formato estándar denominado TLE especificado por el NORAD.

Los elementos orbitales usados corresponden a un modelo de órbita Kepleriano, en el cual se asume que la forma y orientación de la órbita son estáticos. Esto es una simplificación, por lo cual para calcular la posición del satélite es necesario realizar correcciones para aproximar el cambio que la órbita sufre a lo largo del tiempo.

La posición se calcula por medio del modelo SGP4. Este permite propagar la posición con respecto al sistema ECI a partir de un TLE y el tiempo transcurrido desde que se generó el mismo. Esto hace posible que el satélite conozca su posición sin requerir un nuevo TLE en cada cálculo, lo cual requeriría comunicación con la Tierra en forma continua.

4.4.8. Modelo del campo magnético terrestre

Para calcular el vector de campo magnético se utilizó IGRF [24], una serie de modelos matemáticos del campo magnético de la Tierra y de como este varía año a año. Los parámetros de entrada del modelo

incluyen la latitud, longitud y la fecha en la cual se desea conocer el vector. Como resultado, entre otras cosas, produce el vector de campo magnético en el sistema de referencia NED.

El modelo IGRF fue publicado por la IAGA (*International Association of Geomagnetism and Aeronomy*) una organización que se dedica al estudio del campo magnético terrestre. Una implementación del modelo está disponible en Internet con su código fuente en C, libre para ser utilizada.

4.4.9. Modelo del vector de Sol

La rutina que determina la posición del Sol se basa en el trabajo de David Vallado [37], que presenta un algoritmo por el cual se puede calcular la posición del Sol en coordenadas ECI con una precisión de 0.01°a partir del día juliano. El algoritmo de Vallado calcula el Vector de Sol en 3 ejes y otros datos que no son relevantes para determinar la actitud.

4.4.10. Estados del Sistema de Determinación de actitud

Se definieron tres estados posibles del sistema de determinación de actitud en base al tiempo transcurrido desde que se recibió el último TLE, la velocidad de giro y el consumo de energía deseado.

- **Sin TLE**. Este estado significa que todavía no se ha recibido ningún TLE o se ha pasado más de 7 días sin recibir uno. En este estado no es posible calcular la posición y por ende no es posible determinar la actitud.
- **Ahorro de Energía**. Cuando el sistema de determinación de actitud se encuentra en este estado el programa se encuentra detenido y el microcontrolador está en un modo de bajo consumo. Es posible salir de este estado a través de una interrupción. El programa está en modo de ahorro de energía cuando no está realizando cálculos, haciendo mediciones o transmitiendo datos. También puede entrar en este modo por pedido del Módulo de Control Principal.
- **Spin**. El satélite se encuentra en estado Spin cuando gira fuera de control. Más específicamente, se considera que el satélite está en Spin cuando el cambio que sufre la orientación durante el tiempo que lleva determinarla y transmitirla al sistema de control de actitud es mayor a 10° en alguno de sus ejes. Se estimó en 1,5 minutos el límite mínimo de tiempo que toma realizar la determinación de actitud y transmitirla al sistema de control. Según esta estimación, el estado Spin corresponde al punto en que la velocidad angular es mayor a 0.00199 rad/s (0.019 rpm). En esta situación ADCS deberá actuar para detener el giro (esto no forma parte del alcance del proyecto de determinación de actitud). El algoritmo desarrollado en este proyecto demora 18.13 s en determinar la actitud cuando el satélite se encuentra en estado Spin.
- **Estable**. El satélite está en estado estable cuando su velocidad angular no supera los 0.00199 rad/s en ninguno de sus tres ejes. Cuando se está en este estado la actitud cambia más lentamente, por lo que las medidas de los sensores pueden hacerse a intervalos más largos. La frecuencia con la cual se realizan las medidas en estado Estable es indicada a través de un mensaje I^2C.

4.4.11. Software existente

En esta sección se describe el software desarrollado en el proyecto "Sistema de Determinación de Actitud"[2]. El programa desarrollado en C consiste en una máquina de estados que implementa el algoritmo de determinación de actitud junto con los modelos necesarios. El diseño original previa comunicación I^2C para intercambio de datos con otros módulos pero esta funcionalidad no fue implementada por no ajustarse al cronograma de trabajo. El proyecto del responsable de integración Ing. Gustavo De Martino [10] en el cual se desarrolló un kernel para usar en los módulos de software del AntelSat tuvo lugar a posteriori por lo que el software de determinación de actitud desarrollado no está integrado con dicho kernel.

La implementación realizada ejecuta el algoritmo de determinación de actitud incluso cuando el satélite se encuentra en estado Spin. Esto puede no ser adecuado ya que el algoritmo de estabilización de actitud que se planea usar, solo requiere como dato el campo magnético, no la actitud (ver sección "Control de actitud").

El programa determina el estado actual (ver sección anterior) en base a los sensores. Cuando el estado es Estable y es de día, se ejecuta la rutina de determinación de actitud normal. La Figura 4.8 muestra al flujo del algoritmo de determinación.

El programa desarrollado incluye rutinas para medir el campo magnético en el sistema de referencia del satélite a partir del magnetómetro y para medir el vector de sol en el mismo sistema en base a los valores de los fotodiodos.

Fue desarrollada también una rutina para calcular la velocidad angular en estado Spin. La misma utiliza la actitud determinada en dos instantes de tiempo para aproximar la velocidad angular en base a las derivadas de los ángulos Roll, Pitch y Yaw.

Se realizaron estudios del tiempo de ejecución del programa y la cantidad de ciclos de reloj empleada por cada componente. La frecuencia de reloj usada es de 1 MHz. El subprograma que requiere más tiempo es el cálculo del campo magnético usando IGRF, abarcando el 71.9 % del tiempo seguido por el SGP4 que corresponde al 15.15 %. El uso de valores de punto flotante de doble precisión en 64 bits tiene un impacto importante en el tiempo de ejecución. Cuando el satélite se encuentra en estado Estable el programa entero requiere 9.28 segundos para ejecutar. Cuando el estado es Spin el tiempo es de 18.13 segundos.

4.4.12. Optimizaciones

La primer versión del programa usaba doble precisión en todas las variables y el compilador del IAR estaba configurado para tratar estos datos como de 32 bits. Aunque esto lograba buenos tiempos de ejecución el resultado del programa era incorrecto. Se decidió configurar el compilador para que tratara los elementos de doble precisión como de 64 bits. El resultado obtenido mejoró a costa de aumentar considerablemente el tiempo de ejecución hasta 3 minutos en el modo de simulación. Se cambiaron entonces algunas de las variables de doble precisión a precisión simple (`float`). Éste cambio mejoró el tiempo de ejecución, aunque no lo suficiente. Se intentó cambiar la biblioteca que implementaba las funciones trigonométricas pero en lugar de obtener mejoras se empeoró el rendimiento. Se realizaron varias optimizaciones en el programa para mejorar el rendimiento. La versión final del programa ejecutando en el microcontrolador tiene, como se mencionó anteriormente, el tiempo de ejecución de 9.28 segundos.

4.4.13. Ensayos

Se realizaron experimentos para evaluar el funcionamiento de la rutina de medición del vector de Sol. Las mismas tuvieron lugar en el laboratorio de fotometría de la Facultad de Ingeniería. Se orientó una lámpara hacia un modelo del satélite con los fotodiodos colocados en sus caras y se registró el error de medición, entendido como el ángulo formado entre el vector real y el medido. En estas pruebas el error máximo obtenido fue de 5.5°. Éste tiende a hacerse máximo cuando alguna de las caras del satélite se encuentra paralela a la dirección de la luz.

También se experimentó con el magnetómetro para evaluar el error de medición, colocando un modelo del satélite dentro de un cilindro metálico cuyo material actúa como aislante del campo magnético de la Tierra. En estas condiciones, el campo magnético medido dentro del cilindro idealmente es 0. Sin embargo se encontraron valores medidos distintos de 0 pero casi constantes, evidenciando un corrimiento fijo del magnetómetro que puede ser restado en software para mejorar la precisión de la medida. Este valor de error es originado tanto por el magnetómetro como por el kit de desarrollo. Es importante tener en cuenta que el valor del corrimiento es característico del prototipo del satélite usado y por lo tanto la prueba mencionada debe ser realizada para cada prototipo. Se concluye que el magnetómetro requiere un trabajo de calibración en cada instalación.

Una segunda prueba, buscó evaluar las perturbaciones ocasionadas por la circuitería del satélite en la medida de campo magnético. Se observó que el valor medido por el magnetómetro no varió. Sin embargo la ausencia de luz dentro del cilindro hizo que los fotodiodos no generaran una corriente importante por lo que este resultado no es concluyente.

Por último se realizó una prueba para determinar el umbral del vector de sol a usar para decidir si se está en día o noche. Se colocó un modelo del satélite en una posición donde era iluminado por el Sol, pero donde el mismo estaba próximo a ocultarse tras una pared. Se registraron los valores de tensión

de los fotodiodos y el módulo del vector de sol a medida que el modelo pasaba de estar completamente iluminado hasta estar completamente en sombra. Como resultado se decidió usar 0.9 V como umbral.

4.5. Control de actitud

La tesis de maestría "Control de orientación para el nanosatélite AntelSat" del Ing. Matías Tassano - responsable de ADCS, aborda el problema de la estabilización del satélite, trabajo que se encuentra aún en desarrollo al momento de la redacción de este documento. Este proyecto comprende las siguientes tareas:

- Rediseño del hardware del ADCS.
- Implementación de algoritmos para el control de actitud.
- Fabricación y validación en Tierra.

En esta sección se describe el avance actual en este proyecto.

4.5.1. Rediseño de hardware

El circuito integrado del ADCS que forma parte del AntelSat ha sido diseñado y fabricado. La Figura 4.9 muestra el diagrama de bloques del hardware del ADCS, donde se puede apreciar la conexión del chip MSP430 con los demás componentes.

La determinación y el control de la actitud del satélite involucran modelos físicos que implican realizar una gran cantidad de operaciones de punto flotante. Dado que la arquitectura del MSP430 no soporta este tipo de operaciones por hardware, éstas deben ser implementadas en software. Las pruebas realizadas con MSP430 en el proyecto de determinación de actitud mostraron que dicho hardware no permite obtener el rendimiento requerido para poder realizar un control adecuado de la orientación del satélite.

Como solución a este problema se decidió incorporar al ADCS un microcontrolador adicional con mayor poder de procesamiento para usarlo como coprocesador. El chip en cuestión es un microcontrolador STM32F3 del fabricante STMicroelectronics que cuenta con un CPU de 32 bits Cortex M4F de arquitectura ARM y diversos periféricos. Este módulo cuenta con una unidad de punto flotante que permite optimizar este tipo de cálculos. De esta forma el módulo MSP430 se encargará del flujo principal del programa incluyendo la comunicación con el bus I^2C, delegando sobre el módulo ARM los cálculos matemáticos. La comunicación entre ambos microcontroladores se realizará usando un bus SPI. Las tres líneas que lo forman (CLK, MISO y MOSI) pueden verse en la Figura 4.9.

La Figura 4.9 también muestra la conexión del módulo MSP430 del ADCS con los demás módulos del satélite, MCS, COM1, COM2, EMS y Payload.

La interfaz de depuración Spy-By-Wire permite seguir el flujo del programa ejecutando en el MSP430 desde un PC.

La Figura 4.9 permite observar las conexiones con el magnetómetro, el giróscopo y el sensor de sol.

4.5.2. Algoritmos de control

El problema del control de actitud se puede descomponer en dos etapas:

- Estabilización: frenar la rotación del satélite para evitar que gire fuera de control (*detumbling*).
- Alineación: aplicar las correcciones necesarias para mantener los instrumentos del satélite (por ej. antenas y cámaras) apuntando hacia la Tierra. Observar que aquí es donde la determinación de la actitud se hace relevante.

Fueron evaluadas diversas alternativas de algoritmos para realizar el detumbling y su uso en proyectos CubeSat desarrollados por otras universidades. Estos incluyen el Swisscube de la Ecole Polytechnique Fédérale de Lausanne en Suiza y el AAUSAT de la Aalborg University de Dinamarca ([11], [1]).

En estos trabajos el algoritmo B-dot, método basado en minimizar la derivada del campo magnético medido por los magnetómetros, es de los más usados para realizar el detumbling. Existen numerosas publicaciones describiendo la robustez y eficiencia de dicho algoritmo, respaldando la decisión de utilizar B-dot para la estabilización del AntelSat ([34]).

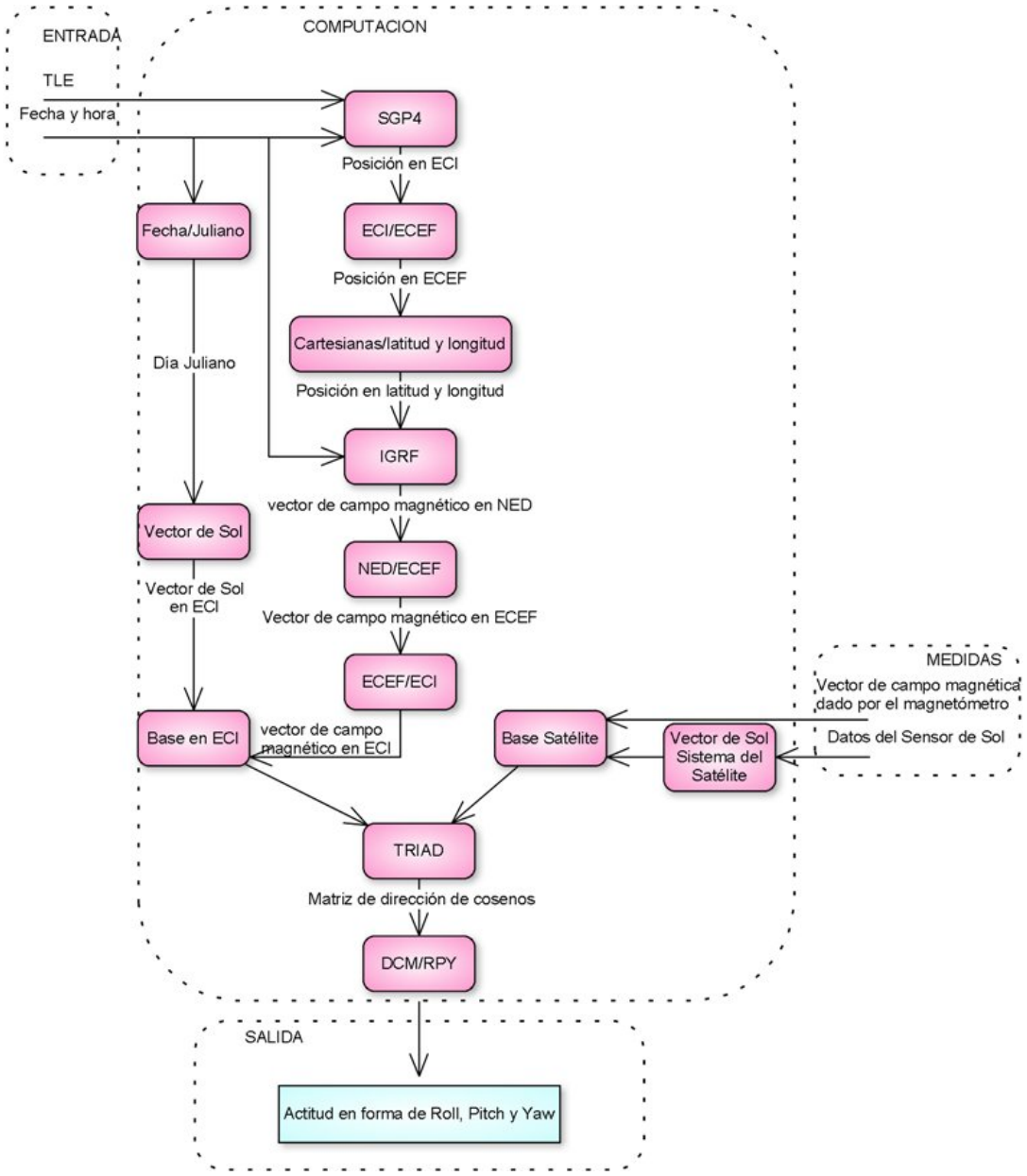

Figura 4.8. Diagrama de flujo del algoritmo de determinación de actitud. Autores de imagen: Victoria Alonsoperez, Adriana Castro, Santiago Zito

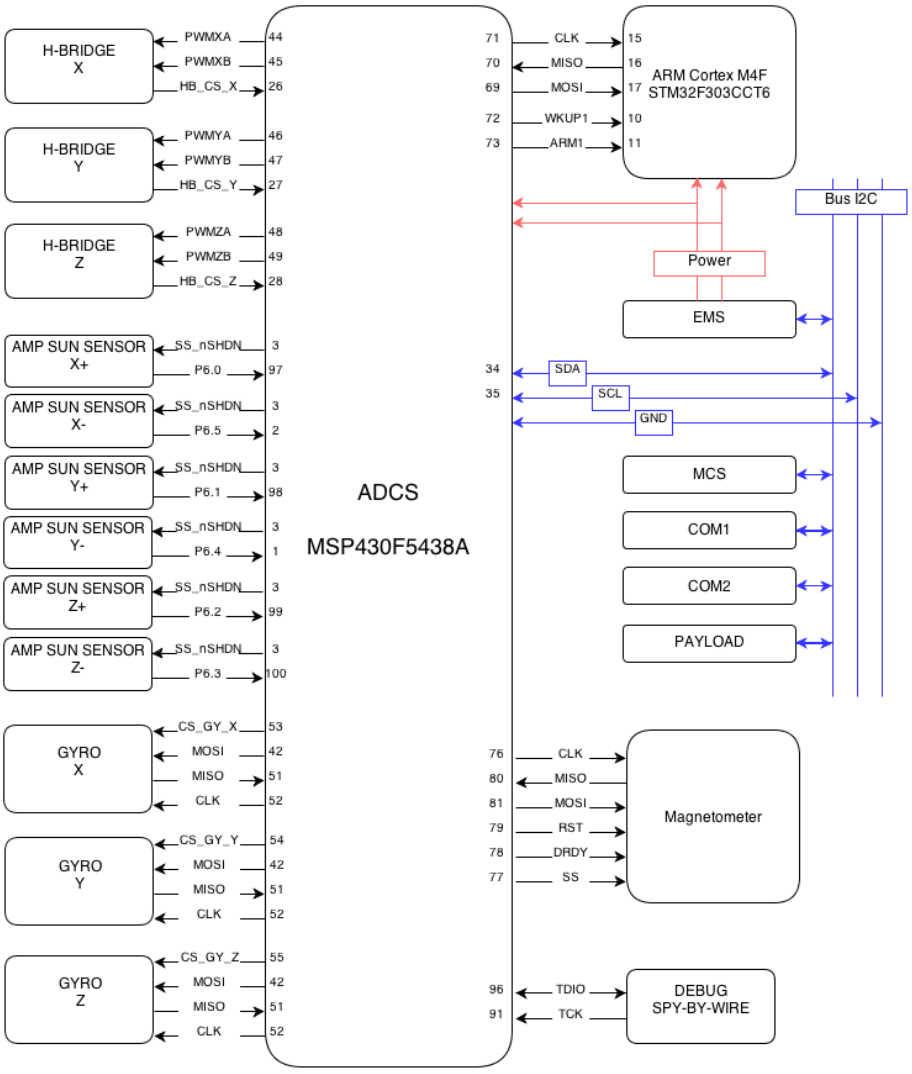

Figura 4.9. Diagrama de bloques ADCS. Autor de la imagen: Matías Tassano

4.6. Gestión de Energía

Esta problemática fue analizada en el proyecto "Desarrollo del sistema de gestión de energía para un satélite"[2] concluído en Febrero de 2011. El objetivo de este proyecto fue analizar, desarrollar y poner en funcionamiento el sistema de gestión de energía para un satélite, junto con su modelado térmico. A continuación se presentan los aspectos más relevantes desde el punto de vista del desarrollo del software de dicho trabajo.

4.6.1. La Importancia del Sistema de Gestión de Energía

La principal función del sistema de gestión de energía es capturar la energía solar a través de celdas fotovoltaicas, almacenarla en baterías y transmitirla luego a los diversos subsistemas. Además debe tener la capacidad de controlar las fallas referentes al consumo de energía de los demás módulos, ya sea por sobrecargas, sobretensiones o fenómenos generados por eventos espurios. El satélite no recibirá todo el tiempo los rayos del sol, por lo que en los tiempos de exposición se vuelve necesario almacenar la energía eficientemente. Se vuelve importante entonces maximizar la cantidad de energía que es extraida de los paneles en los periodos de sol. Durante los preriodos de sombra, la energía ha de ser tomada de las baterías y debe ser suficiente para alimentar al satélite hasta el próximo periodo de sol.

Otro aspecto importante es que la radiación puede causar errores en dispositivos lógicos. Esto ocurre cuando una partícula de alta energía golpea al dispositivo y pueden acontecer dos situaciones: el primer escenario es cuando la partícula atraviesa el material del dispositivo ionizando la trayectoria en su camino. Esta ruta actuará como un corto circuito entre las partes individuales/capas del dispositivo; mientras que el segundo escenario se presenta cuando la partícula de alta energía golpea un átomo con la energía suficiente como para dividirlo, provocando que cada una de las partes trace un camino de partículas ionizadas a través del material del dispositivo. Este fenómeno se conoce como un Single Event Upset (SEU), y puede cambiar los datos almacenados digitalmente, así como también cambiar el valor lógico de una compuerta en el momento equivocado.

El impacto en el dispositivo de carácter más grave, en el que la partícula de alta energía puede causar daños en el dispositivo, se conoce como un Single Event Latch-up (SEL). Estos efectos también pueden afectar a otros dispositivos, por ejemplo, causando un conflicto de bus. El software puede ser programado para detectar y corregir errores digitales, pero los conflictos en el bus y los SEL deben ser detectados y corregidos por hardware.

La función secundaria de este sistema de gestión energética es por tanto proteger los diferentes subsistemas contra estos fenómenos.

4.6.2. Requisitos de diseño

Existen varios requisitos en cuanto al diseño que el EMS debe cumplir.

- Requisitos Estructurales: la masa debe ser menor a los 90 g, con unas dimensiones de circuito impreso de 92 x 98 mm y dimensiones de baterías de 65 x 37 x 10 mm.
- Requisitos Eléctricos: el dispositivo debe contar con paneles solares, baterías recargables y componentes electrónicos que deben funcionar en ciertos rangos de tensión y corriente, previniendo los drenajes de energía mediante el uso de protecciones. Además, ningún dispositivo electrónico debe estar activo durante el lanzamiento a fin de evitar interferencias eléctricas o de RF.
- Requisitos Térmicos: la electrónica para su correcto funcionamiento debe estar dentro de un rango de operación entre 40°C y 70°C. En el caso de las baterías, el rango varía según el material del que están compuestas con valores mínimos entre -30°C y 0°C y valores máximos de entre 45°C y 70°C.

Otras especificaciones que deben ser cumplidas son:

- Pin de "Remove Before Flight": un "remove before flight (RBF) pin" es necesario para desactivar el CubeSat durante su ensamblaje fuera del P-POD. Este pin es quitado una vez que el CubeSat se coloque dentro. El pin se conecta directamente al sistema de gestión energética.

- Despliegue de Antenas y Retardo en la Transmisión de Datos: para permitir la adecuada separación de los CubeSats, sus antenas son desplegadas unos 15 minutos después de su expulsión del P-POD. Los CubeSats pueden entrar en un modo de transmisión de baja potencia (LPTM-Low Power Transmit Mode) 15 minutos después de su expulsión del P-POD. Se vuelve posible que los CubeSats activen todos los transmisores primarios, o entren en un modo de transmisión de alta potencia (HPTM-High Power Transmit Mode) luego de 30 minutos tras su expulsión del P-POD.

4.6.3. Descripción y Comportamiento del Sistema

Los módulos a los que se les debe proveer energía en esta versión del satélite serían:

- Control Principal (MCS)
- Sistema de Gestión de Energía (EMS)
- Comunicación con base en la Tierra (COMM)
- Control de Posición Activo (ADCS)
- Carga científica (PAYLOAD)

Desde el punto de vista de control de posibles fallas, se consideran tres tipos:

- Sobrecarga
- Cortocircuito
- Latch UP

Cualquiera de estas tres fallas de funcionamiento puede detectarse a partir de una sobrecorriente sobre una resistencia en serie colocada en cada uno de los puertos de alimentación. Al detectarse una sobrecorriente, se actua por hardware cortando el suministro de energía al periférico que presente la falla para luego restaurarle la energía. En el caso que al restaurar la energía se vuelva a detectar una gran corriente por alguno de los puertos de alimentación, el sistema cortará de forma permanente el suministro a ese módulo.

Si bien las acciones de corte de energía son realizadas por hardware, las decisiones de si este corte es permanente, así como la realización de un corte generalizado a toda la nave, es una decisión procesada por software.

4.6.4. Protocolo de Protección de Fallas por Sobrecorrientes

Se requiere la protección de todos los sistemas electrónicos contra sobrecorrientes ocasionadas, principalmente, a consecuencia de single-event up sets (SEU) y/o latch-ups. Al generarse una falla, el microprocesador debe esperar un cierto tiempo para volver a encender el puerto que presentó la falla. Durante este intervalo de espera, se debe informar de la falla al módulo de control principal.

Si al reiniciar el puerto la falla persiste, se fija otro intervalo de espera y se reconecta el puerto. De mantenerse la condición al segundo intento de conexión, se supone que el problema es causado no por un factor externo, sino por un error interno al módulo en cuestión. Por lo tanto se avisa al módulo principal sobre la naturaleza de la falla y no se realizan nuevos intentos de conexión.

Si se recibe un pedido de reconexión por parte del módulo principal éste es realizado, siguiendo el mismo procedimiento descrito anteriormente, con la excepción de que en esta oportunidad solo hay un intento de reconexión. El diagrama de flujo del protocolo de manejo de fallas para un solo puerto se describe en la Figura 4.10.

4.6.5. Adquisición de Datos de Mantenimiento

Para lograr un manejo eficiente y seguro de los distintos dispositivos presentes en el satélite es necesario recabar información sobre distintas magnitudes físicas.

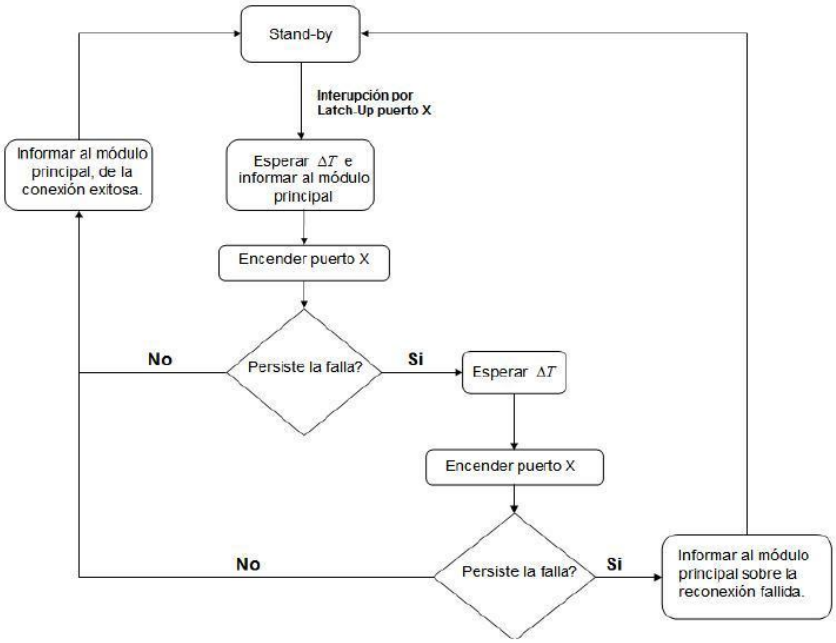

Figura 4.10. Diagrama de flujo para la interrupción por latch-up para un puerto.

También deben recabarse datos referentes al estado de funcionamiento de las baterías, siendo preciso que su operación este restringida a un determinado rango de temperaturas para evitar daños severos.

Por otro lado la corriente y tensión en los paneles resulta útil para calcular la potencia de entrada al sistema.

En resumen, las magnitudes a ser relevadas son: tensiones, corrientes y temperaturas.

4.6.6. Funcionalidad del EMS

Funciones principales:

- Transferir la energía de los paneles a las baterías y los demás subsistemas.
- Optimizar la potencia extraida de los paneles (MPPT)
- Asegurar protección contra "latch-up" a los demás subsistemas.
- Protección de las baterías contra subtensiones y sobretensiones, así como su operación en los rangos de temperatura adecuados.

Funciones secundarias:

- Actuar como "watchdog" del MCS.
- Recabar información de mantenimiento.
- Comunicación con el MCS a través de I^2C.
- Función de baliza.

El software esta dividido en funciones que resuelven el listado anterior. Sin embargo la estructura de ejecución está dividida en dos partes, un bucle infinito y una serie de rutinas de atención a interrupciones (ISR).

El bucle principal consta del recabado de datos de mantenimiento, la protección de la batería y parte de la función de comunicación i^2c. Las rutinas de atención a interrupciones comprenden el contador del

"watchdog" MCS, el manejo de la carga de los paneles solares (MPPT), la baliza y la protección contra "latch-up".

Bucle Principal

El bucle principal consta de la secuencia de arranque y un bucle infinito, ver Figura 4.11.

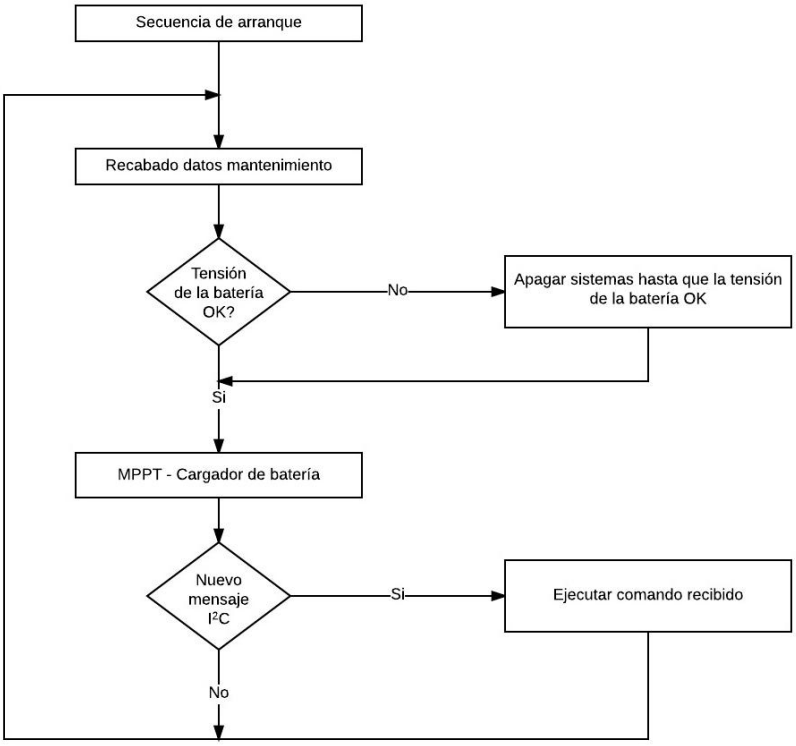

Figura 4.11. Diagrama de flujo del bucle principal.

Secuencia de Arranque La secuencia de arranque del satélite es responsabilidad natural del EMS por ser el primero de los subsistemas en activarse.

1. Abandono del P-POD
   - Señal física de liberación.
   - 0.3s de espera en el sistema.
   - Arranca el EMS.
2. Espera de 5 minutos
   - El EMS se toma 5 minutos para permitir que los satélites se separen al salir del P-POD.
3. Liberación de antena.
   - El EMS da la orden de liberar las antenas.
4. Arranque del sistema de comunicación (COMM) y el MCS.
   - El EMS enciende el COMM y pone en funcionamiento la baliza.
   - El EMS enciende el MCS con el puerto de arranque en la PROM.

5. Secuencia de arranque del MCS.
    - Enciende el MCS.
6. Inicio del "watchdog".
    - El EMS inicia el contador del "watchdog" para el MCS.
7. Apagado de la baliza.
    - El EMS continúa con la baliza encendido hasta que el MCS lo detiene.

Adquisición de Datos de Mantenimiento La adquisición de datos se realiza durante el bucle principal.

Protección de Baterías La protección de la batería abarca cuatro aspectos: sobretensión, subtensión, sobrecorriente y rango de temperatura. La protección del rango de temperatura asegura que la batería no sea cargada cuando la temperatura esté por fuera del rango 0°C a 50°C, y en descarga entre -20°C a 50°C.

Función i²c La implementación de la comunicación I^2C está dividida en dos funcionalidades: interrupción y bucle principal. La primera se encarga de las operaciones de comunicación de bajo nivel (chequeo de CRC por ejemplo), mientras que la segunda es responsable de la interpretación de los datos recibidos.

Selección de arranque Esta función será implementada para que sea posible modificar el software del satélite durante la propia misión. Esto permite aumentar la vida útil del satélite ya que en la eventualidad de haber completado la misión original, se puede modificar sus instrucciones para que realice nuevas tareas. La selección de arranque se lleva a cabo por orden del MCS. Con dicha orden se seleccionará la memoria de arranque que podrá ser la PROM cargada con el código original o una memoria FLASH con el código de arranque enviado al satélite desde la Tierra.

Interrupciones

Hay tres tipos de interrupciones en el programa. Una de ellas es periódica y las otras son aperiódicas. Las aperiódicas son: la protección contra "latch-up" y la de I^2C. La periódica es una interrupción que corre a una frecuencia de 76Hz y es usada el "watchdog" del MCS.

Baliza La baliza funciona cuando se inicia el arranque del satélite, es decir una vez liberado del P-POD transmitirá información de identificación del satélite en código morse almacenado en una memoria PROM. Luego de que el arranque del satélite es exitoso, el MCS puede ordenar el apagado de la baliza.

Contador Externo del "Watchdog" La función del "watchdog" externo es ser un respaldo del "watchdog" interno. Si no hay comunicación por 10s entonces el MCS es apagado y no se enciende de nuevo hasta después de 5 minutos. La función usa un contador creciente de 10s cuando el MCS está encendido, y un contador decreciente de 5 minutos cuando el MCS está apagado. Cuando han pasado 5 minutos el MCS es encendido en el mismo modo de arranque que estaba antes de la cuenta (PROM o FLASH). Sin embargo si esto sucede dos veces el modo de arranque es cambiado al original (PROM). Esto significa que si el MCS nunca se comunica con el EMS será arrancado en la PROM. El COMM nunca es apagado mientras el EMS tenga encendida la baliza.

MPPT La relación tensión-potencia de las celdas solares, las fluctuaciones de su eficiencia con la temperatura y el nivel variable en la iluminación de los paneles, requieren necesariamente del uso de un algoritmo de MPPT que permita extraer la máxima potencia disponible en los paneles. Para ello se diseñó el módulo de MPPT-Cargador de Baterías, que implementa un convertidor DC-DC con una interfaz de control, para que el microprocesador pueda realizar la función de MPPT. La utilización del

microprocesador permite la utilización de algoritmos digitales de MPPT, que presentan varias ventajas sobre sus contrapartes analógicas. Estos presentan una mayor flexibilidad, permitiendo su modificación y el agregado de mejoras en forma sencilla. También conllevan un costo de volumen y masa inferiores, ya que el hardware que se necesita es el microprocesador y un filtro RC. La idea es que un algoritmo de MPPT digital puede alcanzar niveles de sofisticación mayores a los analógicos.

4.7. Entorno de software

Nuestro proyecto hace uso de algunos módulos de software desarrollados en proyectos anteriores. En esta sección se describen los componentes de software que existían previamente y las herramientas de desarrollo usadas.

El entorno de desarrollo usado en el proyecto AntelSat para trabajar con la línea de microcontroladores MSP430 es **IAR Embedded Workbench**[18]. Esta herramienta ha resultado la más adecuada según la experiencia de proyectos anteriores [2]. El entorno permite manejar proyectos, provee un editor de texto y un compilador de C/C++ para MSP430. Se integra con el dispositivo programador usado para instalar el programa desarrollado en el microcontrolador. Incluye también el depurador C-SPY, que permite depurar la aplicación mientras ejecuta en el hardware objetivo, con funcionalidades como inspección de memoria y breakpoints.

Inter-Module Messaging Protocol o IMMP es el protocolo de comunicaciones desarrollado en el proyecto del Ing. Gustavo De Martino[9] para la comunicación I^2C entre los módulos. El software para los módulos MSP430 se desarrolló utilizando este componente.

El microkernel desarrollado en el proyecto del Ing. Gustavo De Martino [10] fue el sistema operativo de tiempo real a utlizar en el desarrollo del software para los módulos MSP430. Los términos kernel o microkernel en el resto de este documento se refieren al mismo a menos que se indique lo contrario.

El kernel permite aumentar el nivel de abstracción y modularización del software por medio del concepto de *aplicación*. Esta noción es similar en este contexto al concepto usual de hilo, es decir un flujo de ejecución independiente dentro del programa. El kernel implementa esta abstracción alternando el CPU entre una aplicación y otra, y manteniendo en forma consistente el contexto de ejecución de cada una.

También se provee comunicación entre aplicaciones a través del concepto de *recurso*. Un recurso es un canal de comunicación unidireccional, por medio del cual las aplicaciones pueden enviar y recibir datos a través de las primitivas `read` y `write`. El recurso también puede ser usado como primitiva de sincronización, con operaciones de señalización y espera. El kernel es *no expropiativo*, esto significa que el pasaje del CPU de una aplicación a otra no ocurre nunca fuera del control de las mismas sino explícitamente, cuando la aplicación en ejecución realiza una operación sobre un recurso. Puede encontrarse más información sobre el diseño del kernel en su manual de usuario [10].

El software desarrollado en el proyecto de Determinación de Actitud fue reutilizado en parte. Este software fue revisado y adaptado para el uso en este proyecto.

El desarrollo para el microcontrolador ARM se realizó utilizando el **GNU ARM Embedded Toolchain**. Se trata de una implementación para ARM del compilador GCC y sus herramientas asociadas, publicada como código abierto con licencia GPL v2.

Open On-Chip Debugger[26] es la biblioteca usada para depurar e instalar software en el microcontrolador ARM. Junto con el depurador gdb proviso por el **GNU ARM Embedded Toolchain**[15], permite inspeccionar la memoria del programa mientras ejecuta en el microcontrolador, establecer breakpoints, etc. Es software libre publicado con licencia GPL.

4.8. Equipo ANTELSAT y roles

En el desarrollo del presente proyecto nuestro grupo ha trabajado en interacción con otros miembros del proyecto en distintos roles.

- Ing. Gustavo De Martino: responsable de integración (esta asignación es informal).

- Ing. Gonzalo Sotta, Ing. Ignacio de León, Ing. Gonzalo Gutierrez: equipo EMS.
- Ing. Matías Tassano: responsable ADCS.

Cabe notar que los roles anteriores son denominaciones informales basadas en el área de trabajo de cada miembro a grandes rasgos, no indicando una participación exclusiva de cada miembro en cada módulo. Por motivos de claridad en el resto de este documento se refiere a los miembros del proyecto AntelSat por los roles aquí descritos.

Módulo de Gestión de Energía

En esta sección se describe el trabajo realizado en el Módulo de Gestión de Energía. Se presenta una breve descripción de su funcionalidad, los requerimientos relevados, cómo fue su desarrollo y verificación. También se documentan los resultados obtenidos durante la integración del módulo con el resto del satélite. Finalmente, como lecciones aprendidas, se resume la experiencia ganada durante esta etapa del proceso.

5.1. Objetivos

El propósito del EMS es capturar energía a través de los paneles solares y administrarla a los demás módulos del satélite. Se trata de una función que es crítica para la correcta operación del satélite. Las funciones del EMS son:

- Capturar la energía solar a través de celdas fotovoltaicas.
- Proveer energía a los diversos subsistemas.
- Almacenar en baterías la energía remanente para alimentar al sistema en los periodos de baja o nula radiación solar.
- Encender y apagar los restantes módulos según sea necesario.
- Almacenar el estado de los restantes módulos (encendido, apagado, falla).
- Reportar a la Tierra mediante una baliza morse el estado de salud del satélite y otros datos básicos.

5.2. Requerimientos

En esta sección se presenta un resumen de los requerimientos relevados para el EMS. Se trata de una síntesis del contenido del documento **EMS Especificación de Requerimientos** elaborado al principio del proyecto y adjunto a este informe.

5.2.1. Arranque

El EMS recibe alimentación en el momento en que sale del P-POD. Debido a la proximidad del resto de los satélites que son lanzados en la misión, es necesario esperar 30 minutos antes de desplegar las antenas. La espera se realiza mediante un mecanismo por hardware. El microcontrolador puede obtener el estado de despliegue de antenas realizando la lectura de un puerto. Si las antenas no se despliegan, el módulo debe esperar 24 horas antes de poner en funcionamiento a la baliza y los restantes módulos. Si se alcanza este plazo límite, el módulo debe dar por desplegadas las antenas y reportar la situación.

5.2.2. Carga y mediciones de las baterías

Con el fin de maximizar la transferencia de potencia entre los paneles solares y el resto del sistema, EMS debe implementar la funcionalidad de *Maximum Power Point Tracking* (MPPT). Esta característica

consiste en controlar la tensión de trabajo de los paneles solares a fin de maximizar en todo momento la potencia obtenida de los mismos. Deben haber dos modos de control, modo hardware y modo software. En el primero la tensión de trabajo de los paneles se fija en un valor determinado por el hardware, en el segundo se debe implementar un bucle de control que varie la tensión de trabajo de forma de buscar la máxima potencia.

EMS debe monitorear periódicamente las protecciones de las baterías para determinar la situación energética del sistema. Se toma el nivel de tensión de salida de las baterías como una indicación proporcional a la cantidad de energía presente en ellas. Las protecciones deben ser reconfiguradas periódicamente para mitigar fallos debidos a los efectos de la radiación. También se debe reconfigurar las protecciones si se detecta que una de ellas tiene un comportamiento inadecuado. Se entiende por comportamiento inadecuado la obtención de un valor inconsistente en cualquiera de las magnitudes que se obtengan de las protecciones. El ADC usado para realizar las lecturas de tensión también debe ser reconfigurado si se observa una lectura inconsistente.

5.2.3. Estados del EMS

El ciclo de vida del EMS desde que el satélite se energiza debe pasar por tres estados: `DEPLOY_WAIT`, `RECOVERY` y `SAFE`. EMS debe mantenerse en estado `DEPLOY_WAIT` desde que se energiza hasta pasados 15 minutos luego del despliegue de antenas. En este estado EMS debe ser el único módulo encendido.

Luego del despliegue de antenas EMS debe pasar a modo `RECOVERY` (modo de bajo consumo). En estado `RECOVERY` solo la baliza de EMS debe estar encendida.

Cuando EMS detecta que se la acumulación de energía en baterías ha alcanzado un nivel suficiente debe pasar a modo `SAFE`. En este estado los siguientes módulos deben estar encendidos: Baliza asociada al EMS, bus I^2C, MCS, COMM1 y COMM2.

El encendido de módulos al pasar a modo `SAFE` debe realizarse en el siguiente orden: bus I^2C, MCS, COMM1, COMM2. Los módulos ADCS y PAYLOAD deben encenderse únicamente en modo `SAFE` y a demanda del Módulo de Control Principal. Los módulos BANDA_S1 y BANDA_S2 deben encenderse y apagarse a demanda del módulo PAYLOAD y mantenerse apagados cuando este esté apagado.

5.2.4. Transmisión y Recepción de mensajes I^2C

El EMS debe ser capaz de enviar y recibir mensajes a través del bus I^2C para comunicarse con el resto de los módulos.

5.2.5. Encendido y apagado de módulos

Si EMS tiene una falla eléctrica (sobre-consumo), el hardware lo apagará y volverá a encender luego de un tiempo prudencial. El resto de los módulos debe ser controlado (encendido y apagado) por el EMS.

EMS debe ser capaz de recibir de MCS el estado deseado de los módulos como un mensaje a través del bus I^2C.

EMS debe detectar si un módulo encendido ha generado un cortocircuito. Para esto debe verificar un puerto de entrada-salida luego de un intervalo de tiempo a partir del encendido, y periódicamente mientras el estado deseado indique que el módulo debe estar activo.

Mientras EMS esté en modo `SAFE`, debe asegurar que los módulos COMM1 y COMM2 no estén simultáneamente apagados, salvo que ambos estén en falla.

Cada vez que EMS detecte una falla debe reportarlo a MCS.

Además debe intentar encender un módulo que ha fallado una cantidad predefinida de veces, siempre que el estado deseado sea “encendido”. Una vez alcanzada esa cantidad de fallas en forma consecutiva, ésta se considera permanente.

Si el estado deseado de un módulo en el que se ha detectado una falla permanente es “encendido” y ha pasado cierto tiempo desde el último intento, EMS realizará un nuevo intento de encender el módulo (sujeto a sus precondiciones de encendido particulares).

Si el Bus I^2C está encendido y MCS, COMM1 o COMM2 están encendidos, EMS intentará enviar un mensaje. Si el mensaje no se puede enviar, EMS deberá reiniciar todo el satélite.

Salvo en caso de falla, antes de apagar al MCS, EMS debe enviarle a MCS un mensaje de notificación y esperar un tiempo de al menos 1 segundo.

Una vez que se alcanza el modo `SAFE` por primera vez, luego de haber encendido el MCS, el EMS debe solicitar al MCS sus parámetros iniciales.

EMS debe encender y apagar los transmisores BANDA_S1 y BANDA_S2 a solicitud del PAYLOAD. Cuando EMS recibe del PAYLOAD una solicitud para encender alguno de los transmisores y el encendido del mismo ocurre sin problemas, EMS debe responder a PAYLOAD con un mensaje de confirmación que indica que el transmisor está encendido.

Cuando EMS recibe de PAYLOAD una solicitud para encender alguno de los transmisores BANDA_S1 o BANDA_S2 y el mismo está o queda en estado de falla, EMS debe responder a PAYLOAD con un mensaje que indica que el transmisor no está encendido.

Si EMS recibe la solicitud de encender uno de los transmisores de banda S y el mismo ya está encendido, responderá con un mensaje de confirmación.

EMS apagará los transmisores BANDA_S1 y BANDA_S2 a solicitud de PAYLOAD o si ha transcurrido un tiempo de 3 segundos desde la última solicitud de encendido.

EMS debe ser capaz de reiniciar todo el sistema a solicitud de MCS.

5.2.6. Precondiciones para el encendido de módulos

Existen una serie de relaciones entre módulos que vinculan sus cambios de estado. En ciertas situaciones puede que el estado deseado de un módulo sea encendido y sin embargo el estado de los demás módulos haga que encenderlo carezca de sentido. Las siguientes son algunas de estas relaciones.

El Bus I^2C debe estar apagado en los modos `DEPLOY_WAIT` y `RECOVERY`. En modo `SAFE` debe estar encendido salvo que estén en falla simultáneamente MCS, COMM1 y COMM2.

MCS debe estar apagado en los modos `DEPLOY_WAIT` y `RECOVERY`. En modo `SAFE` debe estar encendido salvo que el Bus I^2C esté en falla.

COMM1 debe estar apagado en los modos `DEPLOY_WAIT` y `RECOVERY`. En modo `SAFE` debe estar encendido salvo que el Bus I^2C esté en falla o Beacon y MCS estén ambos en falla.

5.2.7. Envío de datos a través de Baliza

La baliza es el elemento de comunicación básico del que dispone el satélite. Una vez que se despliegan las antenas la baliza se enciende y permanece encendida hasta la próxima vez que el satélite se reinicie, incluso cuando los módulos COMM estén apagados. La información enviada por medio de la baliza contiene datos básicos acerca del estado de salud del satélite.

MCS enviará periódicamente a EMS un mensaje con tres identificadores numéricos que EMS deberá interpretar como el identificador del último mensaje correctamente procesado por MCS, la indicación de si el servicio de digipeater se encuentra activado y la indicación de si el servicio de SSTV se encuentra activado respectivamente.

Cada uno de los módulos, COMM1 y COMM2, enviarán de forma periódica al EMS un mensaje compuesto por tres identificadores numéricos. EMS deberá registrar dichos valores como rangos de RSSI, Temperatura 1 y Temperatura 2.

El ADCS enviará periódicamente a EMS un mensaje con dos identificadores numéricos que deberán interpretarse como estado de actitud del satélite. El primer identificador indica el tipo de dato a ser

transmitido mientras que el segundo el rango en el que se encuentra. EMS verificará que los valores recibidos se encuentren dentro del rango de 0 a 10 e interpretará los identificadores inválidos por el carácter de borrado.

EMS debe construir y transmitir a la Tierra estos datos mediante el uso de la baliza morse.

En modo `RECOVERY` la baliza debe transmitirse cada 5 minutos, mientras que en modo SAFE, debe transmitirse cada 20 segundos. En modo `RECOVERY`, la baliza contendrá los siguientes datos:

- Identificador del satelite
- Voltaje pack baterías 1
- Tension batería 1
- Tension batería 2
- Tension batería 3
- Tension batería 4
- Vector de Estado de los módulos

En modo SAFE, a los datos anteriores se agregarán los siguientes:

- RSSI de COMM1
- Temperatura 1 de COMM1
- Temperatura 2 de COMM1
- RSSI de COMM2
- Temperatura 1 de COMM2
- Temperatura 2 de COMM2
- Último mensaje de MCS
- Estado de actitud

EMS debe ser capaz de recibir de MCS la solicitud de agregar a la baliza un mensaje de usuario, cadena de texto restringida a lenguaje morse internacional, junto con un valor que indicará la cantidad de emisiones. En ese caso, el mensaje se agregará a la baliza en modo SAFE y se mantendrá hasta que cambie el modo a RECOVERY, se alcance la cantidad de emisiones solicitada o se solicite otro mensaje de usuario. Si el mensaje tiene tamaño cero, este no se emitirá (esto último provee un mecanismo para cancelar el mensaje de usuario antes que se complete la cantidad prevista de repeticiones).

Para ahorrar energía, la baliza solo debe estar encendida mientras dura el valor alto de la señal morse. Cuando comienza un pulso se debe encender la baliza, mantener el valor alto el tiempo que corresponda y apagar la baliza.

Cada vez que MCS recibe un mensaje desde la Tierra, éste envía a EMS una notificación. EMS debe recibir dicha notificación y enviar a través de la baliza el carácter morse R (*Roger*) en respuesta. Luego de enviar una R, no se debe enviar otra hasta haber transcurrido un tiempo T a definir. Si durante el tiempo de espera T llegan notificaciones, al cumplirse el tiempo se debe enviar una única R, sin importar que haya llegado más de una notificación en ese periodo.

La transmisión del mensaje de usuario debe estar separada de la baliza por un espacio (separación de palabras en morse). Las palabras del mensaje también deben ser separadas por espacios.

5.2.8. Reporte de telemetría

Mientras el MCS esté encendido, EMS debe enviar periódicamente un reporte de estado para que sea incluido en la baliza telemetría. El conjunto de elementos a ser incluidos en el reporte puede consultarse en la sección “Apéndice D: Tablas de Telemetría” del documento adjunto “Apéndices”.

5.2.9. Reloj

EMS debe mantener la hora de respaldo del sistema. Cuando EMS inicia debe establecer la hora por defecto, y actualizarla cada vez que MCS lo indique mediante un mensaje. EMS debe enviar a MCS la hora de respaldo cada vez que éste lo solicite mediante un mensaje.

5.2.10. Eventos

EMS debe reportar un evento a MCS cuando ocurre un timeout en la espera por el despliegue de las antenas. También debe reportar como evento el cortocircuito o la falla de un módulo.

5.2.11. Comandos de entrada

Los siguientes son los mensajes I^2C que EMS debe aceptar provenientes de otros módulos.

- Actualización de hora desde MCS.
- Solicitud de hora desde MCS.
- Reporte de datos de baliza de COMM1 desde MCS.
- Reporte de datos de baliza de COMM2 desde MCS.
- Reporte de MCS para baliza desde MCS.
- Reporte de ADCS para baliza desde MCS.
- Reiniciar todo el sistema desde MCS.
- Encender BANDA_S1 desde PAYLOAD.
- Encender BANDA_S2 desde PAYLOAD.

5.2.12. Parámetros

El conjunto de parámetros de EMS debe incluir lo siguiente:

- Umbral para pasaje de modo `RECOVERY` a modo `SAFE`.
- Umbral para pasaje de modo `SAFE` a modo `RECOVERY`.
- Paso para el algoritmo de MPPT.

5.3. Diseño

Dado que el desarrollo del software de EMS era un problema complejo que debía resolverse en un tiempo reducido se decidió junto al equipo de AntelSat realizar una división del trabajo entre los diferentes involucrados para aprovechar en forma óptima los recursos y conocimientos disponibles. Nuestro equipo realizó el análisis de requerimientos, diseño e implementación de un prototipo de software de alto nivel que resuelve la lógica del problema. Este prototipo deja de lado los detalles de interacción con el hardware, abstrayéndolos en funciones stub. Paralelamente, el responsable de integración y el equipo de EMS trabajaron en el desarrollo y verificación del software de bajo nivel asociado al manejo de los periféricos.

El prototipo del software del módulo de gestión de energía fue diseñado teniendo en cuenta dos criterios básicos:

1. Mantener el código de cada aplicación lo más simple y acotado posible.
2. Evitar la dependencia entre aplicaciones.

El primer criterio busca simplificar las pruebas unitarias. Cuanto menos responsabilidades se le atribuyan a una aplicación, más sencillo es el grafo de decisión y por lo tanto menos casos deben ser contemplados en las pruebas. Sin embargo, esto conlleva a aumentar número de aplicaciones que requieren comunicarse entre sí. El segundo criterio busca entonces disminuir el acoplamiento entre las aplicaciones.

La solución diseñada utiliza un sistema de comunicación entre aplicaciones basado en los accesos a memoria global del proceso. El microkernel provee de mutua exclusión entre aplicaciones, evitando problemas de concurrencia en el acceso a memoria compartida.

Todo dato que necesite ser transmitido entre dos o más aplicaciones, es almacenado en una variable global definida en un módulo independiente. El acceso a una variable global requiere la inclusión de éste módulo y no de los módulos en los que se definen las aplicaciones. La separación en módulos de las variables globales facilita su manipulación, brindando una interfaz con métodos de acceso y escritura. Internamente las variables se encuentran triplicadas, a fin de proveer redundancia ante mutaciones de bits. Para las aplicaciones sin embargo esto es transparente.

Las aplicaciones escriben y leen en las variables globales, en adelante denominadas variables de estado, sin conocer qué otras aplicaciones acceden a ellas. En los casos que se requiere sincronización entre aplicaciones para acceder a cierto dato, se hace uso de los recursos del kernel y las primitivas de sincronización que este provee.

A continuación se enumeran las variables de estado utilizadas:

- `tension_threshold_ascend`: umbral de tensión de baterías usado para la transición de modo `RECOVERY` a SAFE. Recibido como parámetro.
- `tension_threshold_descend`: umbral de tensión de baterías usado para la transición de modo `SAFE` a `RECOVERY`. Recibido como parámetro.
- `mppt_step`: valor numérico que indica el valor absoluto del paso a usar en el algoritmo de Hill-climbing. Recibido como parámetro.
- `retries`: vector que contiene la cantidad de intentos de encendido pendientes de cada módulo antes de considerar que está en falla. Tiene una entrada por cada módulo. El vector se inicializa con el valor 5 para todos los elementos. Este se decrementa cada vez que un intento de encender el módulo falle y se vuelve a 5 cuando el mismo inicia correctamente.
- `modules`: arreglo que guarda la correspondencia de un índice de 0 a 7 a un identificador de módulo. Sus entradas son los identificadores de los módulos MCS, COMM1, COMM2, ADCS, PAYLOAD, I^2C, SBAND1 y SBAND2. Los vectores `effective_states` y `desired_states` usan la misma correspondencia entre módulos e índices.
- `effective_states`: vector que almacena el estado efectivo de los módulos. Tiene una entrada por cada uno. Los valores posibles de las entradas son: ON, OFF, FAIL. Este último valor indica un estado de falla considerado permanente.

  Nota: Cuando un módulo está en falla su entrada en el vector de reintentos es 0, sin embargo se decide guardar que el estado es falla en el vector de estados también. Esto se debe a que el

estado puede ser falla aún cuando la entrada correspondiente del vector de reintentos es distinta de 0. Esta situación ocurre cuando el contador de reintentos ha sido incrementado por la aplicación restauradora de módulos pero el intento de encender el módulo todavía no se ha realizado.

- `desired_states`: vector que almacena el estado deseado de los módulos. Tiene una entrada por cada uno. Se inicializa de la siguiente manera: BEACON = ON, I^2C = ON, MCS = ON, COMM1 = ON, COMM2 = ON, ADCS = OFF, PAYLOAD = OFF, SBAND1 = OFF, SBAND2 = OFF.
- `module_timeouts`: vector que almacena valores de timeout (expresados en minutos) para los reintentos de encendido de los módulos fallidos. Tiene una entrada por cada uno. Cuando un módulo falla este contador se inicializa y comienza a decrementarse. Cuando llega a 0 se hace un nuevo intento de encender el módulo.
- `tension_batteries`: valor de tensión aplicado a las baterías.
- `energy_mode`: indica el estado del sistema (`SAFE`, `RECOVERY` o `DEPLOY_WAIT`).
- `beacon_data`: estructura de datos que contiene los datos que se envían en la baliza (tanto la básica como la completa). Su tipo de datos es `baliza_completa`, cuya definición se encuentra más adelante en este documento.
- `user_message`: último mensaje de usuario recibido.
- `beacon_enable`: indica si el envío de mensajes por beacon está habilitado.
- `timer_callbacks`: arreglo de punteros a funciones usado por la aplicación `timer_loop`.
- `timer_counters`: arreglo de contadores usado por la aplicación `timer_loop`.
- `user_message`: estructura de datos donde se guarda el mensaje de usuario.
- `user_message_counter`: contador que almacena la cantidad de repeticiones restantes del mensaje de usuario.
- `morse_unit`: variable que guarda la duración (en milisegundos) de un punto en el código morse de la baliza.

Las aplicaciones se acoplan a las distintas variables de estado para generar canales de transferencia de datos asincrónicos. Los recursos del kernel son utilizados para realizar comunicaciones sincrónicas, básicamente señalizaciones que indican que los datos ya se encuentran listos en las variables de estado.

A continuación se describen los recursos utilizados por las aplicaciones como medio de sincronización. Puede encontrarse más información sobre el concepto de recurso en la documentación del sistema operativo ([10]).

- `REC_AWAKE_MANAGER`: usado para despertar la aplicación `on_off_manager`.
- `REC_RESPONSE_MANAGER`: usado para recibir la respuesta de la aplicación `on_off_manager`.
- `REC_MORSE_COMMUNICATOR`: usado para despertar la aplicación `morse_communicator`.
- `REC_EMS_APPLICATION_PAYLOAD_ON_OFF_COMMAND_PROCESSOR`: usado por la aplicación `payload_on_off_command_processor` para recibir mensajes por I^2C.
- `REC_EMS_APPLICATION_BEACON_DATA_COLLECTOR`: usado por la aplicación `beacon_data_collector` para recibir mensajes por I^2C.
- `REC_EMS_APPLICATION_PARAMS_LOADER`: usado por la aplicación `params_loader` para recibir mensajes por I^2C.
- `REC_EMS_APPLICATION_CLOCK`: usado por la aplicación `clock` para recibir mensajes por I^2C.
- `REC_EMS_APPLICATION_MCS_ON_OFF_COMMAND_PROCESSOR`: usado por la aplicación `mcs_on_off_command_processor` para recibir mensajes por I^2C.
- `REC_EMS_APPLICATION_ROGER_RECEIVER`: usado por la aplicación `roger_receiver` para recibir mensajes por I^2C.
- `REC_BEACON_SERVICE`: usado para sincronización con el servicio que maneja el beacon (`beacon_service`).

- `RESOURCE_IMMP_REQUEST`: recurso utilizado para enviar mensajes a la aplicación `immp_tx`, encargada de enviar mensajes por I^2C hacia otros módulos del satélite. Este recurso es manejado internamente por el módulo `immp`.
- `RESOURCE_IMMP_REPLY`: usado para recibir la respuesta de una operación de envío I^2C de parte de la aplicación `immp_tx`. Este recurso es manejado internamente por el módulo `immp`.

5.3.1. Funcionalidades

Las distintas funcionalidades del módulo pueden agruparse en 4 grupos:

- gestión de encendido y apagado de módulos.
- optimización energética (mppt).
- comunicación con la Tierra (baliza morse).
- comunicación con otros módulos.

Para cada uno de estos grupos se define un conjunto de aplicaciones, variables de estado y recursos que implementan las funcionalidades correspondientes a cada uno. Las figuras 5.1, 5.2, 5.3, 5.4 y 5.5 muestran como se interconectan los distintos componentes. Los detalles de cada grupo se desarrollan en las siguientes subsecciones. Una explicación de cada aplicación, incluyendo pseudocodigos, puede ser encontrada en la sección 1.1: "Apéndice A: Aplicaciones de EMS" del documento adjunto "Apéndices".

Gestión de encendido y apagado de módulos

El satélite posee cuatro baterías con las cuales se alimenta la totalidad del sistema. La disponibilidad de energía en un momento dado esta determinada por el nivel de carga de dichas baterías. EMS tiene la responsabilidad de gestionar el consumo energético, priorizando unos subsistemas sobre otros. Se busca mantener las secciones más críticas en funcionamiento el mayor tiempo posible.

La gestión del consumo energético se realiza mediante la definición de tres modos de energía: `DEPLOY_WAIT`, `RECOVERY` y `SAFE`.

En modo `DEPLOY_WAIT` únicamente EMS debe estar encendido, siendo su única función optimizar la energía que se obtiene de los paneles solares para cargar las baterías. No se realizan transmisiones con la baliza morse. El satélite inicia en este modo y se mantiene en el mismo hasta 15 minutos después de detectar que las antenas hayan sido desplegadas o hasta 24 horas tras el encendido en caso de no desplegar antenas. Luego se pasa a modo `RECOVERY`.

Durante el modo `RECOVERY` EMS y la baliza morse están encendidos. La optimización energética continúa. Cada 5 minutos se realizan transmisiones con la baliza morse transmitiendo el estado del módulo.

En modo `SAFE` todas las funcionalidades del satélite se encuentran habilitadas. La optimización energética continúa. Cada 1 minuto se realizan transmisiones con la baliza morse transmitiendo el estado de todos los módulos encendidos. Solo los módulos I^2C, MCS, COMM1 y COMM2 son encendidos automáticamente. ADCS, PAYLOAD y los transmisores de banda-S son habilitados para poder ser encendidos mediante órdenes desde la Tierra.

El nivel de energía de las baterías es deducido a partir del nivel de tensión a la cual trabajan. La tensión es periódicamente medida en los modos `RECOVERY` y `SAFE`. Según el nivel de tensión se realizan las transiciones entre uno y otro modo.

Es posible determinar si un modulo esta en cortocircuito mediante la lectura de un puerto. Los módulos en cortocircuito son apagados y no pueden volver a ser encendidos. Se definen entonces tres estados en los que puede estar un módulo:

- `ON` módulo encendido.
- `OFF` módulo apagado.
- `FAIL` módulo en cortocircuito.

Debido a que un cortocircuito puede dejar de existir tras un tiempo, los módulos en estado FAIL pueden ser nuevamente encendidos luego de trascurridas 24 horas. Si el módulo es encendido y no se detecta cortocircuito, su estado vuelve a ser `ON`.

Los comandos desde la Tierra son transmitidos a EMS por el MCS. Ante la solicitud de encender o apagar un módulo EMS debe actuar teniendo en cuenta el modo de energía y el estado actual de los módulos de los cual depende, incluyendo el propio EMS. Por ejemplo, ADCS no puede ser encendido si MCS esta en cortocircuito.

PAYLOAD puede solicitar el encendido o apagado de los transmisores de banda-S. Debido a que los transmisores tienen un alto consumo energético, deben ser apagados automáticamente tras 100 segundos.

La gestión de encendido y apagado de módulos implica por tanto controlar el nivel de energía, determinar el modo actual, encender y apagar módulos según el modo, recibir comandos de encendido y apagado de los módulos no automáticos, detectar cortocircuitos y restaurar los módulos en falla.

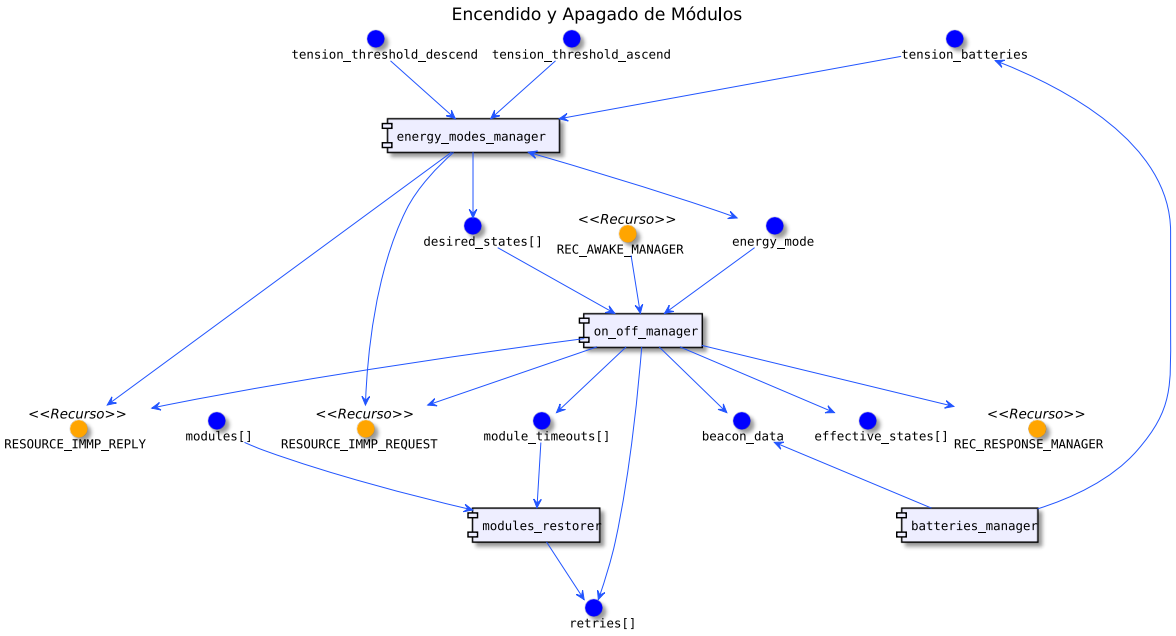

Figura 5.1. Componentes encargados de la gestión de encendido y apagado de módulos.

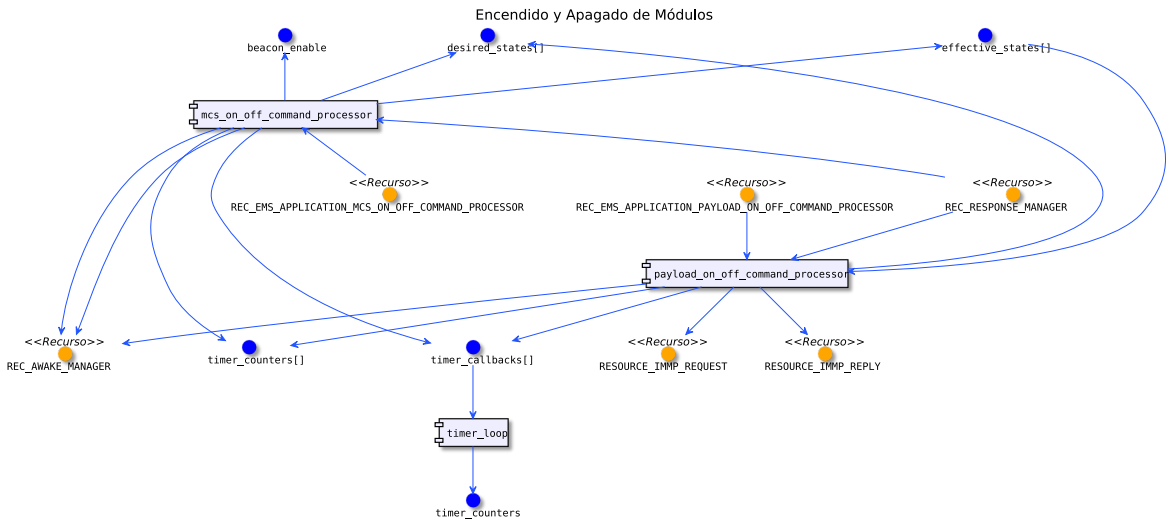

Figura 5.2. Componentes encargados de la recepción de comandos de encendido y apagado.

Las figuras 5.1 y 5.2 muestran los distintos componentes que implementan esta gestión. Se cuenta en total con 7 aplicaciones. Se ha usado el lenguaje de *diagramas de componentes* de UML para representar la interacción entre las aplicaciones. El uso de UML aquí es informal y no necesariamente adhiere los criterios estándar del lenguaje. Los elementos que aparecen en las figuras son los siguientes:

- Aplicaciones: se representan como *componentes* (rectángulos)
- Variables de estado: se representan como *interfaces* (círculos azules). Los nombres que terminan en `[]` indican vectores. Una flecha desde una aplicación a una variable indica que la aplicación escribe la variable. Una flecha desde una variable hacia una aplicación indica que la aplicación lee la variable.
- Recursos: se representan también como interfaces, pero en color amarillo y con la etiqueta «`Recurso`» encima.

La aplicación `batteries_manager` se encarga de medir periódicamente el nivel de tensión de las baterías y almacenarlo en la variable de estado `tension_batteries`. Este valor de tensión es usado por el `energy_modes_manager` para determinar el modo de energía actual. Cuando la tensión supera el valor indicado en la variable `tension_threshold_ascend` se pasa a estado `SAFE`, cuando desciende por debajo de `tension_threshold_descend` se vuelve a `RECOVERY`. El modo de energía actual se registra en la variable `energy_mode`.

Las aplicaciones `mcs_on_off_command_processor` y `payload_on_off_command_processor` reciben comandos desde MCS y PAYLOAD respectivamente. En ellos se indican qué módulos deben ser encendidos o apagados según las solicitudes recibidas desde la Tierra. El estado deseado de cada módulo es actualizado por estas aplicaciones en la variable de estado `desired_states`.

La aplicación `on_off_manager` lee el estado deseado de cada módulo y el modo de energía actual. A partir de estos datos y la relación de dependencia entre módulos se encienden y/o apagan los módulos que corresponda. El estado deseado es solo una indicación, la decisión de prender o apagar un módulo es tomada por la aplicación. A su vez se corrobora que los módulos no estén en cortocircuito.

Cada módulo posee un número de reintentos de encendido asociado almacenado en la variable `retries`. Si no es posible encender un módulo sin registrar cortocircuito se decrementa en 1 su número de reintentos y se vuelve a probar encender. Al llegar a cero finalmente se le asigna el estado `FAIL`.

Por su parte, la aplicación `modules_restorer` habilita a los módulos que estén en `FAIL` por más de un día a poder ser encendidos por `on_off_manager` incrementando su contador de reintentos. El tiempo en estado `FAIL` es mantenido en la variable `module_timeouts`.

El tiempo que permanece encendido un transmisor de banda-S, el cual está limitado, es controlado por la aplicación `timer_loop`, responsable de proveer temporizadores de uso genérico a todo el módulo.

Optimización energética (mppt)

Sin importar el modo de energía, EMS debe en todo momento maximizar la cantidad de energía que es extraída de los paneles solares y almacenada en las baterías. Al variar la tensión de trabajo de los paneles solares, la potencia generada cambia formando una curva cuya forma depende de la intensidad registrada en cada instante, la cual varía con el tiempo. Debido a que esta curva posee un único máximo es posible, situando la tensión de trabajo en el valor apropiado, maximizar la potencia generada.

Se cuenta con tres aplicaciones, `mppt X`, `mppt Y` y `mppt Z` encargadas de optimizar la tensión de trabajo de los paneles solares presentes en las caras perpendiculares a los ejes x, y, z respectivamente. Las tres tienen un comportamiento análogo siendo instancias de un mismo código. (ver Figura 5.3). Por más detalles del algoritmo utilizado para optimizar la tensión ver la descripción de la aplicación `mppt` en el apéndice.

Comunicación con la Tierra (baliza morse)

EMS cuenta con un transmisor a través del cual se emite una señal que sirve como baliza. La señal contiene información de estado del satélite codificada en morse. El envío periódico de esta señal, denominada en el resto del documento como baliza morse, es responsabilidad de EMS. Aún en modo `RECOVERY`, cuando la

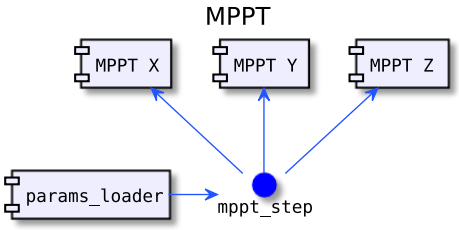

Figura 5.3. Componentes que realizan la optimización energética.

mayor parte del satélite se encuentra apagada, la baliza debe seguir funcionando. Esta señal hace posible conocer desde la Tierra el estado del satélite en todo momento.

Los datos transmitidos por la baliza dependen del modo de energía. En modo `RECOVERY` se envía una baliza básica que contiene únicamente los datos pertenecientes al EMS, pues es el único módulo encendido. Los mismos son:

- Identificador del satélite
- Voltaje de baterías
- Tensión batería 1
- Tensión batería 2
- Tensión batería 3
- Tensión batería 4
- Vector de Estado de los módulos

En modo `SAFE`, se envía la baliza completa, que contiene además los siguientes datos, provenientes de otros los módulos:

- RSSI de COMM1
- Temperatura 1 de COMM1
- Temperatura 2 de COMM1
- RSSI de COMM2
- Temperatura 1 de COMM2
- Temperatura 2 de COMM2
- Último mensaje de MCS
- Estado de actitud de ADCS

Es posible solicitar a EMS el envío de un mensaje de usuario a través de la baliza. En caso de existir un mensaje de usuario para enviar, a la baliza completa se le concatena dicho mensaje formando la baliza extendida. Al indicar el mensaje de usuario se indica el número de repeticiones para dicho mensaje. Para cada mensaje, se envían tantas balizas extendidas consecutivas como repeticiones se le hayan asignado.

Cada módulo es responsable de enviar periódicamente a EMS una actualización de los datos que le corresponden en la baliza completa. Si un módulo esta apagado, los campos que le corresponden son llenados con un valor por defecto. Es responsabilidad de EMS escuchar por estos mensajes de actualización.

Además del envío de la baliza, el transmisor también es usado para enviar un mensaje de confirmación (ROGER) ante la recepción de un comando desde la Tierra por parte de MCS. EMS escucha por solicitudes de envío de ROGERs desde MCS y gestiona el uso del transmisor para intercalar el envío de ROGERs con la transmisión de la baliza.

Existen 4 aplicaciones encargadas de gestionar el uso del transmisor y construir la baliza como se puede apreciar en la Figura 5.4.

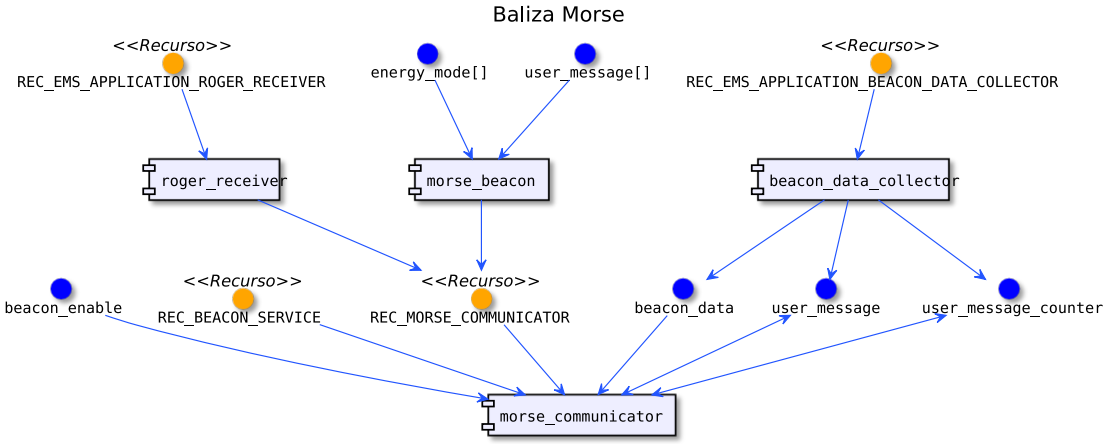

Figura 5.4. Componentes encargados de la transmisión de la baliza morse.

La aplicación `morse_beacon` se encarga de gestionar el envío de la baliza morse. Esta define la frecuencia de envío y el tipo de baliza (simple, completa o extendida) según el modo de energía. Los datos para la baliza simple también son recolectados por esta aplicación. Los datos de la baliza completa son recolectados por la aplicación `beacon_data_collector`. Esta escucha por los mensajes provenientes de otros módulos y actualiza la baliza con estos datos. La aplicación `roger_receiver` se encarga por su parte de escuchar por las solicitudes de envío de ROGERs por parte de MCS. Las aplicaciones `morse_beacon` y `roger_receiver` señalizan a la aplicación `morse_communicator` utilizando un recurso del kernel y le solicitan la transmisión de la señal. El `morse_communicator` construye el mensaje a transmitir (indicado en la señalización) e invoca un servicio que se encarga de transmitir dicho mensaje.

El servicio que realiza la transmisión es interruptivo y se compone por dos capas. La primera se encuentra implementada en el módulo `beacon_service`. Esta itera sobre cada carácter morse del mensaje y lo traduce en la secuencia de puntos y rayas correspondientes. Luego, para cada componente de un carácter, se solicita a la segunda capa `ems_hal_beacon` la transmisión de la señal, indicando el tiempo que debe estar alta según se trate de un punto o una raya. El amplificador y el modulador de la señal deben ser encendidos antes de cada transmisión y apagados al finalizar. Una máquina de estados interruptiva es usada para controlar los tiempos de espera requeridos por los periféricos para su encendido y apagado.

Los caracteres morse son codificados por el módulo `morse_encoder` como una palabra de 16 bits dónde los últimos cuatro bits indican la cantidad de componentes y los primeros doce codifican los mismos, siendo 0 el valor del punto y 1 el valor de la raya. De esta forma con un simple shift a la izquierda es posible iterar sobre los componentes de un carácter, manteniendo en un mínimo el espacio de memoria requerido para procesar el mensaje codificado. Este modo de codificación esta basado en lo implementado por Chris Noe [25].

Comunicación con otros módulos

EMS se comunica con otros módulos a través del bus I^2C. Para ello se utilizan las aplicaciones provistas por el protocolo immp (`immp_rx` e `immp_tx`), las cuales proveen la interfaz para el envío y recepción de mensajes a través de los recursos `RESOURCE_IMMP_REPLY` y `RESOURCE_IMMP_REQUEST`. Estas aplicaciones fueron desarrolladas por el responsable de integración.

En total existen 4 aplicaciones cuya función primordial es la recepción y transmisión de datos desde y hacia otros módulos. Ver Figura 5.5.

La aplicación `clock` provee la hora del sistema al resto del satélite. La aplicación `telemetry_report` envía datos finos del estado del módulo a MCS, para ser transmitidos a la Tierra. La aplicación `params_loader` carga desde MCS los parámetros del módulo. La aplicación `i2c_monitor` monitorea el estado del bus.

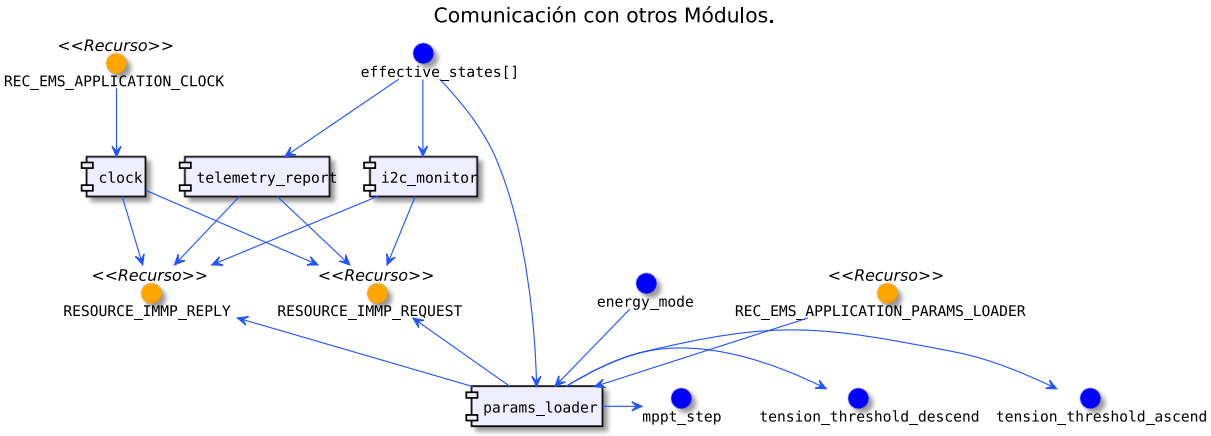

Figura 5.5. Componentes dependientes de la comunicación con otros módulos.

Por más detalles ver el apéndice “Aplicaciones de EMS”.

5.4. Verificación

En esta sección se presenta el proceso de verificación de software realizado para EMS. Se describen las tecnologías utilizadas, casos de prueba definidos y dificultades encontradas.

5.4.1. Verificación unitaria

Las pruebas unitarias de EMS se desarrollaron haciendo uso de la herramienta **MinUnit** ([22]). Se trata de un framework de verificación unitaria para C, similar a otras herramientas como ser **JUnit** o **CppUnit**. Es extremadamente simple y está compuesto unicamente de un archivo header en que define los macros `mu_assert` y `mu_run_test`. El primero permite exigir que se cumpla una condición y detener la prueba en caso contrario, el segundo ejecuta un caso de prueba. El archivo header no utiliza ninguna biblioteca externa lo cual lo hace portable a cualquier implementación de C, y provee una funcionalidad completa a pesar de su simplicidad.

Se desarrolló una suite de prueba para cada aplicación de EMS. Todas las suite de prueba unitaria se compilaron y ejecutaron en PC. Cada suite es un programa que define un conjunto de casos de prueba, una función de inicialización y una función de finalización. Para cada caso de prueba se ejecuta la función de inicialización, el caso de prueba en sí y la función de finalización. De esta forma cada caso ejecuta en un entorno aislado de los demás, y las fallas que se produzcan en un caso no invalidan los que se ejecutan posteriormente.

Todas las aplicaciones consisten en un bucle infinito que ejecuta una lógica dada cada cierto tiempo. Dependiendo del caso esta lógica puede involucrar leer y/o modificar variables de estado e invocar primitivas de HAL. Cada aplicación provee una función de nombre `<nombre_aplicacion>_control` que implementa un paso de control (ej. `params_loader_control`). Cada caso de prueba de una suite configura las variables de estado de una cierta forma, ejecuta un paso de control y verifica que el estado final sea el esperado. Las suite de prueba proveen implementaciones stub de las funciones del HAL. Esto permite simular en los diferentes casos el comportamiento de los dispositivos.

Para la elaboración de los casos de prueba, se buscó cubrir los casos más relevantes, lo cual incluye casos de borde en las lecturas de los periféricos, transiciones de estado, falla de los módulos, etc. El criterio utilizado fue de caja blanca. Se estudiaron las bifurcaciones del código en varias aplicaciones y se elaboraron los casos para cubrirlas, eligiendo los valores apropiados para las variables de estado leídas y las primitivas del HAL.

A modo de ejemplo se presentan a continuación dos casos de prueba, tomados de la suite para la aplicación `batteries_manager`.

Caso: test1

Estado inicial:

- configurar stub de HAL para que la próxima lectura de energía para todas las baterías sea 100.
- configurar stub de HAL para que la próxima lectura de tensión de baterías sea 100.
- todas las variables de estado de EMS en sus valores por defecto.

Acción: ejecutar un paso de control.

Estado final esperado:

- el timeout para reprogramar protecciones todavía no ha ocurrido.
- el valor de tensión de baterías en la baliza completa no ha cambiado.
- los valores de energía de baterías en la baliza completa no han cambiado.
- el valor de tensión de baterías no ha cambiado.
- las protecciones no han sido reprogramadas.

Caso: test2

Estado inicial:

- configurar stub de HAL para que la próxima lectura de energía para las baterías 2, 3 y 4 sea 100 y para la batería 1 devuelva un valor por encima del rango esperado.
- configurar stub de HAL para que la próxima lectura de tensión de baterías sea 100.
- todas las variables de estado de EMS en sus valores por defecto.

Acción: ejecutar un paso de control.

Estado final esperado:

- el timeout para reprogramar protecciones todavía no ha ocurrido.
- el valor de tensión de baterías en la baliza completa no ha cambiado.
- los valores de energía de las baterías 2, 3 y 4 en la baliza completa no han cambiado.
- el valor de energía de la batería 1 en la baliza completa indica que está en el rango más alto.
- el valor de tensión de baterías no ha cambiado.
- la batería 1 ha sido reprogramada.

Las pruebas unitarias permitieron detectar y corregir un error en la aplicación `morse_communicator` y dos errores en la aplicación `on_off_manager`.

5.4.2. Verificación de integración

La verificación de integración apunta a detectar errores en la interacción entre los componentes, lo cual en el contexto del EMS se traduce a verificar la comunicación y sincronización entre aplicaciones. Con el fin de probar diferentes combinaciones de ellas, se desarrolló un conjunto de programas de prueba en donde se carga y ejecutan subconjuntos de aplicaciones de EMS.

Para verificar la sincronización entre aplicaciones se implementó un mecanismo basado en las siguientes

ideas:

- Cada una de las aplicaciones bajo prueba incluye dentro de su flujo código que señaliza el pasaje por un cierto punto de interés (a los que llamamos *checkpoints*). Esta lógica adicional se compila sólo en las pruebas de integración, omitiéndose en la versión de vuelo del software, lo cual se logra por medio de macros y compilación condicional.
- Cada programa de prueba de integración se compone de un subconjunto de aplicaciones de EMS y una aplicación auxiliar que dirige la prueba (*aplicación de seguimiento*). Esta aplicación no forma parte del sistema, es definida en el código de la prueba. En todo momento esta aplicación es capaz de establecer una de las demás aplicaciones como *aplicación bajo seguimiento*. Una vez seleccionada esta aplicación puede realizar una espera en forma sincrónica para que la *aplicación bajo seguimiento* pase por el próximo *checkpoint*.

La *aplicación de seguimiento* especifica entonces un guión de prueba, realizando cambios a las variables de estado y esperando que las demás aplicaciones ejecuten ciertas secciones de interés dentro de su flujo. A modo de ejemplo, una prueba de este estilo es asignar el estado deseado de un módulo en encendido, esperar que la aplicación `on_off_manager` ejecute un ciclo de control y verificar que luego el módulo está encendido.

A diferencia de las pruebas de unitarias, que se ejecutaron en PC, las pruebas de integración fueron construidas sobre el microkernel ([10]). El mismo fue desarrollado específicamente para el microcontrolador MSP430 y no puede compilarse en PC, por lo cual las pruebas de integración necesitaron ser compiladas en **IAR**.

Los programas de prueba imprimen mensajes a la salida estándar, en la terminal de ejecución. Si bien era posible ejecutar estos programas en un microcontrolador MSP430 real, se encontró que al hacerlo la transferencia de la salida del programa a través de la interfaz de programación del microcontrolador producía una latencia significativa al ejecutar. Se decidió entonces correr las pruebas de integración en el entorno de simulación de **IAR**. Esto permitió utilizar el código específico de MSP430 del microkernel y tener una ejecución más ágil.

Las pruebas de integración permitieron detectar y corregir:

- un error en la aplicación `on_off_manager`
- un error en la aplicación `params_loader`
- un error en la aplicación `beacon_data_collector`
- un error en el stub del HAL
- un error en la aplicación `telemetry_report`
- un error en la aplicación `energy_modes_manager`
- un error en la configuración de los recursos.

5.4.3. Dificultades encontradas

Las pruebas de integración se usaron para probar la ejecución de varias aplicaciones en forma concurrente. Cada programa de prueba establece una serie de pasos a seguir que involucra esperas sincrónicas. Cuando el evento por el cual se espera no ocurre, el programa de prueba se bloquea hasta que termina por timeout. Cuando esto sucede no es posible identificar de forma instantánea el punto en donde se ha bloqueado el programa. Por esta razón es necesario utilizar el depurador de **IAR** en conjunto con la salida de log para determinar dónde está el problema. Este proceso es muy laborioso.

Inicialmente las pruebas de integración se realizaron sin incluir la aplicación MPPT, que al ser independiente del resto de las aplicaciones, no estaba directamente involucrada en sus casos de prueba. Llegado el momento de verificar el conjunto completo de aplicaciones, se encontró que su presencia generaba dificultades a la hora de depurar las demás, ya que al tratarse de un bucle muy ajustado toma el control del CPU con mucha frecuencia. Además la aplicación MPPT estaba instanciada tres veces pues el diseño prevía una instancia de la aplicación por eje a controlar, agravando el problema.

Al trabajar con el depurador de **IAR** se encontró un problema que se producía con frecuencia. En ocasiones al ejecutar un programa de prueba en el depurador el IDE mostraba un mensaje indicando que el stack pointer estaba en 0. La sesión quedaba bloqueada a partir de ese punto, no permitiendo detener el programa o establecer un breakpoint. Este problema no ocurría en forma determinística pues bajo las mismas condiciones algunas veces se manifestaba y otras no. No parecía tratarse de un problema en el programa bajo prueba sino de un bug del depurador del **IAR**. Cada vez que se presentaba era necesario cerrar el IDE y volver a abrirlo, comenzando el trabajo nuevamente.

Otro problema que ocurría con frecuencia al trabajar con el depurador del **IAR** consistía en un bloqueo de la interfaz del programa que dejaba de responder con 100 % de uso de CPU en uno de los núcleos. Estos problemas con el **IAR** no parecían presentarse en respuesta a ninguna acción en particular (más que el uso del depurador) y aparecían en forma aparentemente aleatoria, agregando un esfuerzo adicional al trabajo. En ocasiones este problema persistía incluso luego de cerrar el IDE y abrirlo nuevamente. Como *workaround* para esta situacióin se encontró que es útil borrar los archivos intermedios generados por el IDE (extensiones `*.tmp`, `*.sfr` y `*.dep`, y los directorios `settings` y `Debug`).

A la hora de instalar el software en el microcontrolador se usó la interfaz de programación **MSP430 USB Debug Interface MSP-FET430UIF**. Este dispositivo provisto por Texas Instruments permite instalar el programa en el microcontrolador a través de una conexión USB desde **IAR**. Al trabajar con esta interfaz ocurrieron problemas en los que la instalación del programa fallaba por un problema de comunicación con el microcontrolador. En ocasiones se trataba de un problema de versiones de firmware en la interfaz de programación, y se resolvía con instalar un firmware más reciente o más antiguo dependiendo del microcontrolador con el que se estuviera trabajando. En estos casos era necesario además sustituir el archivo `msp430.dll` de IAR por la versión correspondiente al firmware que se estaba instalando en la interfaz. También se encontró que en algunos casos la instalación de software era exitosa o no dependiendo del cable USB que se usara.

5.5. Validación

En esta sección se describen los resultados del trabajo de prueba y validación realizado por el responsable de integración y el equipo de EMS sobre el prototipo de software de EMS desarrollado por nuestro grupo. El trabajo del equipo anterior consistió en integrar el prototipo de software de EMS con el módulo HAL desarrollado paralelamente por ellos, y verificar el comportamiento del conjunto sobre el hardware objetivo. Una vez culminadas las pruebas sobre el prototipo se utilizó toda la experiencia generada para construir la versión final del software de EMS. Este desarrollo se llevó a cabo por el responsable de integración dado que no formaba parte del alcance de nuestro proyecto. Aunque este trabajo no forma parte de nuestro proyecto en sí, permite valorar la utilidad del mismo dentro del conjunto. A continuación se detalla la devolución realizada por el equipo en relación al prototipo del EMS y cómo este contribuyó al desarrollo final.

El trabajo de relevamiento y análisis de requerimientos se consideró de gran ayuda al permitir aclarar el problema en cuestión y establecer criterios de prueba claros. Al momento del comienzo de nuestro proyecto los requerimientos de software para el AntelSat estaban muy poco claros. El relevamiento de requerimientos realizado permitió detectar problemas que fueron dando lugar a requerimientos nuevos hasta obtener la especificación final que se tomó como referencia de ahí en adelante.

El equipo expresó que el código del prototipo estaba más segmentado de lo necesario en un microcontrolador de la talla del **MSP430**. La cantidad de módulos `.c` pasó de 42 en el prototipo a 11 en la versión final.

Respecto al beacon, se decidió no apagar el amplificador cuando la señal está baja, contrariamente a la decisión que se había tomado inicialmente al definir requerimientos. El motivo detrás de este cambio es que la decisión original agregaba complejidad al sofware asociada a las temporizaciones finas que la aplicación debía realizar, y no se justificaba por el ahorro de energía obtenido.

Se agregó un parámetro que indica la cadencia del beacon en palabras por minuto.

En cuanto a la funcionalidad de MPPT, se encontró que el nivel de tensión que fijaba el modo hardware, que era relativamente cercano al óptimo en los paneles de prueba, estaba muy por debajo del óptimo en

los paneles de vuelo, según reveló la caracterización de los mismos. Se decidió agregar un tercer modo de MPPT, modo software-fijo, en el cual se fija la tensión en un valor que es controlable por parámetro.

El diseño del prototipo prevía tres aplicaciones de MPPT, una para controlar cada eje. Estas fueron cambiadas por una única aplicación que se encarga de los tres ejes. El motivo de esta decisión está vinculado al manejo de las salidas PWM hacia los paneles, que son tres salidas separadas pero se generan con el mismo temporizador. El control desde tres aplicaciones generaba ruido en las salidas, que se elimina al pasar a una sola aplicación.

Al integrar el módulo COMM2 se observó que cuando el mismo está transmitiendo, la lectura de tensión de baterías en EMS se reducía considerablemente, a pesar de que la tensión real permanecía igual. Esto hacía que EMS detectara una baja de energía en baterías y cambiara a modo `RECOVERY`, apagando todos los demás módulos. Para resolver esto se implementó un filtro por software para la medida de tensión. Para cambiar de modo se requiere que dos medidas separadas 5 s estén ambas en el mismo rango. A su vez cada medida es un filtro de mediana realizado sobre 10 muestras consecutivas.

Se agregó a la máquina de estados de EMS un modo adicional llamado modo de programación, que pasó a ser el modo en el cual el módulo inicia. El propósito de este modo de operación es soportar el trabajo en la Tierra al proveer una manera de controlar los reguladores manualmente.

En la aplicación reporte de telemetría se encontraron problemas de alineación en la estructura que se usaba para enviar el reporte de telemetría a MCS (el compilador insertaba bytes de relleno entre algunos campos). Se solucionó definiendo los bytes de relleno como parte de la estructura.

Se agregó una aplicación de reporte que recopila periódicamente un conjunto de datos del EMS y los envía al log de MCS, con un intervalo controlable desde la Tierra. Las variables medidas son corriente de los paneles, tensión de los paneles, tensión de salida de MPPT, corriente del EMS (intensidad o consumo), corriente del COMM2 y de ADCS. Este reporte se puede iniciar y detener desde la Tierra.

En el prototipo existía una aplicación monitor I^2C que se encargaba de verificar regularmente el estado de salud de dicho bus. Esta aplicación realizaba tres intentos de enviar un mensaje a MCS separados por un segundo. Luego hacía lo mismo con COMM1 y COMM2. Si ocurrían tres errores de comunicación consecutivos se consideraba que el bus había quedado bloqueado y se reiniciaba el satélite. En la versión final se cambiaron los tres mensajes separados por un segundo a dos mensajes separados por un minuto, para que el satélite no se reiniciara cuando se estaba programando o depurando un módulo. Además el mensaje de prueba pasó a enviarse sólo a MCS.

La aplicación `on_off_manager` se reescribió y se fusionó con `batteries_manager`, en una nueva aplicación `energy_manager`. Se quitó la regla de que si MCS no está encendido entonces los COMM tampoco deben estarlo, para que el satélite pueda enviar el Roger aunque MCS no está encendido.

Las aplicaciones `mcs_on_off_command_processor`, `payload_on_off_command_processor`, `roger_receiver` y `beacon_data_collector`, la recepción de mensajes desde la Tierra relativos a la aplicación de reporte, el ingreso y salida del modo de programación y el control de los reguladores cuando se está en modo de programación se fusionaron en una única aplicación `command_processor`.

Se agregó el módulo `config_manager`. Este módulo define selectoras para las variables de estado, guarda los parámetros y gestiona los temporizadores para la restauración de módulos (funcionalidad desempeñada por la aplicación `modules_restorer` en el prototipo). El `config_manager` también realiza las temporizaciones relativas a la comunicación con los transmisores de Banda S e implementa un reinicio forzado del satélite cada 45 días como mecanismo de contingencia. Este módulo implementa un "temporizador" por software (análogo a la aplicación `timer_loop` del prototipo). El propósito de módulo `config_manager` es reducir el acoplamiento entre módulos, permitiendo que las aplicaciones se acoplen a él en lugar de acoplarse entre sí.

5.6. Lecciones aprendidas

Esta sección resume las experiencias adquiridas con el trabajo realizado en el Módulo de Gestión de Energía.

El uso de la salida estandar al desarrollar sobre el microcontrolador MSP430 es poco práctico. Los mensajes se transfieren al IDE a través de la interfaz de programación del microcontrolador lo cual presenta una latencia notoria (el texto aparece en pantalla a razón de pocos caracteres por segundo). Por este motivo al realizar pruebas se ejecutaron los programas en el entorno de simulación de **IAR** en lugar de en el microcontrolador.

El diseño del programa se realizó siguiendo el criterio de que cada componente fuera lo más simple posible, lo cual resultó en una gran cantidad de aplicaciones que interactúan entre sí. La complejidad del diseño impacta en gran medida a la hora de verificar.

El código del prototipo de EMS además de compilarse en el entorno **IAR** fue compilado para arquitectura Intel usando el compilador **GCC**, con todas las advertencias habilitadas (opción `-Wall`). Esto resultó útil para detectar en tiempo de compilación múltiples problemas que no surgían al compilar en **IAR** (este último muestra como advertencias problemas que se marcan como errores en **GCC**, y algunas advertencias marcadas por **GCC** simplemente no aparecen en **IAR**).

Si se trabaja con la biblioteca `stdio` de `C`, se debe seleccionar la implementación `CLIB` en la configuración del proyecto en **IAR**. Si se usa la implementación por defecto `DLIB` las invocaciones a `printf` con especificadores de formato de string pueden dejar la memoria corrupta.

La arquitectura de MPS430 no soporta accesos a memoria de más de un byte con direcciones no alineadas a palabra. Al manipular datos en memoria es necesario tener esto en cuenta. Si se realiza un acceso a una palabra de memoria no alineada el microcontrolador se reinicia, y si se está ejecutando con el depurador de **IAR** el mismo pierde la conexión con el microcontrolador.

Módulo de Determinación y Control de actitud

En esta sección se describe el trabajo realizado en el Módulo de Determinación y Control de Actitud. La sección **Objetivos** presenta una descripción breve de la funcionalidad del módulo. En **Requerimientos** se desarrollan los requerimientos relevados. A continuación se describe la solución desarrollada en la sección **Desarrollo**. La sección **Verificación** presenta el proceso de verificación realizado y en la sección **Validación** se documentan los resultados obtenidos al integrar el módulo desarrollado con el resto del software del satélite. Finalmente en **Lecciones aprendidas** se resume la experiencia ganada con el trabajo realizado en el Módulo de Determinación y Control de Actitud.

6.1. Objetivos

Al ser dejado libre en el espacio, el satélite gira libremente sobre sí mismo a una cierta velocidad angular. La carga científica sin embargo requiere que el vehículo se encuentre estable y orientado de forma que las cámaras y antenas apunten hacia el centro de la Tierra. Es responsabilidad del módulo de Determinación y Control de Actitud estabilizar el satélite y orientarlo en esta dirección. Para ello el módulo debe ser capaz de lograr los siguientes objetivos:

1. Determinar la velocidad angular y orientación del satélite con respecto a la Tierra a intervalos regulares de tiempo.
2. Controlar la velocidad angular y orientación del satélite a fin de lograr la estabilización y orientación deseadas.
3. Comunicar la información de estado del módulo al resto del satélite, incluyendo la actitud del vehículo.
4. Recolectar información a demanda de variables de interés y comunicar la misma a Control Principal.

Es por lo tanto necesario proveer de un componente de software que interactúe con los distintos sensores y actuadores del módulo, proveyendo la lógica necesaria para lograr el cumplimiento de sus objetivos. El alcance de este proyecto de grado no contempla la totalidad del software requerido por el módulo. A fin de optimizar los recursos, en acuerdo con el equipo de AntelSat se definió que la interacción con el hardware, en especial en lo referente al uso de los sensores y actuadores sería llevado a cabo por el responsable de integración y el responsable de ADCS, mientras que el software requerido para lograr la comunicación del ADCS con el resto del satélite estaría a cargo del responsable de integración. La implementación de los algoritmos de determinación y control de actitud sería realizada por el responsable de ADCS utilizando la herramienta MatLab.

Como se detalla en posteriores secciones, el ADCS cuenta con un microprocesador dedicado al cálculo de los algoritmos. La comunicación entre el procesador principal del ADCS y este segundo microprocesador o coprocesador, como se le hará referencia de aquí en adelante, así como la capa de software necesaria para la ejecución de los algoritmos generados en MatLab definen el alcance de este proyecto en lo que respecta a ADCS. La siguiente lista enumera los objetivos particulares a cumplir:

1. Crear un protocolo que permita la comunicación de los dos microprocesadores a través de un bus SPI.

2. Integrar el código generado en MatLab que provee los algoritmos de detección y control.
3. Crear un servicio de invocación remota de procedimientos (RPC) que provea la interfaz de ejecución de los distintos algoritmos en el coprocesador desde el procesador principal.

6.2. Requerimientos

En esta sección se presenta un resumen de los requerimientos relevados para el ADCS. Se resume el contenido del documento **ADCS Especificación de Requerimientos**.

6.2.1. Estados

El estado de actitud del satélite depende de la velocidad angular y la orientación del mismo en un momento dado. A fin de simplificar el análisis posterior de los requerimientos, se establece una serie de estados por los que puede pasar la actitud del satelite:

- `DETUMBLE`: Estado en el que el satélite se encuentra girando con gran velocidad angular (módulo mayor a 0.00175 rad/s). Observar que el umbral aplica al vector velocidad angular considerando los tres ejes, no sobre un eje individualmente.
- `NOMINAL`: Estado en el que la velocidad angular se encuentra por debajo del umbral definido por el estado `DETUMBLE` pero la orientación no es la deseada (Cara +Z apuntando al nadir).
- `NOMINAL_OK`: Estado en el cual la velocidad angular se encuentra por debajo del umbral definido por el estado `DETUMBLE` y los valores absolutos de los ángulos roll y pitch de la rotación que lleva el sistema de referencia SCB al sistema RPY son menores a 10 grados en valor absoluto. Una vez que se alcanza este estado la cara +Z en la cual se encuentra la cámara apunta hacia el nadir con un error menor de 10 grados.

6.2.2. Detumbling

ADCS debe ser capaz de determinar la velocidad angular del satélite a partir de las mediciones del campo magnético realizadas con el magnetómetro. A partir de la velocidad angular se debe determinar si el satélite se encuentra en estado `DETUMBLE`. En este estado ADCS debe utilizar periódicamente el algoritmo de detumbling B-dot para calcular los valores de corriente necesarios a aplicar en los magnetorquers. Se debe actuar sobre los magnetorquers usando estos valores con el fin de disminuir la velocidad angular. Este proceso debe continuar hasta que se pase del estado `DETUMBLE` al estado `NOMINAL`.

6.2.3. Determinación de actitud

ADCS debe determinar la actitud del satélite en los estados `NOMINAL` y `NOMINAL_OK` a fin de generar la entrada necesaria para los algoritmos de control. Se debe integrar el modelo de campo magnético usado en el proyecto de grado Sistema de Determinación de Actitud del grupo SensHor, descartando el algoritmo de determinación utilizado (TRIAD [38]) en favor del uso de otros dos algoritmos de determinación, uno determinista (QuEst [29]) y otro estadístico (Filtrado de Kalman [19]). La implementación de los algoritmos es provista por el responsable de ADCS.

6.2.4. Control fino de actitud

ADCS debe implementar los siguientes algoritmos de control fino de actitud: PID, LQR, PD_PR y Sliding Mode Controller. La implementación de los algoritmos es provista por el responsable de ADCS (Por más información ver [12]). Estos algoritmos proveen la salida necesaria para controlar la orientación del satélite cuando éste se encuentra en estado estable, buscando colocarlo con la cámara apuntando hacia el nadir con un error menor a 10°. Dicha salida es el dipolo magnético necesario para controlar al satélite. Esto debe ser traducido luego a los valores de corriente a aplicar sobre los tres magnetorquers.

Dado que la acción de los magnetorquers puede generar interferencia para el magnetómetro y viceversa, la actuación de los magnetorquers y el uso del magnetómetro deben estar mutuamente excluídos en el tiempo. Esto significa que al final de cada ciclo de control de actitud, debe dejarse en 0 la corriente de los magnetorquers antes de acceder al magnetómero para realizar una medición.

6.2.5. Ciclo de Control

La duración del ciclo de control, esto es, sensado, cálculo y actuación debe durar entre 500ms y 800ms.

6.2.6. Integración MSP430-ARM

Se debe implementar un protocolo de comunicación entre el MSP430 y el ARM que funcione sobre SPI. Se debe proveer la posibilidad de seleccionar los algoritmos de determinación y control a utilizar indicándolos en los parámetros y obteniendo como resultado la salida de los algoritmos ejecutados, junto con valores intermedios de interés. Los parámetros para todos los algoritmos pueden ser comunicados al ARM cuando este es encendido. Aspectos cambiantes como la hora deben ser pasados junto a las mediciones como entrada en cada ejecución de los algoritmos. Cuando el ARM no se encuentre ejecutando un algoritmo, este debe permanecer en estado de bajo consumo.

6.2.7. Hora y TLE

El ADCS debe mantener internamente la hora, solicitando periódicamente al MCS una actualización de la misma. Cuando se recibe de MCS una actualización de hora, ya haya sido solicitada o no, se debe actualizar la referencia interna con dicha hora. Cuando la hora del satélite no es confiable, ADCS será apagado, por lo que no deberían recibirse referencias de hora inseguras.

6.2.8. Telemetría

ADCS debe enviar regularmente a MCS los datos recabados de telemetría. Los datos a enviar por telemetría están indicados en forma detallada en el apéndice “Tablas de telemetría”.

Los mensajes de telemetría deben enviarse a MCS cada 5 minutos.

6.2.9. Mensajes aceptados desde otros módulos

ADCS debe poder recibir y responder los siguientes mensajes provenientes de otros módulos.

1. Actualización de Hora desde MCS. Puede ser en respuesta a una solicitud por parte de ADCS o no.
2. Parámetros de entrada desde MCS. MCS únicamente envía este mensaje tras recibir una solicitud desde ADCS.
3. Solicitud de estado de actitud desde cualquier módulo.

6.2.10. Mensajes enviados hacia otros módulos

ADCS debe poder enviar los siguientes mensajes dirigidos a otros módulos.

1. Solicitud de Hora al MCS.
2. Solicitud de Parámetros al MCS.
3. Estado de actitud hacia el módulo que envió la solicitud. Este mensaje debe contener:
   a) Estado de la máquina de estados de ADCS.

b) Timestamp del momento en que se envía el mensaje (4 Bytes).

c) Valores de roll, pitch y yaw actuales (3 enteros de 16 bits).

4. Reporte de Telemetría.

5. Estado de actitud hacia EMS. La tabla 6.1 muestra el contenido de este mensaje. La primer columna lista los parámetros que se envían, las columnas siguientes corresponden a sus valores posibles. Para algunos de los parámetros como "Vector de estado act" los valores son una enumeración. Para otros, cada columna representa un bit, como en el caso de "Vector de estado SS1". En todos los casos ADCS debe enviar el valor a EMS como un número mayor o igual que 0 y menor o igual a 9. Cada vez que ADCS envía a EMS un mensaje, envía uno solo de los vectores de datos indicando cuál es. Esto se debe hacer siguiendo la siguiente secuencia:

   a) Vector de estado act

   b) Vector de estado SS1

   c) Vector de estado SS2

   d) Vector de estado MT

   e) Vector de estado act

   f) Vector de estado GY

   g) Vector de estado MM

   h) Vector de estado Exp

   i) Razón de entrada al estado `CRASH`

VECTOR ESTADO ACT	DET	TO_DET	NOM	NOM_OK	TO_NOM	CRASH
VECTOR ESTADO SS 1	FT_SS_+X	FT_SS_-X	FT_SS_+Y			
VECTOR ESTADO SS 2	FT_SS_-Y	FT_SS_+Z	FT_SS_-Z			
VECTOR ESTADO MT	FT_MT_X	FT_MT_Y	FT_MT_Z			
VECTOR ESTADO GY	FT_GY_X	FT_GY_Y	FT_GY_Z			
VECTOR ESTADO MM	FT_MM					
VECTOR ESTADO EXP	EXP_MODE_ON					
CRASH REASON						

Tabla 6.1. Reporte de estado para el Beacon desde ADCS.

Las etiquetas `DET`, `TO_DET`, `NOM`, `NOM_OK` y `TO_NOM` se refieren a los estados `DETUMBLE`, `TIMEOUT_DETUMBLE`, `NOMINAL`, `NOMINAL_OK` y `TIMEOUT_NOMINAL` respectivamente. Los estados `TIMEOUT_DETUMBLE` y `TIMEOUT_NOMINAL` son estados de error a los que se ingresa si el algoritmo permanece demasiado tiempo en los estados `DETUMBLE` o `NOMINAL` respectivamente.

6.2.11. Eventos

ADCS debe enviar una notificación a MCS al momento de ocurrir cualquiera de los eventos descriptos en la tabla 6.2

Evento	Descripción	Parámetro 1	Parámetro 2
EV_ADCS_PERIPHERAL_FAULT	Un periférico falló	Qué periférico fue	
EV_ADCS_STATUS_CHANGE	Cambio de estado de ADCS	Nuevo estado	Causa de la transición (ej. para CRASH)

Tabla 6.2. Reporte de eventos de ADCS.

6.2.12. Parámetros

Los parámetros de ADCS son un conjunto de datos de entrada que regulan el comportamiento del módulo. Se definió la estructura de datos `adcs_parameters` que recopila estas variables. Algunos de los valores reales se codifican como enteros de 16 bits usando una cierta constante de proporcionalidad, para permitir que la estructura quepa en un solo mensaje del protocolo IMMP. A continuación se indican algunos de los campos de esta estructura. Algunos parámetros están vinculados al funcionamiento interno del algoritmo de control siendo usados solo en ese módulo, por lo que su función está fuera del alcance de este texto. Todos los valores de punto flotante son en simple precisión. Los campos para los que no se especifica unidad son cantidades adimensionadas. Ver el anexo "Parámetros ADCS" por la lista completa de parámetros con información más detallada.

- `to_det`: Entero sin signo de 32 bits conteniendo el timeout de detumbling en segundos. Si ADCS permanece en estado `DETUMBLE` más de este tiempo, se interrumpe el procesamiento y se va a un estado de error.
- `to_nom`: Entero sin signo de 32 bits conteniendo el timeout nominal en segundos. Si ADCS permanece en estado `NOMINAL` más de este tiempo, se interrumpe el procesamiento y se va a un estado de error.
- `s_frame_size`: Valor de punto flotante conteniendo el intervalo del lazo de control en segundos.
- `tol_w`: Umbral de velocidad angular para salir de estado `DETUMBLE` (0.00175 rad/s por defecto). El valor se codifica en un entero de 8 bits que indica el valor en radianes por segundo multiplicado por 10^5.
- `on_off_flags`: Mapa de 8 bits conteniendo banderas para habilitar o deshabilitar el uso de ciertos periféricos. Solo se usan los cuatro bits menos significativos. Del más al menos significativo las opciones son `mt_off_flag` (magnetorquers), `s_GYRO_OFF` (giróscopos), `MM_OFF_flag` (magnetómetro), `SS_OFF_flag` (fotodiodos).
- `algorithms_flags`: Mapa de 8 bits conteniendo banderas para seleccionar los algoritmos de determinación y control. Solo se usan los tres bits menos significativos. Del más al menos significativo las opciones son `DET_ATT_flag`, `s_NOMINAL_ALG_MSB` y `s_NOMINAL_ALG_LSB`. La bandera `DET_ATT_flag` selecciona el algoritmo de determinación, 0 indica Quest y 1 corresponde a UKF. Los bits `s_NOMINAL_ALG_MSB` y `s_NOMINAL_ALG_LSB` juntos representan un entero de 0 a 3 que indica el algoritmo de determinación a usar (0 indica LQR, 1 indica PID, 2 indica PD_PR y 3 indica Sliding Mode).

6.3. Diseño

Esta sección describe el diseño del Módulo de Determinación y Control de Actitud.

ADCS se diferencia del resto de los módulos del satélite en un aspecto fundamental. En lugar de poseer un único microcontrolador MSP430, este cuenta además con un coprocesador ARM Cortex M4. Durante

el proyecto de grado antecedente en el cual se implementó el algoritmo de determinación de actitud, se evaluaron las plataformas de hardware disponibles y se decidió usar el microcontrolador MSP430. Después de haber desarrollado y verificado el algoritmo de determinación sobre dicha plataforma los resultados mostraron que el MSP430 presenta serias limitaciones para realizar tareas intensivas en cálculo de punto flotante.

Esto llevó a que se decidiera (previo al comienzo del presente trabajo) la inclusión de un microcontrolador ARM en el ADCS, en adición al MSP430 ya presente. Se planteó una arquitectura donde el microcontrolador ARM actúa como coprocesador realizando cálculos matemáticos en base a lecturas de sensores tomadas por el procesador principal MSP430, devolviéndole a este último la salida que se debe enviar a los actuadores. La lógica de manejo de sensores y actuadores es implementada por el procesador principal, que además se encarga de la comunicación con los demás módulos del satélite.

Esta arquitectura introduce el desafío de la comunicación entre los microcontroladores, que debe realizarse a través de un bus SPI por el cual están interconectados. Así, uno de los principales objetivos de este proyecto fue el diseño e implementación de un protocolo de comunicación entre ambos procesadores.

En las siguientes secciones se presenta individualmente las funciones realizadas por cada uno de los procesadores, para luego abordar los protocolos de comunicación desarrollados. Por mayor detalle de las aplicaciones implementadas, incluyendo pseucódigos de las mismas, referirse a la sección 1.2: "Apendice B: Aplicaciones de ADCS" del documento adjunto "Apendices".

6.3.1. Definiciones comunes

Ciertas definiciones son de uso común al procesador principal y al coprocesador. Se definen dos estructuras de datos que permiten suministrar los datos de entrada al algoritmo de control y comunicar el resultado. La estructura `att_det_ctrl_input_t` contiene los datos de entrada para el algoritmo de control y sus campos son los siguientes:

- `current_time`: Entero de 32 bits que expresa el tiempo epoch actual (segundos transcurridos desde el 1ero de Enero de 1970 UTC). Unidad: segundos.
- `offset_time`: Entero de 32 bits que expresa los segundos transcurridos desde la última vez que ADCS inició (dato usado por el algoritmo de control para determinar si debe realizar detumbling o control fino). Unidad: segundos.
- `angular_velocity_x`, `angular_velocity_y`, `angular_velocity_z`: Tres valores en punto flotante que indican la lectura de velocidad angular del giróscopo. Unidad: radianes por segundo.
- `sun_plus_x`, `sun_minus_x`, `sun_plus_y`, `sun_minus_y`, `sun_plus_z`, `sun_minus_z`: Seis valores en punto flotante que indican la lectura de los fotodiodos +X, -X, +Y, -Y, +Z y -Z respectivamente. Unidad: milivoltios.
- `magnetic_field_x`, `magnetic_field_y`, `magnetic_field_z`: Tres valores en punto flotante que indican las lecturas de los magnetómetros X, Y y Z respectivamente. Unidad: microteslas.

La estructura `att_det_ctrl_output_t` contiene la respuesta del algoritmo de control y está compuesta de los siguientes campos:

- `ctrl_alg_num`: Entero de 32 bits que indica qué algoritmo de control se va a usar a continuación, siendo 0 si se encuentra ejecutando el detumbling (0 = DETUMBLE, 1 = PID, 2 = PID, 3 = PD_PR, 4 = Sliding Mode). Cualquier valor distinto de cero implica que el satélite se encuentra en estado nominal, el tipo especifico de algoritmo es meramente informativo.
- `nominal_ok`: Entero de 32 bits que indica si se debe pasar al estado `NOMINAL_OK`. El valor 1 significa sí, 0 significa no. Por seguridad, cualquier valor mayor a 0 es interpretado como sí.
- `light_est`: Entero de 32 bits que indica si es de día según las lecturas de los fotodiodos. El valor 1 significa sí, 0 significa no.
- `roll`, `pitch`, `yaw`: Tres valores en punto flotante que expresan la actitud del satélite según el algoritmo de determinación de actitud en coordenadas de Euler. Unidad: grados decimales.

- ang_vel_x, ang_vel_y, ang_vel_z: Tres valores en punto flotante que expresan la velocidad angular calculada. Unidad: radianes por segundo.
- q_B0_est_a, q_B0_est_b, q_B0_est_c, q_B0_est_d: Cuatro valores en punto flotante representando la actitud del satélite devuelta por el algoritmo de determinación de actitud, en forma de cuaternión. Unidad: adimensionado.
- position_x, position_y, position_z: Tres valores en punto flotante que expresan la posición del satélite en el sistema de referencia ECI. Unidad: kilómetros.
- velocity_x, velocity_y, velocity_z: Tres valores en punto flotante que expresan la velocidad del satélite en el sistema de referencia ECI. Unidad: metros por segundo.
- sun_model_x, sun_model_y, sun_model_z: Tres valores en punto flotante indicando el vector de sol calculado según el modelo solar, expresado en el sistema de referencia orbital. Unidad: adimensionado.
- magnetic_field_model_x, magnetic_field_model_y, magnetic_field_model_z: Tres valores en punto flotante indicando el vector de campo magnético calculado según el modelo magnético, expresado en sistema de referencia orbital. Unidad: teslas.
- v_sun_est_x, v_sun_est_y, v_sun_est_z: Tres valores en punto flotante indicando el vector de sol estimado a partir de las medidas de los fotodiodos, expresado en el sistema de referencia body. Unidad: Adimensionado.
- current_x, current_y, current_z: Tres valores en punto flotante que indican los niveles de corriente para los tres magnetorquers a aplicar en la iteración actual del lazo de control. Unidad: amperios.
- pwm_x_a, pwm_x_b, pwm_y_a,pwm_y_b, pwm_z_a, pwm_z_b: Enteros de 16 bits conteniendo los valores a escribir en los registros de control de los magnetorquers para obtener los niveles de corriente indicados en las campos current_x, current_y y current_z. Cada magnetorquer tiene dos registros de control, por lo que se necesitan seis valores. Observar que estos campos son la principal salida del algoritmo, ya que corresponden a los valores de actuación del bucle de control. Unidad: adimensionado.

Todos los valores de punto flotante son de precisión simple (32 bits).

Se define también una estructura tle_t para guardar el TLE. La misma contiene un arreglo de caracteres de largo 139. Este arreglo guarda dos líneas de 69 caracteres cada una, separadas por un salto de línea (un caracter LF, hexadecimal 0A).

6.3.2. Procesador Principal - MSP430

El trabajo de nuestro equipo en ADCS estuvo enfocado en el coprocesador y la integración entre los microcontroladores. Del lado del MSP430 esto implicó el desarrollo del lado maestro de los protocolos spimp y spirpc, quedando la lógica de aplicación a cargo del reponsable de integración. Sin embargo, cuando esta división de trabajo se acordó, ya se había desarrollado un prototipo de software para el procesador principal a nivel de aplicación. Este prototipo no se usó en la versión final de software de ADCS, por lo que el trabajo de verificación de ADCS no abarca este componente. Se incluye aquí la descripción del diseño de este prototipo por motivos de completitud.

Siguiendo la organización utilizada en EMS, se diseñaron un conjunto de aplicaciones, variables de estado y recursos que resuelven los requerimientos del procesador principal.

La comunicación de datos se resuelve asincrónicamente mediante el uso de las variables de estado, mientras que los recursos brindan la posibilidad de sincronización. A continuación se enumeran las variables de estado y recursos utilizados.

Variables de Estado

- adcs_state: indica el estado en el que se encuentra el ciclo de control. Los valores posibles son DETUMBLE, NOMINAL, NOMINAL_OK, TIMEOUT_DETUMBLE, TIMEOUT_NOMINAL, CRASH.
- timer_callbacks: arreglo de punteros a funciones usado por la aplicación timer_loop.

- `timer_counters`: arreglo de contadores usado por la aplicación `timer_loop`.
- `attitude_parameters`: estructura que contiene los parámetros provenientes de la Tierra. Es una instancia del tipo `adcs_parameters` descrito en la sección Requerimientos. Los mismos se envían a ARM como parámetros para los algoritmos de determinación y control.
- `attitude_tle`: estructura que contiene el tle proveniente de la Tierra. Este se envía a ARM como parámetro para los algoritmos de determinación y control.
- `crash_reason`: enumerado conteniendo la razón de entrada al estado `CRASH`
- `parameters_received`: booleano que indica si un nuevo conjunto de parámetros fue recibido desde MCS y no ha sido aún enviado al coprocesador.
- `tle_received`: booleano que indica si un nuevo tle fue recibido desde MCS y no ha sido aún enviado al coprocesador.
- `attitude_input`: estructura que contiene los datos de entrada para la ejecución de los algoritmos de determinación y control.
- `attitude_output`: estructura que contiene los datos de salida de la ejecución de los algoritmos de determinación y control.

Recursos

- `RESOURCE_IMMP_REQUEST`: recurso utilizado para enviar mensajes a la aplicación `immp_tx`, encargada de enviar mensajes por I^2C hacia otros módulos del satélite. Este recursos es manejado internamente por el módulo `immp`.
- `RESOURCE_IMMP_REPLY`: usado para recibir la respuesta de una operación de envío I^2C de parte de la aplicación `immp_tx`. Este recurso es manejado internamente por el módulo `immp`.
- `RSRC_COMMAND_PROCESSOR`: usado por la aplicación `command_processor` para recibir mensajes de otros módulos a través de I^2C.
- `RSRC_CLOCK`: usado por la aplicación `clock` para recibir mensajes de otros módulos a través de I^2C.
- `RSRC_PARAMS_LOADER`: usado por la aplicación `params_loader` para recibir mensajes de otros módulos a través de I^2C.
- `RSRC_SENSOR_DATA_READY`: usado para que el servicio de medida de sensores que forma parte del hal señalice a la aplicación invocadora una vez que las medidas están listas.
- `RSRC_CONTROL_READY`: usado por el bucle de control para señalizar que el cálculo de los algoritmos de determinación y control ha finalizado y se cuenta con una actualización de los valores de corriente para efectuar la actuación en los magnetorquers.

Al diseñar las aplicaciones de ADCS, a diferencia de en EMS, se priorizó mantener bajo el número de aplicaciones. Esto busca evitar el inconveniente encontrado en EMS al momento de realizar las pruebas de integración, dónde el alto número de aplicaciones involucradas en una misma tarea vuelve compleja su prueba. En lugar de subdividir tareas complejas en subtareas simples y realizarlas en varias aplicaciones, se decidió implementar una única aplicación que realice la tarea y separar en varios módulos las subtareas. De esta forma se mantiene la simplicidad de las pruebas unitarias, las cuales se realizan a nivel de módulos, sin incrementar el número de aplicaciones necesarias.

Las funcionalidades requeridas por el procesador principal se dividen en dos grupos:

- Comunicación con otros módulos
- Ciclo de Control

Las aplicaciones encargados de la comunicación con otros módulos son prototipos no funcionales de alto nivel, realizadas sobre un stub del HAL. No se contempla en el alcance la implementación de la interacción con los periféricos o la comunicación real con otros módulos. Estas aplicaciones pretenden servir de punto de partida para el desarrollo del software final del módulo. La aplicación encargada del ciclo de control por su parte, es un prototipo funcional que implementa la comunicación con el coprocesador, haciendo uso de los protocolos de comunicación desarrollados y otros módulos auxiliares. Las pruebas sobre el protocolo de comunicación utilizan esta aplicación en un ambiente aislado del resto del módulo.

Comunicación con otros módulos

Las tareas de comunicación con otros módulos se resumen en el siguiente listado:

- **Recepción de parámetros** La aplicación `params_loader` se encarga de la recepción de parámetros desde MCS. Los parámetros son almacenados en la variable de estado `attitude_parameters` para poder ser accedidos desde el resto del módulo.
- **Recepción de datos de órbita (TLE)** La aplicación `params_loader` se encarga también de la recepción de datos de órbita. El TLE es almacenado en la variable de estado `attitude_tle` para poder ser accedido desde el resto del módulo.
- **Envío de datos de telemetría** La aplicación `telemetry_report` recolecta los datos requeridos por telemetría y los envía periódicamente al MCS. Los mismos son recolectados principalmente a partir de las variables de estado `attitude_output` y `adcs_state`, así como también mediante accesos al hal.
- **Recepción de hora** La aplicación `clock` recibe actualizaciones de hora desde MCS, actualizando la referencia interna del RTC.
- **Envío de datos de baliza** La aplicación `beacon_report` envía a EMS los datos a ser transmitidos por la baliza morse.
- **Procesamiento de comandos** La aplicación `command_processor` responde a comandos desde otros módulos. Particularmente responde al pedido de estado de actitud proveniente de PAYLOAD.

Ciclo de Control

El principal objetivo del módulo ADCS es controlar la actitud del satélite a fin de lograr que las cámaras apunten al nadir en forma continuada. Se parte de la base de que inicialmente, luego de la puesta en órbita, el satélite se encuentra girando a una velocidad constante, por lo que lograr la actitud objetivo implica el realizar dos grandes pasos. El primero es el de frenado o detumbling, cuyo objetivo es el de disminuir la velocidad de giro del vehículo hasta acercarlo a una velocidad angular de cero. Una vez lograda una baja velocidad de rotación, el segundo paso busca realizar el control fino, donde se ajusta constantemente la actitud para tender el eje z hacia el nadir.

Durante ambos pasos el accionar básico es el mismo. Se realizan mediciones con los distintos sensores proveyendo de los datos de entrada necesarios para determinar la actitud actual, luego a partir de esos datos se computa dicha actitud y comparándola con la actitud deseada se computan los valores de corriente que deben ser impactados en los actuadores. Finalmente se controla la actitud mediante el accionar de los actuadores. Este ciclo de sensado, cómputo y actuación es repetido constantemente en lo que se denomina ciclo de control. El estado nominal deseado no se logra hasta que se hayan realizado suficientes iteraciones para que se completen los pasos de frenado y control fino. Una vez lograda la actitud objetivo, el ciclo continúa para mantener el satélite en este estado durante toda la vida del módulo.

La aplicación `control_loop` se encarga de mantener este ciclo de control. La misma es responsable de accionar el sensado, solicitar el cómputo al coprocesador y accionar la actuación una vez obtenidos los resultados. Esta aplicación hace uso de los siguientes módulos auxiliares para realizar su tarea:

- `attitude_control` Solicita el computo de los algoritmos de determinación y control de actitud. Los datos de entrada son obtenidos de la variable de estado `attitude_input` y los comunica al coprocesador mediante el uso del protocolo `SPI_RPC`. La respuesta es almacenada en la variable de estado `attitude_output`.
- `adcs_state_controller` Mantiene la máquina de estados de actitud. Realiza los cambios de estado del ciclo de control, encapsulando y manteniendo la coherencia entre el estado actual, los timers que controlan los eventos de Timeout y el envío de notificaciones a MCS. El estado de actitud es mantenido en la variable de estado `adcs_state`. La aplicación `timer_loop` es utilizada por este módulo para controlar el tiempo de permanencia en cada estado.
- `parameters_communicator` Se encarga de realizar la comunicación con el coprocesador a fin de transmitirle el conjunto de parámetros requeridos para los algoritmos de determinación y control,

así como también el TLE. Además provee la posibilidad de verificar que la comunicación con el coprocesador se encuentre funcional.

6.3.3. Coprocesador - ARM

El módulo `adcs_hal_coproc` implementa la interacción con el hardware. En el desarrollo del procesador principal se siguió el criterio de no usar bibliotecas externas para mitigar el riesgo de introducir defectos por software de terceros. Para el coprocesador, debido al escaso tiempo disponible y la falta de experiencia con la plataforma, se decidió usar una biblioteca externa para resolver la interacción con el hardware. La biblioteca **STM32F3 Standard Peripherals Library** provista por el fabricante del microcontrolador provee una interfaz de abstracción de hardware con soporte para todos los dispositivos del mismo. El módulo HAL `adcs_hal_coproc` consiste en una capa fina por encima de esta biblioteca que provee acceso a los dispositivos que son de interés para el proyecto, a saber:

- SPI: la interfaz de comunicación serial que implementa SPI en el microcontrolador.
- GPIOs: usados para tomar medidas precisas del tiempo de ejecución del algoritmo y el tiempo de comunicación SPI (ver sección de verificación).
- LEDs: usados como herramienta de depuración en las pruebas.

El coprocesador mantiene las siguientes variables de estado:

- `attitude_parameters`: estructura que guarda los parámetros de ADCS recibidos desde el procesador principal. Es del mismo tipo que la variable `attitude_parameters` del procesador principal, a pesar de que no todos sus campos son usados en el coprocesador. Se usa el mismo tipo para evitar duplicación de código.
- `attitude_tle`: estructura usada para guardar el TLE recibido desde el procesador principal.

El módulo `spimp_slave` implementa la parte del esclavo en la comunicación de mensajes por SPI con el procesador principal, mientras que `spi_rpc_slave` implementa la capa de aplicación de la comunicación. Ver la sección Protocolos por más detalles.

El componente `model` implementa los modelos utilizados en los algoritmos (IGRF como modelo del campo magnético de la Tierra, SGP4 como modelo orbital, y el modelo de movimiento del sol). En este componente se reutilizó parte del código desarrollado en el proyecto de Sistema de Determinación de Actitud por el grupo SensHor. A su vez, parte del software de este proyecto fue adaptado de la biblioteca que acompaña el texto "Fundamentals of Astrodynamics and Applications" de David Vallado ([37]). Para las funciones que se extrajeron, se compararon la versión original y la versión de SensHor para verificar los cambios realizados. En este proceso se eliminaron algunas funciones y variables que no se usan. El resto se tomó de la versión de SensHor sin cambios excepto por el uso de las variables de punto flotante en doble precisión como en la versión original de Vallado.

Se cambiaron los coeficientes usados por el modelo orbital IGRF por los valores de la última versión del algoritmo, publicada por el **National Geophysical Data Center** de la **National Oceanic and Atmospheric Administration**. Esta versión más reciente de los datos fue publicada posteriormente al proyecto del grupo SensHor.

El algoritmo de cálculo del vector de sol que había sido adaptado de la biblioteca de David Vallado se sustituye por la versión original. Se realizó a este código un cambio consistente en sustituir las invocaciones de la función `abs()` de la biblioteca estándar de C por `fabs()` (el código original usaba la versión para enteros aplicada a valores de punto flotante, causando una pérdida de precisión). Se notificó al autor de este problema y el mismo confirmó que se trataba de un bug. Dado que el código original de Vallado era C++ se sustituyeron los parámetros por referencia por punteros por valor.

Los algoritmos en sí fueron implementados en Matlab y traducidos a lenguaje C por medio del generador de código de dicha herramienta. Los cálculos se realizan en simple precisión. Este trabajo fue llevado a cabo por el responsable de ADCS Ing. Matías Tassano paralelamente a nuestro trabajo. El algoritmo desarrollado en Matlab consta de las siguientes entradas:

- Vector de campo magnético sensado expresado en sistema solidario al satélite.
- Vector de campo magnético de referencia (calculado por el modelo) expresado en Sistema Orbital.

- Valores sensados por fotodiodos.
- Vector de sol de referencia (calculado por el modelo) expresado en sistema de orbital.
- Valores sensados por giróscopo.

La salida incluye los siguientes datos:

- Valores de corriente para los magnetorquers X, Y y Z.
- Valores para los registros de control de los magnetorquers (seis valores, dos por cada magnetorquer).
- Identificador del algoritmo de control en uso actualmente (0 indica LQR, 1 indica PID, 2 indica PD_PR y 3 indica Sliding Mode).
- Bandera de si se está en estado `NOMINAL_OK`.
- Bandera indicando si es de día o no.
- Valores de roll, pitch y yaw (resultado del algoritmo de determinación).
- Estimación de cuaternión de actitud (resultado del algoritmo de determinación).
- Estimación de velocidad angular.
- Estimación de vector de sol.

La integración del algoritmo Matlab y los modelos dentro del programa C que ejecuta el microcontrolador y la comunicación entre este programa y el procesador principal fueron las tareas desarrolladas por nuestro equipo en ADCS. El algoritmo Matlab y los modelos se integran en el módulo `att_det_ctrl`. Esta integración requiere ciertas conversiones en los sistemas de coordenadas y la representación de las matrices. La información de entrada debe ser convertida al formato en que Matlab la espera, y su salida debe luego ser convertida del formato en que Matlab la devuelve al formato con el cual se envía hacia el procesador principal. La integración se realizó varias veces, dado que al algoritmo Matlab fue desarrollado en sucesivas versiones, a veces con diferencias de API como el agregado de parámetros de entrada y salida. El módulo `att_det_ctrl` encapsula el algoritmo Matlab y la conversión de datos asociada detrás de una interfaz simple.

El módulo `att_det_ctrl` recibe como entrada una estructura de tipo `att_det_ctrl_input_t` conteniendo entre otras cosas las lecturas de los sensores y devuelve una estructura `att_det_ctrl_output_t` conteniendo los valores de actuación para la presente iteración del lazo de control. Los cálculos realizados se pueden describir en los siguientes pasos:

- Calcular la fecha juliana a partir del tiempo epoch.
- Calcular posición y velocidad del satélite en sistema ECI usando algoritmo SGP4 en base a la fecha juliana y el TLE.
- Transformar posición de ECI a ECEF.
- Calcular latitud y longitud expresadas en grados en base a posición en ECEF.
- Calcular vector de campo magnético en sistema ECEF usando modelo IGRF en base a la fecha juliana, la latitud y la longitud.
- Transformar el vector de campo magnético de ECEF a ECI.
- Transformar el vector de campo magnético de ECI a Sistema Orbital.
- Calcular vector de sol en ECI en base a la fecha juliana.
- Transformar vector de sol de ECI a Sistema Orbital.
- Invocar algoritmo Matlab con los datos calculados anteriormente y las lecturas de sensores provenientes del procesador principal.

El kernel empleado en los módulos MSP430 no se utilizó en el desarrollo del coprocesador. El módulo `command_loop` implementa la lógica principal del programa, la cual consiste en un bucle infinito que espera mensajes del procesador principal y los procesa. Cuando el programa se encuentra esperando la llegada de un mensaje desde el procesador principal el coprocesador entra en modo de bajo consumo. Esto se hace por medio del modo *sleep* del microcontrolador STM32F3, en el cual se detiene el reloj del

Figura 6.1. Interfaz de programación ST-Link. Autor de imagen: STMicroelectronics

CPU pero manteniendo el estado de la memoria y los puertos de entrada / salida. La entrada al modo de bajo consumo se hace por medio de la instrucción `WFI` (*Wait For Interrupt*) de forma que cualquier interrupción hace que se vuelva al estado de ejecución (ver [33]).

El desarrollo en el microcontrolador ARM se llevó a cabo sobre sistema operativo Ubuntu, usando el *toolchain* GNU ARM Embedded ([15]). El mismo provee una implementación del compilador de C y herramientas asociadas para arquitectura ARM, y se distribuye con licencia GPL. Incluye soporte para cálculos de punto flotante en hardware a diferencia de otros compiladores similares para ARM que tienen problemas en este aspecto, y se puede instalar fácilmente en Ubuntu por medio de binarios precompilados. La instalación del *toolchain* en base a compilar los fuentes es una tarea muy laboriosa debido a que el paquete tiene una gran cantidad de dependencias, por lo que la instalación en base a binarios precompilados es la alternativa mas recomendable para la mayoría de los casos.

La interfaz de programación usada para instalar software en el microcontrolador ARM es el ST-Link. En el kit de desarrollo de ARM usado esta interfaz está integrada a la placa principal y se accede mediante un puerto mini-USB. En la placa de vuelo se usa el dispositivo externo mostrado en la Figura 6.1. Como software de programación se usó **Open On-Chip Debugger** ([26]), una herramienta de código abierto que soporta programación y depuración para distintos tipos de microcontroladores e interfaces. Los procesadores soportados incluyen arquitecturas ARM y MIPS entre otras.

6.3.4. spimp

El protocolo de comunicación entre el procesador principal y el coprocesador consta de una arquitectura en dos capas. **Serial Peripheral Interface Message Protocol** abreviado **spimp** constituye la capa inferior del protocolo. Resuelve la transferencia de mensajes (cadenas de bytes arbitrarias) a través del bus SPI. También se encarga de detectar errores de comunicación en ambos sentidos y asegurar al receptor la integridad del mensaje. **Serial Peripheral Interface Remote Procedure Call** abreviado **spirpc** implementa la capa de aplicación del protocolo. Transfiere los mensajes necesarios para resolver los requerimientos específicos de la aplicación que en nuestro caso son realizar la invocación remota de un algoritmo con una cierta entrada y obtener la salida resultante. Esta sección describe el protocolo **spimp**, mientras que **spirpc** se detalla en la sección siguiente. Una explicación detallada de las máquinas de estado que implementan el protocolo spimp puede ser encontrada en la sección 1.3: “Apendice C: spimp - Máquinas de Estado” del documento adjunto “Apéndices”.

Acerca del Bus SPI

Serial Peripheral Interface o Bus SPI es un estándar para la comunicación de datos en serie en forma sincrónica entre dispositivos electrónicos. Trabaja en modo maestro/esclavo y permite la incorporación de múltiples esclavos mediante el uso de líneas de selección independientes. La sincronización y transmisión de datos se realiza a través de cuatro líneas de interconexión:

- `SCLK` (Clock): Transmite la señal del reloj que marca la sincronización. Con cada pulso del reloj se lee o se envía un bit.
- `MOSI` (Master Output Slave Input): Salida de datos del Maestro y entrada de datos al Esclavo.
- `MISO` (Master Input Slave Output): Salida de datos del Esclavo y entrada de datos al Maestro.
- `SS` (Slave Select): Selecciona el Esclavo a activar.

La señal del reloj es provista por el Maestro, quien selecciona un esclavo bajando a tierra la señal SS que le corresponda. En cada pulso del reloj se realiza una transmisión bidireccional: un bit es escrito por el Maestro y leído por el esclavo en la línea MOSI. Un bit es escrito por el Esclavo y leído por el Maestro en la línea MISO.

Ambas transmisiones son simultáneas y siempre ocurren, siendo el Maestro el único capaz de iniciar una transmisión por ser el que controla la señal del reloj.

Típicamente se utilizan registros de desplazamiento de diversos tamaños, ocho bits por ejemplo, para el intercambio de datos, uno en el Maestro y otro en el esclavo. Se interconectan con las líneas MISO y MOSI formando un buffer circular distribuido (Figura 6.2). Durante una transmisión, los registros de desplazamiento del Maestro y del esclavo son desplazados un lugar. En esta transferencia se envía el bit menos significativo del maestro a través de la línea MOSI que es escrito en el bit más significativo del registro del Esclavo, y se envía el bit menos significativo del esclavo a través de la línea MISO que es escrito en el bit más significativo del registro del maestro. Según se configure, la transmisión puede comenzar por el bit más significativo en su lugar.

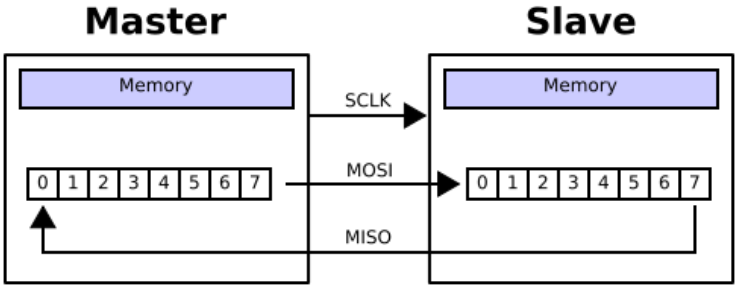

Figura 6.2. Configuración típica de registros de desplazamiento. Imagen publicada bajo licencia *Creative Commons Attribution-Share Alike 3.0 Unported. Autor de imagen: CBurnett*

Bajo este esquema el bus permite la transferencia de una palabra del tamaño de un registro en forma autónoma, un byte siguiendo el ejemplo, en cada dirección. Una vez los datos son leídos de los registros de desplazamiento, se puede iniciar otra transferencia.

El bus puede ser configurado para muestrear en el primer o segundo flanco del reloj (Fase), comenzar con una señal del reloj con un valor alto o bajo (Polaridad) y/o para enviar primero el bit mas o menos significativo (MSB o LSB respectivamente).

En el contexto del satélite, el bus SPI que interconecta el procesador principal con el coprocesador cuenta con únicamente tres líneas de interconexión, SCLK, MOSI y MISO. Por simplicidad de la solución, se establece que el procesador principal funcione en todo momento como Maestro, y el coprocesador como esclavo. La línea SS se vuelve innecesaria al no existir otros esclavos que compartan el bus.

El bus del maestro (MSP430) posee dos registros de desplazamiento de 8 bits y puede ser configurado para realizar transferencias de 7 u 8 bits. Un registro de desplazamiento es utilizado para el envío de datos (TX) mientras que el otro es reservado para la recepción (RX). Además cuenta con dos buffers de 8 bit para la transmisión (TXBUF) y recepción (RXBUF) respectivamente. Una transferencia es iniciada cuando el buffer de transmisión es escrito. Entonces cuando el registro TX se encuentra vacío los datos son copiados del TXBUF al registro TX iniciando la transmisión a través de la línea MOSI hacia el esclavo. Al mismo tiempo los datos provenientes de la línea MISO son desplazados al registro RX. Cuando el último bit llega, los datos son copiados al RXBUF finalizando la transferencia. Ver Figura 6.3

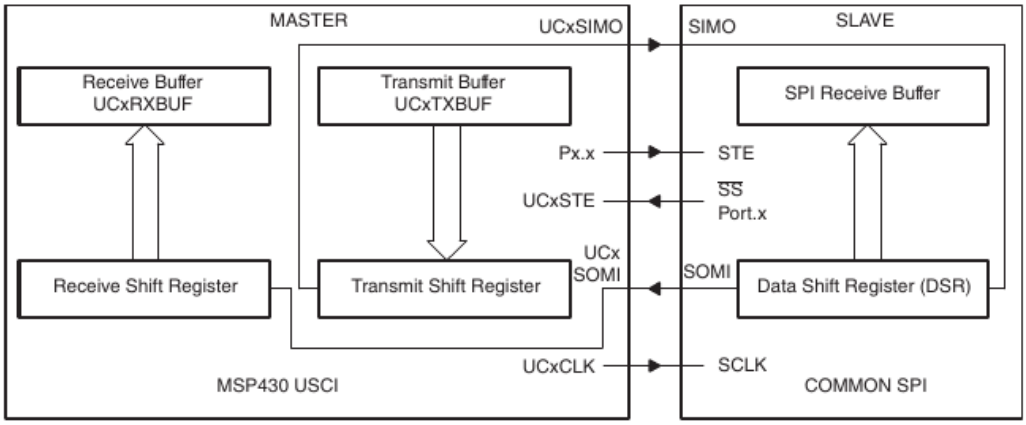

Figura 6.3. Estructura del bus SPI del MSP430. Autor de la imagen: Texas Instruments

El bus del esclavo (ARM) cuenta con un único registro de desplazamiento capaz de realizar transferencias de entre 4 y 16 bits y dos colas FIFO para la recepción (RXBUF) y transmisión (TXBUF) de 32 bit cada una. Ambas colas son accedidas a través de un registro de acceso de 16 bits (SPIx_DR). Un acceso de lectura en el registro SPIx_DR retorna el valor mas viejo en la cola RXBUF, un acceso de escritura en el registro SPIx_DR transfiere los datos escritos al final de la cola TXBUF. Al iniciarse una transmisión, la entrada mas vieja de TXBUF es copiada al registro de desplazamiento y es transmitida al mismo tiempo que se reciben los datos con el esquema antes explicado. Al completarse el desplazamiento los datos son copiados al final de la RXBUF. Ver Figura 6.4

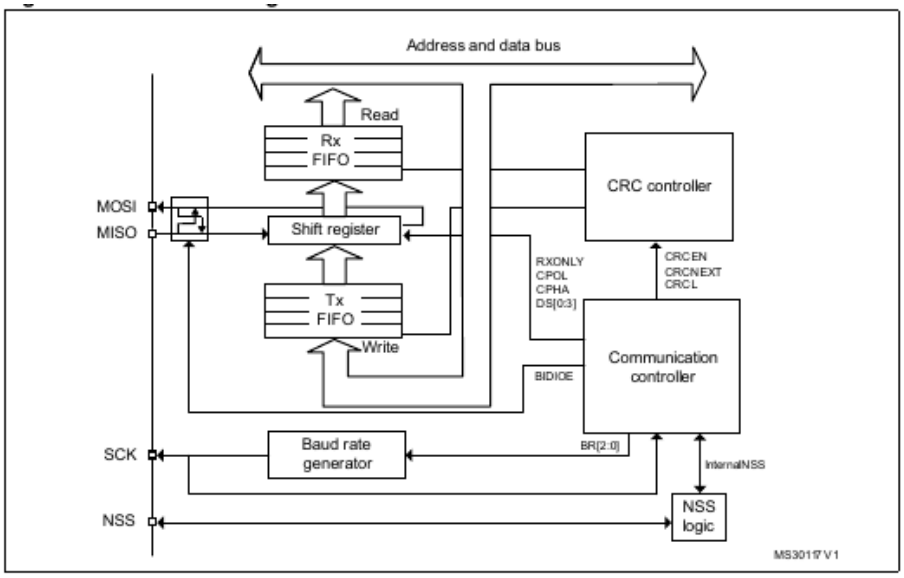

Figura 6.4. Estructura del bus SPI del ARM. Autor de la imagen: STMicroelectronics

Ambos periféricos fueron configurados a fin de lograr que sus tamaños de transferencia, fase, polaridad y dirección coincidieran. La tabla 6.3 muestra la configuración usada.

Atributo	**Valor**
Frecuencia del reloj	1Mhz
Fase	2do Flanco
Polaridad	Alta
Tamaño de transferencia	8 bits
Dirección del registro	MSB

Tabla 6.3. Configuración del bus SPI

Objetivos del protocolo

El objetivo final del protocolo es facilitar el envío de mensajes a través del bus SPI entre el procesador principal y el coprocesador. Dada la función del coprocesador como unidad de computo para los algoritmos de determinación y control, es necesario que la transmisión de mensajes sea confiable. Un dato corrupto en uno de los mensajes puede llevar a la desestabilización del satélite, por lo que tal evento debe ser detectado a fin de poder actuar en consecuencia.

Con este propósito, teniendo en consideración la configuración de hardware y las características de funcionamiento del bus, se tienen las siguientes restricciones:

- El procesador principal funciona como maestro y el coprocesador como esclavo.
- Todas las transferencias en ambas direcciones deben ser iniciadas por el maestro, el esclavo no tiene control sobre el bus.
- Cada transferencia implica el envío de un byte en cada dirección.
- No existe confirmación del bus de que el byte transferido haya sido recibido por la contra-parte.
- El canal no es confiable, por lo que la integridad de los datos no esta asegurada.

A partir de estas restricciones y a fin de cumplir con el objetivo, se establecen los siguientes problemas a resolver:

- **Comunicación bidireccional** Permitir el envío de mensajes en ambas direcciones, ya sea por iniciativa del maestro o del esclavo, teniendo en cuenta que solo el maestro puede iniciar las transferencias.
- **Envío de mensajes** Posibilitar el envío de una cadena de bytes en una dirección utilizando la primitiva de intercambio de bytes del bus.
- **Detección de Errores** Proveer la capacidad de determinar si el mensaje fue transmitido correctamente, debiendo detectar errores tanto del lado del emisor como del receptor.

Primitivas

El protocolo provee dos primitivas simples, una para el envío y otra para la recepción de mensajes. Al tratarse de un protocolo Maestro/Esclavo, la semántica de cada primitiva difiere entre uno y otro. A fin de simplificar el análisis, se considera que existen cuatro primitivas en total:

1. Operación de Envío desde el maestro: **Master Send**
2. Operación de Recepción desde el maestro: **Master Receive**
3. Operación de Envío desde el esclavo: **Slave Send**
4. Operación de Recepción desde el esclavo: **Slave Receive**

Las operaciones funcionan de a pares, un Master Send siempre debe corresponderse con un Slave Receive y un Master Receive con un Slave Send. En ambas direcciones, el esclavo debe invocar primero su operación respectiva. Por ejemplo, para realizar un Master Send en forma exitosa, previamente el esclavo debe haber invocado al Slave Receive. Análogamente, el Master Receive tiene como precondición que previamente se haya llamado un Slave Send. Esta dependencia del accionar del esclavo puede ser vista como una arquitectura Cliente/Servidor. El maestro oficia de cliente mientras que el esclavo como servidor. El servidor debe configurarse para escuchar peticiones antes de poder aceptar mensajes del cliente. Así, las operaciones del lado del esclavo realizan esta configuración y se bloquean, llevando el procesador a modo de bajo consumo, a la espera de una transmisión proveniente del maestro. Por su parte, las operaciones del maestro asumiendo que su contraparte ya fue invocada comienzan la transmisión de datos de inmediato. Si esta asunción fue errónea o no, eso es manejado por el control de errores del protocolo.

Comunicación bidireccional

El inicio de una comunicación con dirección maestro-esclavo se realiza de la siguiente manera: La operación Slave Receive tras realizar la configuración necesaria del bus a fin de aceptar comunicaciones del maestro, se bloquea y lleva el procesador a modo bajo consumo. Cuando el maestro invoca el Master Send, la comunicación inicia mediante el envío de un byte codificando la operación. Al recibir este byte el esclavo se despierta y corrobora que el byte recibido provenga de un Master Send. De ser así, el esclavo debe proceder a comunicar al maestro que se encuentra listo para recibir los datos y para ello escribe en el bus un byte codificando esta información.

Durante la transferencia del primer byte en dirección maestro-esclavo, otro byte es transferido en dirección contraria. Sin embargo, este último puede ser ignorado debido a que el esclavo no escribió datos en el bus previamente. El maestro en segunda instancia, procede a enviar un byte arbitrario con el objetivo de recibir la respuesta del esclavo. Tras esta transferencia, y en caso que el maestro reciba y corrobore la respuesta del esclavo, ambos se encuentran sincronizados para proceder con el envío de datos (Ver **Envío de Mensajes**). Este intercambio inicial se corresponde con la etapa de establecimiento de la conexión o Handshake del protocolo.

Cuando la comunicación es en dirección inversa (esclavo-maestro) el mismo esquema de handshake puede ser aplicado. La operación Slave Send se comporta en forma análoga al Slave Receive en esta primera etapa, bloqueándose a la espera de datos del maestro. Cuando el maestro ejecuta un Master Receive, al igual que el Master Send, envía un byte codificando la operación y un siguiente byte para poder obtener la respuesta del esclavo. El esclavo a su vez, al recibir y corroborar el primer byte, escribe la respuesta confirmando que se encuentra listo para enviar los datos.

Volviendo a la analogía del Cliente/Servidor, el cliente puede tanto enviar datos como recibir datos del servidor simplemente indicando que operación desea realizar. Sin embargo, este esquema obliga a que si el servidor tiene datos para enviar al cliente, esta transferencia deba posponerse hasta que el cliente solicite los datos. Se necesita que el cliente este periódicamente consultando si existen nuevos datos al servidor o que bien el servidor pueda señalizar por algún medio al cliente.

La solución implementada hace uso de la segunda opción mediante la utilización de un canal de comunicación extra. El procesador principal posee una línea de conexión con el coprocesador denominada SS (slave send) por conveniencia. Esta línea es utilizada por el coprocesador para señalizar al procesador principal que una transmisión pendiente en dirección esclavo-maestro se encuentra lista.

Para el protocolo la existencia de esta señalización es indistinta, pudiendo utilizarse igualmente el esquema de consulta periódica al esclavo. Sin embargo, en la implementación del mismo, la señalización es realizada dentro de la operación Slave Send (dando origen al nombre) con el fin de simplificar la solución. Por motivos de construcción del hardware, la señal SS del lado del maestro no puede ser manejada mediante interrupciones, por lo cual el maestro debe consultar la señal periódicamente.

Envío de mensajes

Un byte no puede ser enviado por SPI sin recibir otro simultáneamente, por lo que sin importar quien sea el emisor, el envío de mensajes se realiza mediante un intercambio de datos bidireccional. La transferencia por su parte solo se inicia por acción del maestro, cuando el esclavo escribe un byte en el bus, éste no es transmitido hasta que el maestro comience la transmisión.

La transferencia de una cadena de bytes requiere únicamente que se envíen secuencialmente los bytes del mensaje. Luego de la etapa de handshake, la transferencia del maestro al esclavo comienza por la escritura por parte del maestro, del primer byte que se desea enviar en el bus, aguardando a que se efectúe la transmisión que se inicia automáticamente en consecuencia. Cuando el esclavo detecta la llegada de dicho byte, lo lee y almacena en un buffer destinado a almacenar el mensaje completo. Finalizada la transmisión, el maestro prosigue con el siguiente byte en forma análoga, almacenándolo el esclavo en la siguiente posición del buffer. El proceso se repite hasta que se haya enviado la totalidad del mensaje.

En el envío en sentido inverso en cambio, el esclavo escribe el primer byte del mensaje en el bus y aguarda a que el maestro escriba a su vez un byte arbitrario que inicie el intercambio. Habiendo pasado por el

handshake, el maestro sabe que debe proceder a escribir dicho byte sin señalización alguna. Al hacerlo, recibe lo transmitido por el esclavo para almacenarlo en su propio buffer. El esclavo entonces puede proceder con el siguiente byte y volver a aguardar por la acción del maestro. Esto se repite hasta que la totalidad del mensaje haya sido transmitida.

Este esquema requiere que ambos partícipes estén de acuerdo en el largo del mensaje a ser transmitido. El receptor necesita asegurar que el buffer de recepción es lo suficientemente grande para contener el mensaje entrante, y conocer además cuando el mensaje esta completo para dejar de esperar nuevos bytes del emisor. El emisor debe por su parte verificar que el mensaje a enviar no es demasiado largo y evitar un desbordamiento del lado del receptor. La cadena a transmitir se extiende entonces con este propósito. Un cabezal de ocho bits conteniendo el largo se anexa al principio de un mensaje y un byte de finalización que marca el final del mensaje se agrega al final. Esto permite que el receptor pueda al verificar el largo, detectar un posible desbordamiento y a su vez usarlo para determinar el fin de la transmisión. Una vez recibida la cantidad de bytes indicadas por el largo, basta corroborar que el siguiente byte es el byte de finalización para que el receptor determine que el mensaje fue transmitido completamente y que el largo recibido no es erróneo. Observar que la codificación del largo en ocho bits implica que el largo de mansaje máximo que se puede transmitir es de 255.

El envío del mensaje se enmarca dentro de la etapa de transmisión de datos (Data Transmission).

Detección de Errores

¿Qué ocurre cuándo hay corrupción o perdida de datos? El protocolo debe asegurar que el mensaje recibido es correcto, o dicho en otros términos, que es idéntico al mensaje enviado por la contraparte. Para ello es necesario detectar cuando se produce una corrupción, por lo que requiere de algún mecanismo de verificación de integridad de los bytes transmitidos.

El enfoque que sigue el protocolo para lograr este objetivo hace uso de la transferencia bidireccional de SPI. Observar que al transmitir una cadena de bytes, una cadena de igual largo es recibida. El maestro para recibir un mensaje del esclavo debe enviar un byte arbitrario que de inicio cada transmisión. Ahora bien, si en lugar de un byte cualquiera lo que se envía es el último byte recibido, el esclavo es capaz de saber que fue lo que recibió el maestro en la transmisión anterior y compararlo con lo que efectivamente envió. Si hay coincidencia, las posibilidades de que haya habido corrupción o perdida de ese byte son despreciables, en caso contrario significa que hubo un error. El mismo principio es aplicable cuando el emisor es el maestro, haciendo que el esclavo escriba el último byte recibido en el bus para que sea transmitido en el próximo intercambio.

De esta manera el emisor puede corroborar la integridad del mensaje enviado, sin embargo hace falta transmitir al receptor si hubo algún error a fin de que ambas partes sean consientes del mismo. Las verificaciones son hechas del lado del emisor únicamente.

Al enviar el byte de fin de mensaje, el emisor recibe la confirmación del último byte enviado. Ya con todo el mensaje confirmado, en base a si hubo alguna falla detectada o no a lo largo de la comunicación, se determina si el mensaje fue correctamente transmitido o no. El resultado es enviado al receptor codificando con un valor específico el éxito de la transferencia, cualquier otro valor se traduce en que hubo una falla. En la transmisión complementaria el emisor recibe la confirmación de la recepción del byte de fin de mensaje.

¿Que ocurre si al enviar el resultado el mismo se corrompe? Se tienen dos casos: Si codificaba el caso exitoso, al cambiar su valor va a ser interpretado como falla y el receptor va a culminar la comunicación con un error, descartando el mensaje que en efecto era correcto. Si codificaba el caso no exitoso, y solo si la corrupción hizo mutar el valor para que se convierta al valor que codifica éxito, lo cual es poco probable, el receptor va a asumir en un principio que el mensaje es correcto, cuando en efecto no lo es. Sin embargo, en un siguiente paso se mitiga esta posibilidad.

Del lado del emisor, se corrobora la confirmación del byte de fin de mensaje. Si esta confirmación falla, el emisor naturalmente va a determinar que hubo un error e informa al receptor con una última transmisión codificando si hubo error o no. De la misma forma, si se dio el caso que en el paso anterior se envió un resultado no exitoso, en este paso también se envía sin importar que ocurrió con el byte de fin de mensaje, un byte codificando el error. Esta última transmisión tiene como consecuencia que en presencia

de mutación del caso no exitoso al exitoso, el receptor de todas formas va a recibir en este paso un byte de error y finalmente descartar el mensaje corrupto. Por otra parte, el receptor en esta etapa envía al emisor la confirmación del byte de resultado, logrando que el emisor también pueda finalizar con error en caso de corrupción de dicho byte. Aún así, no es posible evitar que el emisor y el receptor terminen con diferente visión de si la comunicación fue exitosa o no, puesto que estos últimos bytes enviados también pueden sufrir de corrupción.

Este intercambio final de datos codificando el resultado de la transmisión forman la etapa de finalización (Finalization).

Etapas del protocolo

Las tres etapas del protocolo, Handshake, Data Transmission y Finalization derivadas de las secciones anteriores pueden ser vistas en los diagramas de secuencia de la Figura 6.5

Figura 6.5. Diagramas de secuencia del protocolo **spimp**

El envío de datos desde el maestro (Master Send / Slave Receive) sigue la siguiente secuencia:

- Handshake
  1. El maestro envía el byte MASTER_SEND_START indicando el inicio de la transferencia. El esclavo envía un byte indeterminado.
  2. El maestro envía el byte SET_LENGTH para indicar que se procederá a enviar el largo del mensaje. El esclavo envía el byte OK confirmando la aceptación de inicio de transferencia.
- Data Transmission
  1. El maestro envía el largo del mensaje a transmitir. El esclavo confirma la recepción del byte SET_LENGTH con el byte SET_LENGTH_ACK.
  2. El maestro envía el primer byte del mensaje. El esclavo envía la confirmación del largo.
  3. El maestro envía el siguiente byte del mensaje. El esclavo envía la confirmación del byte anterior. Este paso se repite para todos los restantes bytes del mensaje.
  4. El maestro envía el byte MASTER_SEND_END indicando el fin del mensaje. El esclavo envía la confirmación del último byte.
- Finalization

1. El maestro envía el byte MASTER_SEND_CONFIRMATION_OK indicando que la totalidad del mensaje fue transmitida correctamente. El esclavo envía el byte MASTER_SEND_END_ACK confirmando el byte de fin de mensaje.

2. El maestro envía el byte MASTER_SEND_IS_DONE indicando la finalización con éxito de la transferencia. El esclavo envía el byte MASTER_SEND_DONE confirmando la finalización con éxito del lado del receptor.

Por su parte el envío de datos desde el esclavo (Master Receive / Slave Send) sigue la siguiente secuencia:

- Handshake

  1. El maestro envía el byte MASTER_RECEIVE_START indicando el inicio de la transferencia. El esclavo envía un byte indeterminado.

  2. El maestro envía el byte GET_LENGTH para indicar que se procederá a esperar el largo del mensaje. El esclavo envía el byte MASTER_RECEIVE_ACK confirmando que se encuentra listo para enviar datos.

- Data Transmission

  1. El maestro envía el byte GARBAGE para poder obtener el largo del mensaje a transmitir. El esclavo envía el largo del mensaje a transmitir.

  2. El maestro confirma el largo del mensaje. El esclavo envía el primer byte.

  3. El maestro envía la confirmación del último byte recibido. El esclavo envía el siguiente byte del mensaje. Este paso se repite para todos los restantes bytes del mensaje.

  4. El maestro envía la confirmación del último byte. El esclavo envía el byte MASTER_RECEIVE_END indicando el fin del mensaje.

- Finalization

  1. El maestro envía el byte MASTER_RECEIVE_END_ACK confirmando el byte de fin de mensaje. El esclavo envía el byte MASTER_RECEIVE_CONFIRMATION_OK indicando que la totalidad del mensaje fue transmitida correctamente.

  2. El maestro envía el byte MASTER_RECEIVE_DONE confirmando la finalización con éxito del lado del receptor. El esclavo envía el byte MASTER_SEND_IS_DONE indicando la finalización con éxito del lado del emisor.

6.3.5. spirpc

Como se mencionó anteriormente **spirpc** es el protocolo desarrollado para ejecutar cálculos en el coprocesador. Consiste en una capa por encima de **spimp** que permite al procesador principal ejecutar un algoritmo en el coprocesador y obtener la respuesta. En esta sección se elaboran los detalles de su diseño e implementación.

Las primitivas de **spirpc** permiten el envío y la recepción de cada uno de los mensajes que intercambian el procesador principal y el coprocesador. Estas operaciones invocan las primitivas de **spimp** internamente. El resto de la aplicación nunca accede a las primitivas de **spimp** en forma directa sino a través del módulo **spirpc**.

Mensajes

Un mensaje spirpc consiste de un cabezal de largo fijo y un segmento de datos de largo variable. El cabezal se compone de un código de mensaje de 32 bits seguido de 4 bytes de relleno para alinear el cabezal a 8 bytes.

El protocolo spirpc consta de varios tipos de mensajes para distintos propósitos como sincronización, actualización de parámetros, etc. Los mensajes del protocolo son los siguientes:

- SPI_RPC_CMD_ECHO_REQUEST: Enviado del maestro al esclavo para solicitar que el mensaje sea repetido hacia el maestro. El segmento de datos contiene una cadena de hasta 240 caracteres terminada en un caracter NUL. Se incluyó por propósitos de verificación del protocolo.

- SPI_RPC_CMD_ECHO_RESPONSE: Enviado por el esclavo al maestro como respuesta a un mensaje SPI_RPC_CMD_ECHO_REQUEST. El segmento de datos contiene una cadena de hasta 240 caracteres terminada en un caracter NUL que no tiene por qué ser igual a la cadena enviada por el procesador principal.

- SPI_RPC_CMD_TEST: Enviado del maestro al esclavo con datos estructurados. El segmento de datos contiene una estructura con campos de distintos tipos y tamaños. Se incluyó por propósitos de verificación del protocolo.

- SPI_RPC_CMD_TEST_RESPONSE: Enviado por el esclavo al maestro como respuesta a un mensaje SPI_RPC_CMD_TEST. El segmento de datos contiene el mismo tipo de estructura que el mensaje SPI_RPC_CMD_TEST.

- SPI_RPC_CMD_PING: Enviado por el procesador principal al coprocesador al iniciar para determinar si el mismo está listo para recibir mensajes. Cuando ADCS se enciende el procesador principal y el coprocesador reciben alimentación al mismo tiempo, pero se necesita un mecanismo de sincronización que permita saber cuándo el coprocesador terminó su inicialización y está listo para procesar mensajes. Cuando el procesador principal inicia envía mensajes SPI_RPC_CMD_PING al coprocesador, hasta que éste responde con un mensaje SPI_RPC_CMD_PONG. Una vez se obtiene esta respuesta el procesador principal puede comenzar a enviar mensajes de control al coprocesador.

- SPI_RPC_CMD_PONG: Enviado por el coprocesador al procesador principal como respuesta a un mensaje SPI_RPC_CMD_PING. El campo de datos queda vacío.

- SPI_RPC_CMD_SET_PARAMS: Enviado por el procesador principal al coprocesador para notificar a este último de una actualización de parámetros desde la Tierra. El segmento de datos contiene una estructura de tipo adcs_parameters (ver sección Requerimientos por más información sobre los parámetros).

- SPI_RPC_CMD_PARAMS_RECEIVED: Enviado del coprocesador al procesador principal para indicar que la actualización de parámetros fue recibida correctamente. El campo de datos contiene un entero de 32 bits cuyo bit menos significativo es 1 si y sólo si los parámetros se recibieron y actualizaron exiosamente en el coprocesador.

- SPI_RPC_CMD_CONTROL: Enviado del procesador principal al coprocesador para solicitar la ejecución de una iteración del algoritmo de control. El campo de datos es una estructura att_det_ctrl_input_t conteniendo la entrada para la iteración actual. Observar que este es el mensaje principal que el procesador principal envía el coprocesador cuando el lazo de control está activo.

- SPI_RPC_CMD_CONTROL_RESPONSE: Enviado desde el coprocesador al procesador principal conteniendo la respuesta al mensaje SPI_RPC_CMD_CONTROL. El campo de datos contiene una estructura att_det_ctrl_output_t con la salida del algoritmo de control.

- SPI_RPC_CMD_SET_TLE: Enviado del procesador principal al coprocesador para notificar a este de una actualización de TLE desde la Tierra. El campo de datos es una estructura tle_t.

- SPI_RPC_CMD_TLE_RESPONSE: Enviado desde el coprocesador al procesador principal como respuesta al mensaje SPI_RPC_CMD_SET_TLE. El campo de datos contiene un entero de 32 bits cuyo bit menos significativo es 1 si y sólo si el TLE se recibió y actualizó exiosamente en el coprocesador.

6.4. Verificación

En esta sección se presenta el proceso de verificación de software realizado para ADCS. Se describen pruebas realizadas, resultados obtenidos y dificultades encontradas.

6.4.1. Hardware utilizado

A fin de realizar la verificación del software implementado, se requirió de la utilización de kits de desarrollo para los dos micros utilizados en ADCS:

- MSP430: msp-fet430u5x100 [23]
- ARM: stm32f3 discovery [32]

Estos kits brindan una interfaz de depuración mediante un puerto USB, lo cual permite subir código a los microcontroladores y depurarlos facilmente.

En étapas avanzadas, varias pruebas fueron realizadas también sobre la placa de vuelo de ADCS, es decir, el hardware final que posteriormente fuera enviado al espacio. Interfaces de depuración USB fueron conectadas al satélite con este motivo.

6.4.2. Pruebas unitarias de modelos

Como se mencionó anteriormente los algoritmos de control de actitud implementados en el ADCS hacen uso de modelos para calcular la dirección del sol, la posición orbital (modelo SGP4), y el campo magnético terrestre (modelo IGRF). Se desarrolló una prueba unitaria para verificar el comportamiento de estos modelos, así como los cambios de coordenadas que se realizan en los cálculos de actitud (ver sección 6.3.3).

Los datos de referencia para esta prueba se basan en el sofware *Systems Tool Kit* (STK) ([30]), una herramienta de modelado y simulación para sistemas complejos que permite realizar y visualizar cálculos de mecánica orbital. Es usado ampliamente tanto para desarrollo de sistemas de control como para control de misión. Se usó como referencia el satélite UWE-2, un Cubesat de la Universidad de Würzburg, Alemania puesto en órbita en 2009. Este satélite fue usado como referencia también en el proyecto antecedente del grupo Senshor.

El siguiente es el TLE usado como referencia, correspondiente al 12 de Abril de 2011 a las 7:22 GMT:

```
1 35934U 09051D 11102.30706031 .00003318 00000-0 81574-3 0 7771
2 35934 98.3238 202.8887 0007042 20.2291 339.8724 14.52927588 82190
```

Partiendo de dicha fecha se tomaron 50 valores de tiempo avanzando en pasos de 5 horas. Se construyó en STK un reporte que para cada uno de estos instantes calcula los siguientes parámetros:

- Posición del satélite en sistema ECI.
- Posición del satélite en sistema ECEF.
- Velocidad del satélite en sistema ECI.
- Posición del satélite en latitud, longitud y altura.
- Vector de campo magnético en sistema ECEF.
- Vector de campo magnético en Sistema Orbital.
- Vector de sol en sistema ECI.
- Vector de sol en Sistema Orbital.

Estas variables fueron calculadas en el programa de prueba usando los mismos valores de tiempo y el mismo TLE. Los resultados completos del programa y de STK se recopilan en la planilla "Uwe-2_35934 Reporte de sol y campo magnético" que se entrega como anexo. El programa de prueba se compiló y ejecutó en PC sobre Linux. También se compiló y ejecutó en el kit de desarrollo de ARM con el único propósito de probar el proceso de compilación e instalación en el hardware.

La Tabla 6.4 muestra los resultados de posición ECI en el instante inicial en STK y el modelo. Se observa que el error es despreciable. El error es mayor a medida que la propagación se aleja de la hora del TLE. Las Tablas 6.5 y 6.6 muestran los valores luego de cinco horas de propagación y luego de seis días respectivamente. Los resultados concuerdan con los obtenidos por el grupo senshor en la prueba análoga.

Variable	STK	Modelo	Error
Posición x (km)	-6533.75341167664	-6533.75332014849	0.000091528149824
Posición y (km)	-2759.34071918042	-2759.3409366001	0.000217419680212
Posición z (km)	-5.6232243073803	-5.62483769236056	0.00161338498026

Tabla 6.4. Posición ECI en instante inicial

Variable	STK	Modelo	Error
Posición x (km)	-6505.17123797533	-6505.17090399082	0.00033398450978
Posición y (km)	-2598.8832431539	-2598.88395675095	0.00071359704998
Posición z (km)	1100.74463714007	1100.74491528295	0.000278142880006

Tabla 6.5. Posición ECI luego de 5 horas

Variable	STK	Modelo	Error
Posición x (km)	1304.11045579084	1241.43826788956	62.67218790128
Posición y (km)	-445.289581948115	-481.594132837408	36.304550889293
Posición z (km)	-6967.15278575275	-6976.23801155716	9.08522580440967

Tabla 6.6. Posición ECI luego de 6 días

Los resultados de la conversión de la posición a sistema ECEF en el instante inicial se pueden ver en la Tabla 6.7. El error es despreciable.

Variable	STK	Modelo	Error
Posición x (km)	-2167.84267196663	-2167.95249293735	0.109820970720193
Posición y (km)	-6753.09950192182	-6753.06425567795	0.035246243869551
Posición z (km)	-5.63351744283839	-5.62483769236056	0.00867975047783

Tabla 6.7. Posición ECEF en instante inicial

La tabla 6.8 muestra la conversión de posición ECEF a latitud y longitud en el instante inicial. El error está muy por debajo de 1°.

Variable	STK	Modelo	Error
Latitud (grados)	-0.045785058425097	-0.045714515962761	0.000070542462336
Longitud (grados)	-107.797381934076	-107.798313676126	0.00093174205

Tabla 6.8. Latitud y longitud en instante inicial

En cuanto a los valores de sol en ECI el error se calcula sobre el vector normalizado, ya que lo que interesa es la dirección y no el módulo. La Tabla 6.9 muestra los resultados en el instante inicial. Se observa que el error es menor a 1 % en las tres coordenadas.

Variable	STK	Modelo	Error (%)
Sol x	0.927889599859492	0.929027755785823	0.122660705163985
Sol y	0.342088006808762	0.345191677751776	0.90727265535183
Sol z	0.148312798066111	0.149648412054852	0.90053859556047

Tabla 6.9. Vector de Sol ECI en instante inicial

La Tabla 6.10 muestra los resultados de campo magnético en sistema ECEF en el instante inicial.

Variable	STK	Modelo	Error
Campo magnético x (nT)	4641.77295707238	4641.82141951971	0.048462447329257

Continúa en la siguiente página

Variable	STK	Modelo	Error
Campo magnético y (nT)	4370.48299909054	4370.36326553326	0.119733557279687
Campo magnético z (nT)	21354.9938637184	21354.9480892288	0.045774489597534

Tabla 6.10. Campo magnético ECEF en instante inicial

Los resultados de campo magnético en el proyecto antecedente de Determinación de Actitud mostraban una gran diferencia con respecto a STK (del orden de cientos de nT). Al inspeccionar este programa se detectó un error en la conversión de la latitud y longitud de grados a radianes. Se aplicaba la función `floor()` sobre un argumento que podía ser negativo cuando en ese caso correspondía hacer un truncamiento (se trataba de un bug en la utilización de la biblioteca de Vallado). Además el programa realizaba una conversión de grados decimales a grados, minutos y segundos que luego era revertida internamente al algoritmo IGRF, por lo que era innecesaria. Se eliminó esta conversión y el uso incorrecto de la primitiva `floor()`, luego de lo cual el error en el campo magnético con respecto a STK se redujo considerablemente a los valores mostrados en la Tabla 6.10, del orden de décimos de nT.

6.4.3. Pruebas de integración de algoritmos

Como se ha mencionado, el desarrollo de los algoritmos de control fue llevado a cabo por el responsable de ADCS en Matlab. El programa fue traducido a una biblioteca C gracias a una funcionalidad que provee Matlab para ese fin, siendo integrada en la aplicación del microcontrolador ARM.

Se desarrolló un programa de prueba `det_ctrl` con el fin de verificar la integración de los modelos, el algoritmo Matlab y las conversiones de datos y coordenadas. Esta prueba se ejecutó en PC sobre Linux.

El conjunto de datos de prueba consiste en una secuencia de 1500 estructuras `att_det_ctrl_input_t` con valores de tiempo consecutivos separados por 1 segundo. Los valores de este conjunto de datos fueron generados por el responsable de ADCS en Matlab, junto con los resultados esperados del algoritmo. Las salidas esperadas se tradujeron a una secuencia de estructuras `att_det_ctrl_output_t`. Para cada estructura de entrada el programa de prueba aplica los modelos y el el algoritmo Matlab realizando los cambios de coordenadas asociados. Los resultados se comparan con la salida esperada para la entrada actual, y la prueba falla si la diferencia supera cierto umbral. Cabe notar que en esta comparación los umbrales no son los mismos para todas las variables debido a diferencias de escala. Algunos valores pequeños como las corrientes se comparan con una tolerancia baja de 0.0001 mientras que para otros, como la posición en km, la tolerancia es de 0.25.

Los resultados coinciden con los esperados sobre todo el conjunto de prueba.

6.4.4. Pruebas de integración del coprocesador, caja blanca

La siguiente prueba de integración trabaja sobre el software del coprocesador en su totalidad. Los módulos integrados en esta prueba son los modelos, el algoritmo matlab, los procedimientos de conversión de datos, el lado esclavo de los protocolos spimp y spirpc y el intérprete de comandos del coprocesador. La prueba se ejecutó en PC. Se trata de una prueba de caja blanca, cuyos casos buscan lograr el mayor cubrimiento sobre el software del coprocesador como sea posible.

El documento “Pruebas de integración coprocesador, caja blanca” que se entrega como anexo contiene la lista de casos de prueba definidos. La mayoría de los casos consiste en alimentar el intérprete de comandos del coprocesador con un mensaje SPI dado y verificar que la respuesta enviada por SPI coincida con la esperada. El software del coprocesador interpreta el mensaje recibido, extrae datos numéricos conteniendo datos de entrada, invoca los modelos, invoca el algoritmo de control y construye un mensaje spimp como respuesta. La respuesta se compara con la esperada a nivel binario y la prueba falla si no coinciden.

Los casos de prueba definidos están pensados para verificar casos de error en el protocolo spimp tales como bytes incorrectos y fallos por falta de espacio en buffers. Los flujos típicos del protocolo alcanzan un cubrimiento de código muy bajo, por lo que la mayor parte de los casos de prueba corresponden a distintos casos de borde que simulan errores de comunicación.

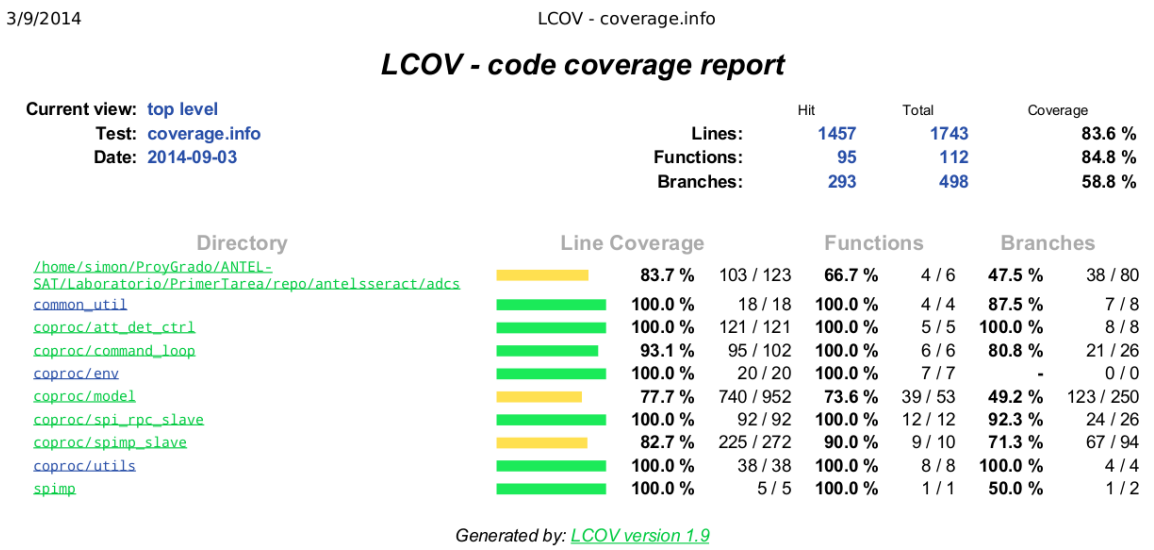

Figura 6.6. Reporte de cubrimiento para pruebas de caja blanca

Uno de los propósitos de esta prueba era obtener un reporte de cubrimiento de código. A estos efectos se evaluó usar el depurador del IAR para ARM, el cual permite generar reportes de cubrimiento. Sin embargo se encontró que esta funcionalidad solo está disponible al ejecutar en el modo simulador del depurador, no al ejecutar en el microcontrolador. Se resolvió entonces compilar y ejecutar el software del coprocesador en PC y generar el reporte de cubrimiento usando la herramienta GCOV ([14]), que se integra fluidamente con el compilador gcc. La comunicación SPI se simula por medio de un stub de la capa de abstracción de hardware del coprocesador, en la cual la entrada del programa en modo de bajo consumo desencadena la ejecución de la rutina de interrupción del módulo SPI leyendo datos de un buffer fijo. Esto permite ejecutar el software del coprocesador en PC simulando la llegada de mensajes SPI predefinidos.

Todos los casos de prueba son exitosos. La Figura 6.6 resume los resultados de cubrimiento. El reporte se generó por medio de la herramienta LCOV [21], que usada en combinación con GCOV permite visualizar los resultados de cubrimiento en forma de un conjunto de páginas HTML vinculadas. Este reporte se entrega como anexo en el directorio `cubrimiento`. Como se aprecia en la figura, el cubrimiento de líneas es de 83.6 % y el cubrimiento de bifurcaciones es de 58.8 %. Los módulos que se toman en cuenta en el reporte son todos los que forman parte del coprocesador excepto los stubs de HAL y los casos de prueba en sí. Dado que los errores pueden ocurrir en cualquier estado del protocolo hay una gran cantidad de flujos que pueden generar un error de comunicación. Tenerlos en cuenta a todos en los casos de prueba es muy complejo, por lo que el cubrimiento no es total.

6.4.5. Pruebas unitarias de protocolo spimp

Se realizó una prueba de transferencia de mensajes a través del protocolo spimp. El programa de prueba envía del MSP430 al ARM una estructura conteniendo una cadena de caracteres, valores de punto flotante, un entero y un caracter. El mensaje es recibido en el ARM y enviado de vuelta hacia el MSP430, donde se verifica que la respuesta sea igual al mensaje original. La prueba fue exitosa.

6.4.6. Pruebas unitarias de protocolo spirpc

Para la verificación unitaria del protocolo spirpc se desarrolló un programa de prueba que envía todos los distintos tipos de mensajes del protocolo spirpc del MSP430 al ARM y verifica que la respuesta sea la esperada. Para el mensaje `SPI_RPC_CMD_CONTROL` se usan datos generados por el responsable de ADCS, comparando la salida con el resultado esperado. Todas las pruebas son exitosas.

6.4.7. Pruebas de integración entre procesador principal y coprocesador

Esta prueba tiene como propósito verificar la integración de todo el software del coprocesador con el procesador principal. Para su ejecución se utilizaron ambos kits de desarrollo.

La Figura 6.7 muestra el hardware usado para la prueba. Sobre la izquierda se ven la placa de desarrollo y la interfaz de programación de MSP430, arriba y abajo respectivamente. Sobre la derecha se ve la placa de desarrollo de ARM que está conectada por USB a una PC. Ambas placas fueron conectadas a través de SPI mediante la interconección de las siguientes señales:

- Slave Send, usada en el coprocesador para señalizar al procesador principal de que el primero tiene datos para enviar. No confundir con la señal *Slave Select* de SPI, que no se usa en nuestro caso (ver sección "Comunicación bidireccional").
- Reset, usado para reinicar el coprocesador desde el procesador principal cuando el primero deja de responder.
- Señal `SCLK` de SPI
- Señal `MOSI` de SPI
- Señal `MISO` de SPI
- Señal `GND` (enlace a tierra)

El programa `Control_test` que se instaló en el MSP430 para esta prueba consiste en un bucle infinito que envía mensajes `SPI_RPC_CMD_CONTROL` al coprocesador por SPI, espera la respuesta y la compara con el resultado esperado. El conjunto de prueba usado es la misma simulación de 1500 segundos usada en la prueba de integración de los algoritmos. Se usa un LED en el MSP430 para indicar el resultado de la prueba. En cada iteración si todos los valores de la salida son suficientemente cercanos a los esperados se hace un toggle del LED, de otro modo se lo hace titilar durante unos segundos. Se hace una espera de 1 segundo entre iteraciones para poder apreciar el movimiento del LED. Cuando se completan las 1500 iteraciones el programa del MSP430 termina su ejecución invocando la primitiva `stop()` del microkernel. En este punto el LED deja de oscilar y la prueba termina. Todas las iteraciones fueron exitosas.

El programa usado en esta prueba del lado del MSP430 consta de tres aplicaciones. La primera hace de interfaz con el coprocesador, la segunda recorre el conjunto de prueba invocando el algoritmo y la tercera implementa el servicio de temporización usado para los timeouts de la máquina de estados. La cantidad de aplicaciones es significativamente menor que en las pruebas de EMS. Esto sumando al uso de LEDs hizo que los problemas ocurridos del lado del MSP430 fueran más fáciles de aislar y depurar.

Esta prueba incluye la sincronización entre los microcontroladores. Cuando el MSP430 inicia envía mensajes `SPI_RPC_CMD_PING` al coprocesador hasta que este responde con un mensaje `SPI_RPC_CMD_PONG`. De esta manera los microcontroladores pueden ser encendidos en cualquier orden y la prueba no comenzará hasta que el coprocesador esté listo para recibir comandos.

Cada 10 iteraciones el programa del MSP430 envía al ARM una actualización de TLE y parámetros, para verificar esta comunicación también.

Se inluyó en el software del coprocesador una bandera en tiempo de compilación que hace que el mismo no responda a ningún mensaje (la bandera está deshabilitada por defecto). Esto permite simular un bloqueo y verificar que el procesador principal reinicie el coprocesador cuando este deja de responder. Se verficó en forma exitosa esta funcionalidad por medio de un LED en el coprocesador que se hace titilar cuando el mismo inicia.

Esta prueba también se usó para realizar medidas de tiempo de ejecución del algoritmo de control y tiempo de comunicación entre micros. Se usó un GPIO en el MSP430 que se pone en valor alto cuando se envía el mensaje al coprocesador y se baja cuando se recibe la respuesta. Otro GPIO en el ARM se sube cuando comienza el cálculo y se baja cuando el mismo termina. Esto hace posible medir el tiemo de ejecución del algoritmo y el tiempo agregado por la comunicación SPI, observando las señales en un osciloscopio. La Figura 6.8 muestra la onda correspondiente al GPIO del MSP430 (tiempo total). Cada cuadrado correponde a 10 ms en el eje horizontal.

Esta prueba se realizó con los kits de desarrollo y también con la placa vuelo de ADCS. Los resultados numéricos fueron los esperados en ambos casos. Se probó configurar la frecuencia de reloj del bus SPI

Figura 6.7. Kits de desarrollo conectados por SPI

en 128 KHz y luego en 512 KHz con el propóstio de evaluar el impacto en el tiempo de comunicación. Los resultados para la placa de vuelo se muestran en la Tabla 6.11.

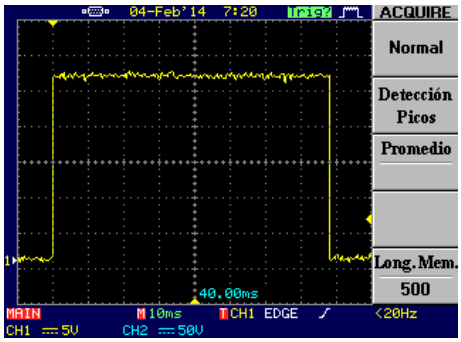

Figura 6.8. Visualización en osciloscopio del tiempo de ejecución del algoritmo de control, reloj de SPI en 512 KHz

Como se puede apreciar al aumento en la velocidad del bus no afecta en gran medida el tiempo total. De todas maneras se decidió configurar la frecuencia de reloj de SPI en 1024 KHz.

Reloj MSP430	**Reloj ARM**	**Reloj SPI**	**Algoritmo (ms)**	**Total (ms)**
8 MHz	60 MHz	512 KHz	62	78
8 MHz	60 MHz	1024 KHz	62	76

Tabla 6.11. Mediciones de tiempo en hardware de vuelo

Cabe notar que los errores detectados en el software del coprocesador se depuraron y corrijieron enteramente en PC por medio del programa `det_ctrl`. Una vez que se obtuvieron los resultados esperados en PC las pruebas con `Control_test` en hardware de vuelo fueron exitosas.

6.4.8. Prueba de integración de máquina de estados del procesador principal

Esta prueba se realizó con el propósito de verificar la máquina de estados del procesador principal. Se usan los kits de desarrollo de MSP430 y ARM conectados por SPI. En el MSP430 se ejecuta el programa `Control_test`. En el ARM se instala un programa que produce salidas fijas elegidas para causar transiciones de estado en el procesador principal. Usando el depurador del IAR conectado al MSP430 se inspecciona que las transiciones esperadas ocurran. Esta prueba se concluyó sin encontrar problemas.

En algunas de las ocasiones en las que se ejecutaron pruebas para los modelos y fallaron se encontró que los vectores en los datos de prueba estaban expresados en un sistema de referencia y el algoritmo los requería en otro.

Las primeras pruebas de integración de algoritmos realizadas usaban un conjunto de datos de prueba con valores de tiempo no contiguos. Se encontró que esto no permitía obtener los resultados esperados, ya que el algoritmo Matlab consta de un estado interno y la salida depende de este estado además de depender de la entrada. Se pasó entonces a usar un conjunto de datos con valores de tiempo contiguos.

Otro problema surgió por un error en la forma como se comparaban los resultados de actitud. Dado que esta se representa con un cuaternión, obtener un valor o su opuesto componente a componente es indistinto, ya que la actitud representada es la misma. El cuaternión obtenido por el software era el opuesto del indicado por STK causando que la prueba fallara. Se modifió la lógica de comparación de resultados para considerar un cuaternión y su opuesto como equivalentes.

Analogamente las variables roll, pitch y yaw se comparan teniendo en cuenta que los valores 180 y -180 son equivalentes.

Se encontró un bug en el módulo que convertía los datos de entrada para el algoritmo Matlab. Este espera las matrices representadas como arreglos con las entradas listadas por filas, pero se las estaba pasando por columnas.

6.5. Validación

El software final del ADCS fue elaborado en conjunto con el responsable de integración y el responsable de ADCS. Esta sección describe el trabajo de integración realizado para construir el software final de ADCS y la participación de nuestro equipo en el mismo.

Para el procesador principal el responsable de integración construyó la lógica de aplicación desde cero. La capa de abstracción de hardware fue implementada por el responsable de ADCS. Nuestro equipo proveyó los componentes asociados a la comunicación SPI, esto es, el lado del maestro de los protocolos spirpc y spimp, mas un módulo basado en el programa de prueba `Control_test` que encapsula la invocación de las primitivas de spirpc, desarrollado para las pruebas de integración entre microcontroladores. Los módulos que implementan los protocolos se usaron sin cambios. La versión final de las aplicaciones del procesador principal implementan una serie cambios de requerimientos, a saber:

- El mensaje de actualización de hora que se recibe en ADCS desde MCS contenía un bit indicando si la hora es suficientemente reciente para ser confiable. Esta bandera pasa a ser ignorada en ADCS, eliminándose la transición a estado `CRASH` que se hacía en la versión original de la máquina de estados cuando la hora estaba desactualizada.
- El mensaje de estado enviado desde ADCS a PAYLOAD pasa a contener las variables de roll, pitch y yaw como valores en punto flotante.
- El valor por defecto del parámetro `to_det` se cambia a 15000.

En lo que respecta al coprocesador se usó integramente el software desarrollado por nuestro equipo sin cambios.

6.6. Lecciones aprendidas

Esta sección resume las experiencias adquiridas con el trabajo realizado en el Módulo de Determinación y Control de Actitud.

La experiencia desarollando en ARM presentó diferencias con el trabajo en MSP430:

- STMicroelectronics provee una biblioteca de acceso a los periféricos del micro ARM (*Standard Peripheral Library*). Su uso facilita en gran medida el trabajo de configuración y uso de los periféricos, abstrayendo la típica configuración mediante accesos a registros en funciones de alto nivel. Esto permitió desarrollar el protocolo de comunicación sin tener que profundizar en la arquitectura,

evitando errores debido al acceso a registros equivocados que son muy difíciles de encontrar. Cabe notar que el uso de hardware en el microcontrolador ARM es mínimo, reducido a la interfaz SPI, GPIOs y LEDs. En esta experiencia el uso de la Standard Peripheral Library fue exitoso.

- El compilador gcc usado para en ARM fue efectivo a la hora de desarrollar una aplicación que hace uso intensivo de cálculos de punto flotante, permitiendo sacar provecho de la unidad de punto flotante del microcontrolador. El soporte para MSP430 es relativamente limitado, solo se cuenta con el IAR como alternativa confiable. El entorno de desarrollo IAR es relativamente inestable, bloqueándose con frecuencia. Estos problemas no ocurrieron con el compilador gcc.
- En cuanto a los compiladores para ARM existen diferencias en la forma como se traducen los programas que pueden tener consecuencias importantes y es necesario tener en cuenta. Durante el proceso de depuración del protocolo spimp se encontró que en el coprocesador compilando con IAR ocurría un problema en la comunicación través del bus SPI (la repetición de un byte se recibía en el procesador principal con un valor distinto del esperado). Al compilar con gcc el mismo código el problema no ocurría.

La depuración del ARM realizada con gdb y OpenOCD presenta problemas a la hora de utilizar el modo de bajo consumo del micro. Si el micro se encuentra en modo de bajo consumo, openOCD, a pesar de ser capaz de escribir el código en la memoria flash, no es capaz de abrir la conexión necesaria para realizar la depuración con gdb.

El uso de los leds como herramienta de depuración resultó ser muy efectivo. A pesar del muy limitado nivel de información que se puede obtener de los leds, su uso permite con un nivel mínimo de intrusión, obtener retroalimentación del estado de ejecución. Un simple led indicando que se produjo un error o que se produjo el resultado esperado brinda una forma rápida y segura de corroborar el correcto funcionamiento de una sección de código. La depuración línea por línea aunque poderosa, afecta en ocasiones el funcionamiento del sistema, por ejemplo cuando se tienen timeouts.

Proceso

En este capítulo se presenta el proceso de desarrollo realizado durante la ejecución del proyecto. Primero se brinda una explicación de la metodología de trabajo utilizada, se enumeran luego las distintas herramientas de gestión usadas y finalmente se presentan datos estadísticos del esfuerzo dedicado.

7.1. Metodología de Trabajo

El desarrollo del proyecto se llevó a cabo en coordinación con el equipo del IIE encargado de la construcción del satélite, así como también con el responsable de integración quien coordinaría el desarrollo de Software. A lo largo del proyecto se fue adaptando el alcance a las necesidades planteadas por el equipo del IIE, debiendo cambiar los objetivos según el progreso realizado y la fecha límite de lanzamiento.

El proyecto puede ser dividido en cuatro grandes etapas.

Durante la etapa inicial se desarrolló la investigación del estado del arte y el proceso de armado del ambiente de trabajo. Las principales tareas correspondientes a esta etapa fueron:

- Lectura de proyectos de grado y otra documentación relacionada al proyecto.
- Documentación parcial del estado del arte.
- Configuración del ambiente de desarrollo.
- Lectura de documentación de los microprocesadores.
- Pruebas de uso de los kits de desarrollo.

Se encontró gran dificultad a la hora de configurar el ambiente de desarrollo para el kit ARM. Se realizaron pruebas con diversos entornos de desarrollo como IAR para ARM, Code Composer, Eclipse y GNUARM a fin de definir cual utilizar, decidiendo usar este último. El entorno de desarrollo usado para MSP430 fue IAR. La duración de esta etapa fue de dos meses, debiendo suspenderse la documentación del estado del arte ante la necesidad de comenzar con el diseño del EMS.

La segunda etapa se corresponde con el desarrollo del Módulo de Gestión de Energía. Esta etapa se subdivide en cuatro subetapas.

- **Análisis** En paralelo a las tareas de la etapa inicial, se desarrollaron reuniones semanales con el resto del equipo de AntelSat. En estas reuniones participaron el responsable de integración, el equipo de EMS, el responsable de ADCS y el Ing. Javier Ramos, que trabajara junto a el equipo de EMS en el desarrollo de la plataforma de hardware de dicho módulo en su proyecto de grado. Durante estas reuniones se relevaron los requerimientos del módulo de gestión de energía, elaborándose un documento de especificación de requerimientos con los datos obtenidos.
- **Diseño** Una vez establecida una versión estable del documento de requerimientos, se prosiguió a elaborar un documento de diseño del módulo. En este se definieron las aplicaciones que compondrían el módulo, incluyendo un pseudocódigo para cada una, así como los recursos y variables de estado. Un documento de Referencias Cruzadas entre Requerimientos y Diseño se elaboró indicando para cada requerimiento la aplicación o aplicaciones diseñadas para resolverlo.
- **Implementación** Tras finalizar el diseño, se prosiguió a implementar las aplicaciones diseñadas. Debido a la cercanía de la fecha de lanzamiento inicial y la dificultad que ello supone, se eliminó del

alcance la capa de interacción con el hardware (hal), la cual sería implementada por el equipo del IIE, implementándose las aplicaciones sobre un stub del hal.

- **Verificación** Se realizó mediante pruebas unitarias de las aplicaciones no triviales y una serie de pruebas de integración. Antes de que el equipo de IIE finalizara la construcción del hal, se requirió comenzar con el trabajo en el ADCS. Se definió en este punto que la integración final sería realizada por el responsable de integración.

El proceso seguido en EMS se corresponde con un proceso de cascada, donde las subetapas anteriores se realizaron en forma secuencial.

En cuanto al proceso de análisis cabe notar que los requerimientos vinculados al encendido y apagado de los transmisores de Banda S y los detalles de la interacción con el módulo PAYLOAD fueron definidos en forma relativamente tardía, debido a que dependían de decisiones conjuntas entre el equipo IIE y el equipo de Antel. Esto sucedió también con la definición de las variables de telemetría, que dependían del diseño del hardware.

En la tercera etapa se realizó el desarrollo del ADCS. Al igual que en EMS, se dividió en cuatro subetapas.

- **Análisis** Una serie de reuniones con el responsable de ADCS fueron realizadas. Un documento de especificación de requerimientos del ADCS fue escrito a partir de las mismas.
- **Diseño** Se prosiguió a elaborar un documento de diseño del módulo. En este se definieron las aplicaciones que compondrían el módulo, incluyendo un pseudocódigo para cada una, así como los recursos y variables de estado. Un documento de Referencias Cruzadas entre Requerimientos y Diseño se elaboró indicando para cada requerimiento la aplicación o aplicaciones diseñadas para resolverlo. También se diseñaron en esta etapa los protocolos de comunicación entre el MSP430 y el ARM.
- **Implementación** Se definió que el alcance para ADCS no comprendería las aplicaciones del MSP430, por lo que la implementación se concentró en el desarrollo de los protocolos de comunicación y el código a ser ejecutado por el ARM. Las aplicaciones diseñadas para el ADCS fueron implementadas como prototipos.
- **Verificación** Se implementaron diversas aplicaciones de prueba que hacen uso de los protocolos, y verifican los algoritmos que corren en el ARM. En esta etapa se realizaron las pruebas sobre el hardware de vuelo.

El trabajo en ADCS se corresponde más con un enfoque iterativo e incremental en el cual las tareas de análisis, diseño, implementación y verificación se realizaron en múltiples iteraciones. Cada iteración implicaba revisión de requerimientos (por ej. agregado o modificación de parámetros), implementación de cambios (por ejemplo incorporación de una nueva versión del código Matlab) y verificación de los cambios realizados (usando los programas de prueba como regresión). El algoritmo Matlab en particular comenzó como una cáscara vacía y se fue mejorando en revisiones sucesivas.

La definición de los datos de telemetría de ADCS fue un elemento de la especificación de requerimientos que se definió en forma relativamente tardía.

Durante la etapa final se escribió la documentación del proyecto.

En la Figura: 7.1 se presentan ordenadas cronológicamente las principales actividades realizadas durante el proyecto.

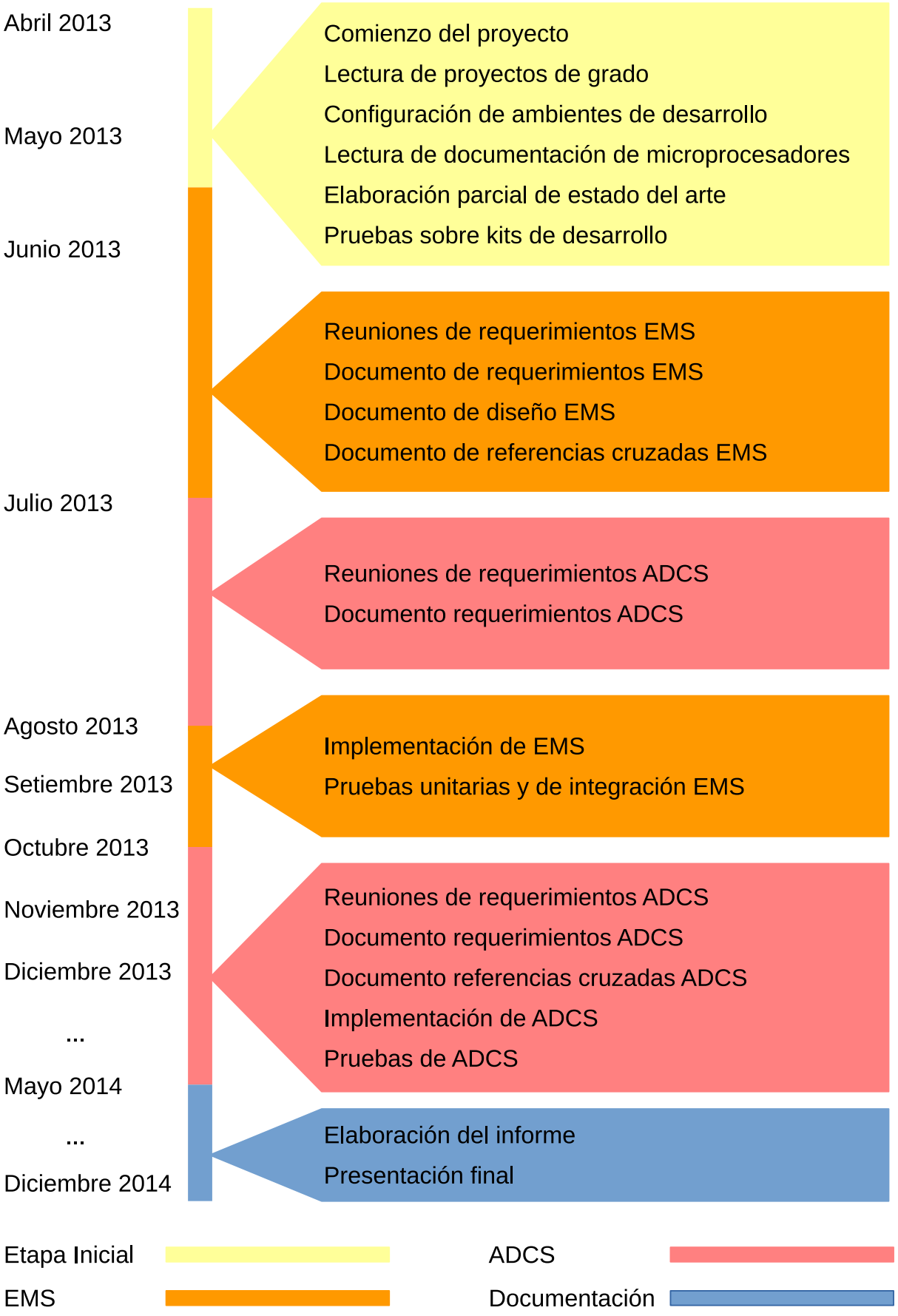

Figura 7.1. Distribución cronológica de actividades

7.2. Herramientas de Gestión Utilizadas

A continuación se enumeran las herramientas usadas.

- **Redmine** La gestión de tiempo fue realizada usando la herramienta Redmine. Se definieron peticiones por cada tarea que debiera ser realizada y se registró el tiempo que se empleó en realizar cada una. El servicio utilizado fue **hostedredmine** ([17]), sitio que provee una servidor de Redmine con acceso privado. Se evaluaron varias herramientas para gestión de proyectos. Las características buscadas eran que fuera una aplicación web, que soportara registro de tiempo y reportes y que fuera de acceso gratuito. Se eligió la herramienta Redmine ya que cumple con estas condiciones y ya estábamos familiarizados con la misma.
- **Google Drive** Toda la documentación generada durante el proyecto, como pueden ser documentos de requerimientos, diseño, datos de pruebas realizadas, etc. fue almacenado en Google Drive y compartido con todos los interesados del proyecto. Esta herramienta fue especialmente útil para la elaboración de los documentos de requerimientos, ya que permitió realizar preguntas en el propio documento que el equipo del IIE fue capaz de contestar sin necesidad de esperar hasta la siguiente reunión.
- **Assembla/Git** El control de versiones se realizó utilizando Git, en un repositorio almacenado en la herramienta en línea Assembla. El software del repositorio Git fue transferido periodicamente al repositorio SVN del proyecto AntelSat.

7.3. Estadísticas

En esta sección se presenta un desglose del esfuerzo realizado durante el proyecto por área de trabajo. Para ello se definieron las siguientes categorías:

- **Configuración** Tareas relacionadas a la configuración del ambiente de desarrollo. Esto comprende instalación de máquinas virtuales y sistemas operativos, selección e instalación de herramientas, obtención de licencias, configuración de proyectos, etc.
- **Estudio** Tareas de autoestudio, como pueden ser lectura de proyectos de grado, papers o documentación de hardware.
- **Análisis** Comprende reuniones de obtención de requerimientos y elaboración de documentos de especificación de requerimientos.
- **Diseño** Comprende elaboración de documentos de diseño y referencias cruzadas y reuniones de validación del diseño.
- **Implementación** Comprende el desarrollo del sistema. Tiempo empleado en escribir el código.
- **Verificación** Tareas de escritura y ejecución de pruebas unitarias, de integración y de hardware. Esto incluye el tiempo dedicado en depurar y corregir los errores detectados.
- **Gestión** Tareas relacionadas a la gestión del proyecto. Esto comprende planificación y reparto de tareas, planificación de reuniones, etc.
- **Comunicación** Tiempo empleado en reuniones de coordinación con el tutor del proyecto o el equipo de IIE.
- **Documentación** Elaboración del Informe y documentos anexos.

La utilización de una herramienta de gestión para coordinar y registrar las tareas se comenzó a realizar al iniciar la implementación del sistema. El esfuerzo realizado durante los meses anteriores en actividades de Configuración, Estudio, Análisis y Diseño no fue registrado.

Con el fin de dar un panorama general del esfuerzo realizado, se estimaron dichas categorías. El criterio fue tomar un aproximado de 10 horas semanales por persona y distribuirlos en cada categoría según la etapa en la que se estaba trabajando en cada semana. Se utilizó como insumo los registros de correos electrónicos y el registro de versiones de los documentos de requerimientos y diseño.

A continuación se presentan los datos recabados:

Total de Horas Registradas
1707

Tabla 7.1. Total de horas registradas sin incluir estimaciones.

Total de Horas
2091

Tabla 7.2. Esfuerzo total realizado incluyendo estimaciones.

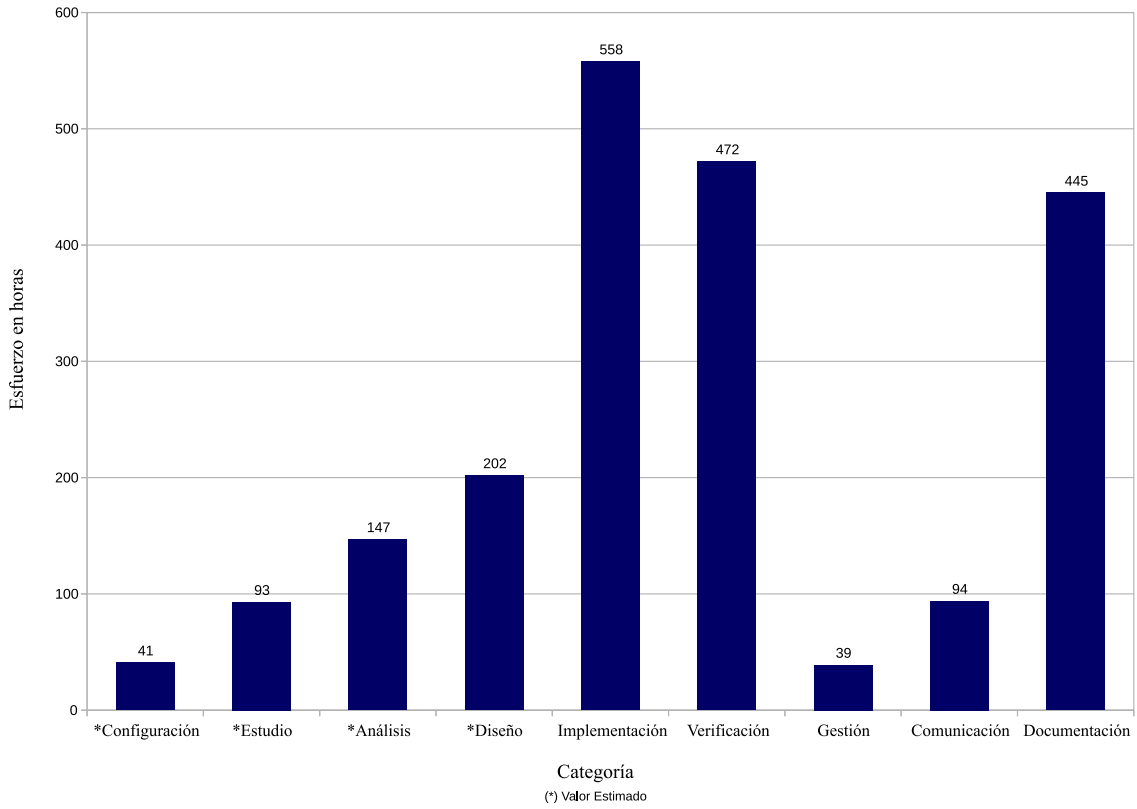

Figura 7.2. Esfuerzo desglosado por categoría.

Podemos realizar aquí una comparación entre el trabajo de verificación en EMS y ADCS. Dado que nuestro conteo de dedicación en verificación incluye el tiempo dedicado en depurar y corregir los problemas, la cantidad de errores detectados y el tiempo total aportan una medida de la complejidad en la resolución de los mismos. En EMS se dedicaron 113 horas hombre y se detectaron 10 errores en total, de forma que podemos establecer una dedicación media de 11.3 horas hombre por error detectado. En ADCS se dedicaron 359 horas hombre y se detectaron 19 errores, correspondiendo a una dedicación media de 18.9 horas hombre por error detectado. Podemos concluir entonces que el trabajo de verificación de ADCS fue en general más laborioso que el de EMS.

Esto se puede explicar por los siguientes factores:

- Las pruebas de ADCS involucraron manejo de hardware (bus SPI) elemento que no estuvo presente en la verificación de EMS.

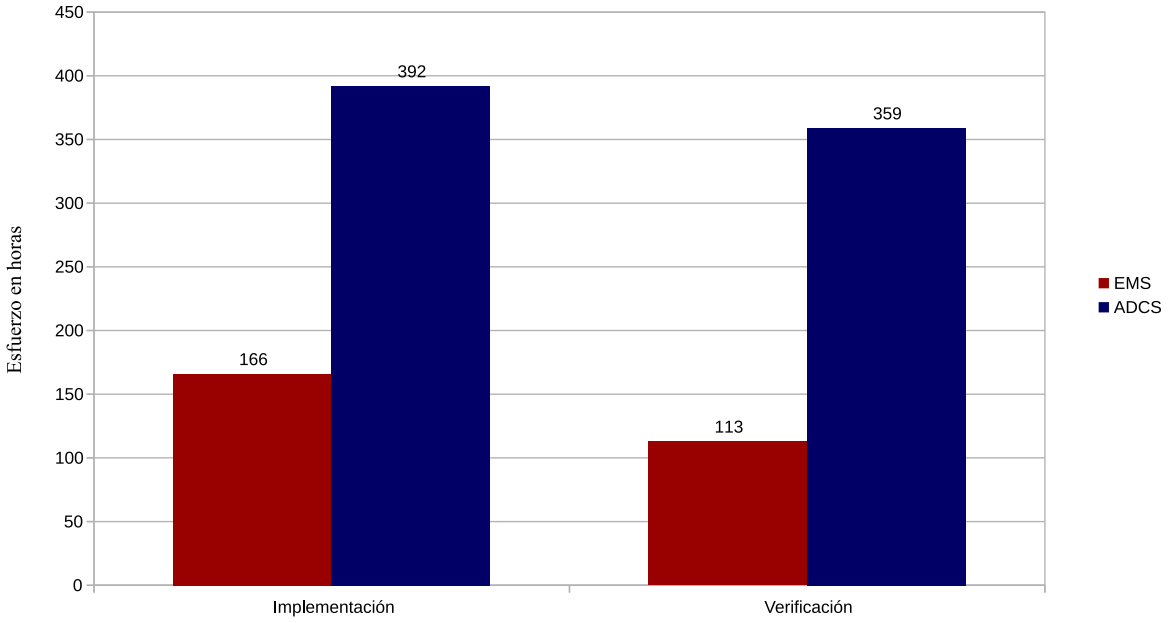

Figura 7.3. Esfuerzo de Implementación y Verificación por módulo.

- ADCS contiene un conjunto de algoritmos con múltiples variables de salida, elemento que no estaba presente en EMS. La verificación de estos algoritmos requirió el trabajo conjunto con el responsable de ADCS que proveyó los datos de prueba, agregando una complejidad de organización.

Síntesis

A continuación se presenta una breve resumen de los capítulos anteriores.

8.1. Módulo de Gestión de Energía

Las funciones del EMS son:

- Capturar la energía solar a través de paneles solares.
- Proveer energía a los diversos componentes del satélite.
- Almacenar en baterías la energía remanente.
- Encender y apagar los restantes módulos según sea necesario.
- Llevar registro del estado de los restantes módulos (encendido, apagado, falla).
- Reportar a la Tierra el estado de salud del satélite y otros datos básicos.

Se realizaron una serie de reuniones para relevar los requerimientos de software para EMS. Esto permitió aclarar y sistematizar el problema a resolver. El resultado de este trabajo se registró en un documento de Especificación de Requerimientos cuyo contenido se puede resumir en las siguientes funcionalidades:

- Inicializar el satélite y desplegar las antenas.
- Extraer energía de los paneles solares con la mayor eficiencia posible y almacenarla en baterías.
- Permitir un funcionamiento de bajo consumo a fin de recuperar energía en caso de agotamiento de la misma por operación de los componentes del satélite.
- Recibir y procesar mensajes provenientes de otros módulos del satélite.
- Enviar a otros módulos del satélite información sobre su propio estado.
- Controlar el encendido y apagado de los demás módulos siguiendo cierta lógica.
- Enviar a la Tierra un mensaje morse ("baliza") conteniendo información básica sobre el estado de salud de todo el satélite.
- Servir como respaldo de la hora para MCS.
- Notificar al MCS cuando ocurre un cortocircuito o sobre-consumo de un módulo.

Se diseñó e implementó un prototipo de software en base a los requerimientos anteriores. Este prototipo incluye la lógica de aplicación necesaria para resolver los requerimientos. Los componentes asociados al control de los dispositivos fueron desarrollados por el responsable de integración paralelamente.

El prototipo fue sometido a pruebas de software que permitieron detectar y corregir 10 errores de programación. Se realizaron pruebas unitarias usando la herramienta **MinUnit** que se ejecutaron en PC. Las pruebas de integración se ejecutaron en el simulador del IAR. Esto permite visualizar los mensajes generados por el programa con mayor rapidez que al ejecutar en el kit de desarrollo. La gran cantidad de módulos del programa dificultó el proceso de verificación.

La versión final del software de EMS fue construida desde cero por el responsable de integración basándose en el prototipo desarrollado por nuestro equipo.

8.2. Módulo de Determinación y Control de actitud

El ADCS tiene los siguientes objetivos:

1. Determinar la orientación del satélite con respecto a la Tierra.
2. Detener la rotación del satélite y orientarlo de forma de apuntar la cámara hacia la Tierra.

El análisis de requerimientos de ADCS se realizó en una serie de reuniones con el responsable de ADCS y el responsable de integración. El resultado de este trabajo se registró en un documento de Especificación de Requerimientos cuyo contenido se puede resumir en las siguientes funcionalidades:

- Estabilizar el giro del satélite cuando sale del P-POD.
- Llevar registro del estado de actitud del satélite, el cual indica la estabilidad del mismo y si se encuentra con la cámara orientada hacia la Tierra o no.
- Determinar la orientación del satélite con respecto a la Tierra. Se debe realizar mediante los algoritmos QuEst y Filtrado de Kalman.
- Controlar la orientación de forma de apuntar la cámara hacia la Tierra. Se debe disponer los siguientes algoritmos de control: PID, LQR, PD_PR y Sliding Mode Controller.
- Desarrollar un protocolo de comunicaciones que permita integrar los dos microcontroladores que forman el ADCS.
- Mantener una copia local de la hora y el TLE y los parámetros que se reciben desde MCS.
- Enviar periódicamente a MCS un reporte de telemetría conteniendo ciertas variables de interés.
- Comunicar la orientación del satélite a los demás módulos.
- Enviar a EMS un reporte de estado para baliza morse.
- Notificar a MCS cuando ocurran cambios de estado o errores en periféricos.

La división de trabajo acordada para ADCS consistió en que nuestro equipo se encargaría de resolver el problema de la comunicación entre los microcontroladores que forman el módulo y la integración de los algoritmos de determinación y control desarrollados por el responsable de ADCS en Matlab. De todos modos se desarrolló también un prototipo del software de aplicación del procesador principal.

Se diseñó e implementó un protocolo de transferencia de mensajes sobre SPI, spimp. Se diseñó e implementó un protocolo de invocación remota sobre spimp, spirpc, para ejecutar algoritmos en el coprocesador desde el procesador principal.

Se realizaron pruebas para el software de ADCS. Esto incluyó pruebas unitarias y pruebas de integración, tanto de caja blanca como de caja negra. Algunas se ejecutaron en PC. Otras fueron ejecutadas en los kit de desarrollo y finalmente en el hardware de vuelo. Se detectaron y corrigieron 19 errores.

Los protocolos spimp y spirpc y el resto del software desarrollado por nuestro equipo para el coprocesador se usaron íntegramente al construir la versión final de software de ADCS.

8.3. Proceso

El trabajo en el proyecto se puede dividir en cuatro grandes etapas.

La primer etapa consistió en la investigación del estado del arte y la configuración de los entornos de desarrollo. Para el kit de desarrollo ARM se evaluaron varios entornos decidiendo finalmente usar GNU ARM Embedded. Esta etapa duró dos meses. Los requerimientos y el alcance recibieron ajustes a lo largo del proyecto en virtud de los cambios en la fecha de lanzamiento.

En la segunda etapa se llevó a cabo el trabajo de desarrollo y verificación de EMS. Se realizaron una serie de reuniones con el equipo de IIE para definir los problemas a resolver, plasmando el resultado en un documento de Requerimientos de EMS. Se diseñó e implementó un prototipo del software de EMS. Dado el ajustado cronograma establecido por la fecha de lanzamiento se decidió dejar fuera del alcance de

nuestro proyecto el desarrollo de la capa de abstracción de hardware, quedando esto a cargo del equipo de IIE. Se realizaron pruebas unitarias y pruebas de integración para el prototipo de software desarrollado.

La tercer etapa estuvo asociada al ADCS. Se realizaron una serie de reuniones con el responsable de ADCS con el fin de relevar los requerimientos de software, elaborándose un documento de especificación de requerimientos. Se diseñaron e implementaron las aplicaciones del procesador principal del ADCS, tomando al misma organización del trabajo empleada en EMS donde la capa de abstracción de hardware fue desarrollada por el equipo de IIE. Se diseñó e implementó un protocolo de comunicaciones para integrar los dos microcontroladores del ADCS. Se diseñó e implementó el software del coprocesador, integrando los algoritmos desarrollados por el responsable de ADCS en Matlab. Dado el escaso tiempo disponible se decidió que el software desarrollado para el procesador principal quedara fuera del alcance y fuera construido de nuevo por el responsable de integración. Esto permitió que nuestro equipo se enfocara en el software del coprocesador y la integración de los algoritmos Matlab. Se realizaron pruebas unitarias y de integración para el software del coprocesador, tanto de caja blanca como de caja negra, ejecutando en PC, en los kits de desarrollo y en el hardware de vuelo. Este trabajo de verificación permitió detectar y corregir 19 errores.

La etapa final correspondió a la elaboración de este documento.

Se empleó la herramienta Redmine para seguimiento de tareas y registro de tiempos. Se evaluaron múltiples opciones de software para gestión de proyecto optando por Redmine dado que proveía las características buscadas y ya estábamos familiarizados con él.

También se usó Google Drive para la edición de documentos en forma colaborativa. Esto fue especialmente útil en la comunicación con el equipo de IIE permitiendo aclarar dudas ágilmente.

La gestión de configuración se realizó con un repositorio Git provisto por el sitio Assembla. El software del repositorio Git fue transferido periódicamente al repositorio SVN del proyecto AntelSat.

El proyecto insumió un total de 2091 horas hombre. La mayor parte del esfuerzo estuvo concentrado en tareas de implementación y verificación. Si se desglosa por módulo, la mayor parte del esfuerzo correspondió al trabajo en ADCS.

Conclusiones

Se atacó el problema del desarrollo de software para dos módulos del AntelSat, el Módulo de Gestión de Energía y el Módulo de Determinación y Control de Actitud. Si bien el software no es grande en volumen (la mayoría de los módulos tienen menos de 200 líneas de código y algunos menos de 100) las condiciones de operación hostiles del espacio y la imposibilidad de operar el dispositivo directamente plantean exigencias particulares. El diseño tanto de software como de hardware se orienta a maximizar la independencia y robustez.

El análisis de requerimientos realizado revela un problema de software complejo, tanto en el caso del Módulo de Gestión de Energía como en el Módulo de Determinación y Control de Actitud. Los requerimientos del software estan fuertemente ligados al diseño del hardware realizado por el equipo de IIE. Algunas características de la plataforma de hardware no estaban definidas cuando se realizó el análisis de requerimientos y fueron determinadas posteriormente (lo cual sucedió por ejemplo con las especificaciones de telemetría y la comunicación con PAYLOAD relativa a los transmisores de Banda S). Esto demoró la estabilización de los requerimientos.

Algunos componentes de infraestructura comunes a varios de los módulos del satélite como ser el microkernel y el protocolo de comunicación sobre I^2C sufrieron revisiones a lo largo del proyecto dando lugar a cambios en el software.

El proyecto tuvo un fuerte componente de comunicación y trabajo conjunto con el grupo de IIE a lo largo de todo su transcurso. Sirva como ejemplo el hecho de que la mayor parte de las horas dedicadas en verificación del Módulo de Determinación y Control de Actitud corresponden a trabajo realizado en IIE junto con el responsable de ADCS.

Las restricciones del proyecto AntelSat involucraban el desarrollo completo de algunos de los módulos de interacción con hardware, a fin de minimizar riesgos por introducción de bibliotecas de terceros. El desarrollo de estos componentes implica un estudio detallado del microcontrolador, trabajo que está detrás de los protocolos desarrollados para integrar los dos microcontroladores del Módulo de Determinación y Control de Actitud.

El uso de LEDs como herramienta de depuración probó tener ventajas sobre el uso de mensajes de depuración. Permiten señalizar condiciones con claridad y hacen posible ejecutar la prueba en el hardware objetivo sin usar del depurador, siendo este necesario solo para depurar problemas.

En el Módulo de Determinación y Control de Actitud nuestro equipo atacó el problema del software del coprocesador y la integración con el procesador principal al través de un bus SPI. Para la verificación se empleó un enfoque basado en una combinación de pruebas en PC, en kits de desarrollo y en el hardware de vuelo usando LEDs para señalizar condiciones.

La ejecución en PC fue efectiva para la verificación y depuración de los algoritmos. Una vez que estos produjeron los resultados esperados las pruebas en hardware de vuelo con comunicación SPI no generaron mayores dificultades. En el desarrollo para MSP430 también permitió detectar algunos problemas que pasan desapercibidos con IAR pero que al compilar en GCC generan errores en tiempo de compilación.

En la verificación del Módulo de Determinación y Control de Actitud los problemas encontrados del lado del MSP430 fueron más simples de depurar en comparación con las pruebas del Módulo de Gestión de Energía debido a que la cantidad de aplicaciones involucradas fue menor; sin embargo la presencia de la comunicación SPI y los algoritmos de determinación y control hicieron más laborioso el proceso de prueba del Módulo de Determinación y Control de Actitud en términos generales.

Se trabajó exitosamente con la biblioteca *Standard Peripheral Library* en ARM. Se integró sin mayores dificultades y fue efectiva para resolver los requerimientos puntuales de manejo de periféricos del lado del ARM que fueron requeridos para el proyecto.

Se trabajó exitosamente con el compilador gcc en ARM, permitiendo sacar provecho del soporte en hardware para cálculos de punto flotante. La experiencia con OpenOCD también fue buena, integrándose con facilidad y no presentando mayores problemas.

En conclusión, como producto generado por este trabajo, se logró la implementación de un prototipo de software del Módulo de Gestión de Energía, el cual sirviera como base para el desarrollo posterior del software final del satélite. El proceso de relevamiento de requerimientos hizo posible aclarar el problema y fue el marco sobre el cual se basó el trabajo de software posterior, tanto el prototipo como la versión final.

Además, se obtuvo una versión final, instalada en el satélite puesto en orbita el 19 de Junio del 2014, del software del coprocesador del Módulo de Determinación y Control de Actitud, así como de los protocolos de comunicación entre el procesador principal y el coprocesador de éste módulo.

Finalmente, se obtuvo la experiencia de haber participado en un proyecto de desarrollo de tecnología espacial de relevancia histórica en nuestro país, oportunidad por la cual agradecemos a todos los involucrados.

Trabajos futuros

En esta sección se proponen alternativas para continuar el trabajo del presente proyecto y del proyecto AntelSat en general.

- **Reimplementación de ADCS en un solo microcontrolador ARM**. La inclusión del microcontrolador ARM en ADCS implicó la necesidad de integrar los dos microcontroladores del módulo. Un trabajo futuro posible es construir el ADCS entero usando un solo microcontrolador ARM, eliminando la complejidad adicional asociada a la comunicación entre microcontroladores.
- **Verificación de EMS y ADCS en entorno Simulink**. Cuando se comenzó el trabajo en el proyecto se previó un mecanismo para verificar el comportamiento de los módulos en forma integral. La idea consiste en usar el entorno de simulación Simulink ([28]), ejecutando en un PC al cual se conecta el módulo de hardware a verificar con su correspondiente software instalado. La conexión permite que cuando el módulo realiza lecturas de sensores obtenga datos provenientes del entorno de simulación, y cuando comanda actuadores los resultados se envían a dicho entorno. De esta manera la lógica de control del módulo ejecutando en el hardware objetivo se somete a una prueba que pemite simular las condiciones de operación del espacio y registrar los resultados. Este mecanismo puede aplicarse tanto a EMS como a ADCS. Estas pruebas no se pudieron implementar por falta de tiempo. Por lo tanto un trabajo futuro interesante sería la realización de las mismas.
- **Verificación de caja blanca de protocolo SPIMP en ADCS**. Las pruebas de integración de caja blanca para el coprocesador de ADCS cubren un conjunto de casos de borde que simulan errores en la comunicación SPI. Agregar casos de prueba para obtener un cubrimiento de código del 100 %.
- **Verificación en modelo de ingeniería**. Diseñar pruebas de integración para todo el satélite y ejecutarlas sobre el modelo de ingeniería. Incluir envíos de telecomandos y verificar su correcto procesamiento.
- **Modo de experimento de ADCS**. Los requerimientos de ADCS inicialmente incluían un modo de operación en el cual el módulo generaría un registro de todas las medidas de sensores realizadas para enviar a la Tierra. Este modo de experimento proveería información de diagnóstico para evaluar el funcionamiento del módulo y depurar problemas, pero por motivos de tiempo se quitó del alcance. Es deseable como trabajo a futuro implementar este requerimiento.
- **Verificación de integridad en spimp con CRC**. El protocolo spimp realiza eco de cada byte recibido a fin de verificar que no ocurrió una mutación de datos en la transmisión. Sustituir esto por verificación por medio de un CRC puede reducir el tiempo de transferencia de mensajes, al eliminar la verificación de cada byte y el paso de confirmación de integridad por parte del emisor. Implementar este cambio y comparar el rendimiento con la versión actual.

Bibliografía

[1] AAUSATLAB. AAUSAT. (Disponible en: http://www.space.aau.dk/ Consultado el: 12 de Enero del 2015)

[2] ALONSOPEREZ, V.; CASTRO, A.; ZITO, S. 2011. Sistema de Determinación de Actitud. Tesis Ing. Elect. Universidad de la República, Uruguay.

[3] ALPUY, S.; CABALO, P.; FORETS, M. 2012. Integración de estación terrena. Tesis Ing. Elect. Universidad de la República, Uruguay.

[4] ANTEL. AntelSat. (Disponible en: http://www.antel.com.uy/antelsat Consultado el: 12 de Enero del 2015)

[5] ARDAO, M.; FERNÁNDEZ, M.; MATO, M. 2012. Módulo de control principal y control de actitud. Tesis Ing. Elect. Universidad de la República, Uruguay.

[6] CALIFORNIA STATE POLYTECHNIC UNIVERSITY. 2009. CubeSat Design Specification Rev. 12. (Disponible en http://www.cubesat.org/images/developers/cds_rev12.pdf. Consultado en: Marzo de 2014).

[7] CELESTRACK. 1995. Orbital Coordinate Systems, Part I: (Disponible en: http://celestrak.com/columns/v02n01/ Consultado el: 12 de Enero del 2015)

[8] DE LEÓN, I.; RAMOS, J.; SOTTA, G. 2011. Desarrollo del sistema de gestión de energía para un satélite. Tesis Ing. Elect. Universidad de la República, Uruguay.

[9] DE MARTINO, G. 2013. Software y Protocolos para CubeSat. Tesis Ing. Comp. Universidad de la República, Uruguay.

[10] DE MARTINO, G. 2013. uKernel Manual del usuario. Anexo Tests Ing. Comp. Universidad de la República, Uruguay.

[11] EPFL. SwissCube. (Disponible en: http://swisscube.epfl.ch/ Consultado el: 12 de Enero del 2015)

[12] LOVERA, M.; SILANI, E. 2005. Magnetic spacecraft attitude control: a survey and some new results. Control Engineering Practice 13.3.

[13] FING. Satélite AntelSat. (Disponible en: http://iie.fing.edu.uy/investigacion/grupos/lai/ Consultado el: 12 de Enero del 2015)

[14] GNU. GCov, a Test Coverage Program. (Disponible en: http://gcc.gnu.org/onlinedocs/gcc/Gcov.html Consultado el: 12 de Enero del 2015)

[15] GNUARM. 2011. GNU ARM Embedded Toolchain. (Disponible en: https://launchpad.net/gcc-arm-embedded Consultado el: 12 de Enero del 2015)

[16] GUTIÉRREZ, G.; LANZARI, 2011. F. Sistema de telemetría, telecomando y control. Tesis Ing. Elect. Universidad de la República, Uruguay.

[17] HOSTED REDMINE. Free Redmine Hosting (Disponible en: http://www.hostedredmine.com/ Consultado el: 12 de Enero del 2015)

[18] IAR. IAR Embedded Workbench. (Disponible en: http://www.iar.com/Products/IAR-Embedded-Workbench/ Consultado el: 12 de Enero del 2015)

[19] KALLMAN, R.E. 1960. A New Approach to Linear Filtering and Prediction Problems. Journal of Basic Engineering, No. 82 (Series D). ISBN: 9780470544334

[20] KROGH, K.; SCHREDER, E. 2002. Attitude Determination for AAU CubeSat. Aalborg University, Institute of Electronic Systems, Department of Control Engineering.

[21] LCOV. LCOV. (Disponible en: `http://ltp.sourceforge.net/coverage/lcov.php` Consultado el: 12 de Enero del 2015)

[22] MINUNIT. MinUnit, a minimal unit testing framework for C. (Disponible en: `http://www.jera.com/techinfo/jtns/jtn002.htmlhttp://www.jera.com/techinfo/jtns/jtn002.html` Consultado el: 12 de Enero del 2015)

[23] MSP430F5XX. 100-Pin Target board and USB Programmer. (Disponible en: `http://www.ti.com/tool/msp-fet430u5x100` Consultado el: 12 de Enero del 2015)

[24] NOAA. 2014. International Geomagnetic Reference Field. (Disponible en: `http://www.ngdc.noaa.gov/IAGA/vmod/igrf.html` Consultado el: 12 de Enero del 2015)

[25] NOE, C. 2004. Design and Implementation of the Communications Subsystem for the Cal Poly CP2 Cubesat Project. Computer Engineering Department California Polytechnic State University, San Luis Obispo. (Disponible en: `http://polysat.calpoly.edu/download/ChrisNoe_srproj.pdf` Consultado el: 12 de Enero del 2015)

[26] OPENOCD. Open On-Chip Debugger. (Disponible en: `http://openocd.sourceforge.net/` Consultado en: Noviembre 2014)

[27] PASTORINO, E.; REYES, N.; SCAPINELLO, I. 2009. Telemetría para Globosats. Tesis Ing. Elect. Universidad de la República, Uruguay.

[28] SIMULINK. Simulation and Model-Based Design. (Disponible en: `http://www.mathworks.com/products/simulink/index.html` Consultado el: 12 de Enero del 2015)

[29] SHUSTER, M.D.; OH S.D. 1981. Three-axis attitude determination from vector observations. Journal of Guidance, Control, and Dynamics, Vol. 4, No. 1. ISSN: 0731-5090

[30] STK. Systems Tool Kit. (Disponible en: `http://www.agi.com/products/stk/` Consultado el: 12 de Enero del 2015)

[31] STMICROELECTRONICS. 2012. ARM Cortex-M4F 32b MCU+FPU. (Disponible en: `http://www.st.com/web/en/resource/technical/document/datasheet/DM00037051.pdf` Consultado el: 12 de Enero del 2015).

[32] STMICROELECTRONICS. STM32F3 Discovery. (Disponible en: `http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF254044?sc=internet/evalboard/product/254044.jsp` Consultado el: 12 de Enero del 2015)

[33] STMICROELECTRONICS. 2012. STM32F302xx, STM32F303xx and STM32F313xx advanced ARM-based 32-bit MCUs Reference Manual.

[34] TASSANO, M. 2012. A B-dot controller for detumbling ANTELSAT. Universidad de la República, Uruguay.

[35] TASSANO, M. 2012. Introducción ADCS y algoritmos de control. Presentación, Universidad de la República, Uruguay.

[36] TEXASTEXAS INSTRUMENTS. 2012. MSP430F543xA, MSP430F541xA Mixed Signal Microcontrollers Rev. B.

[37] VALLADO, D. 2010. Astrodynamics Software, Fundamentals of Astrodynamics and Applications, Computer Software in C++.

[38] WERTZ, J.R. 1959. Spacecraft Attitude Determination and Control. Kluwer Academic Publishers.

Glosario

actitud Orientación de un objeto con respecto a un sistema de referencia inercial. 14, 16–24, 33, 37, 38, 45, 53–59, 61–63, 74, 76, 80, 90

ADC Analog to Digital Converter, Dispositivo capaz de traducir una señal continua como el voltaje en un número que representa su amplitud. 21, 36

ADCS Attitude Determination and Control System (Módulo de Determinación y Control de Actitud). Es el módulo encargado de controlar la orientación del satélite. 16, 17, 21, 22, 24, 29, 34, 36, 37, 39–43, 45, 51, 53–63, 73, 74, 76–80, 83, 84, 87, 88, 90, 91, 93, 95

Antel Empresa estatal de telecomunicaciones de la República Oriental del Uruguay. 13, 16, 84

AntelSat Proyecto de colaboración entre Antel y Facultad de Ingeniería destinado a la construcción y puesta en orbita de un sátelite clase CubeSat. 5, 13–16, 18, 19, 22, 24, 25, 33, 34, 40, 50, 53, 83, 86, 91, 93, 95

ARM Denominación dada en el documento al microcontrolador Cortex M4 de la compañía ARM utilizado como coprocesador del ADCS. 24, 33, 55, 57, 58, 60, 64, 66, 74, 76–81, 83, 84, 90, 94, 95

AX25 Protocolo de capa de enlace diseñado para ser usado por operadores de radio. 17

baliza Señal de onda portadora que se envía periódicamente conteniendo información de estado del satélite. 16, 30–32, 35–39, 41, 42, 44–46, 48, 61, 89, 90

banda-S La banda S es un rango frecuencial que va desde 1,5 a 5,2 GHz. Es parte de la banda de microondas del espectro electromagnético. 42–44

COMM1 Uno de los dos módulos encargados de la transmisión y recepción de datos. 17, 36–42, 45, 51

COMM2 Uno de los dos módulos encargados de la transmisión y recepción de datos. 17, 36–42, 45, 51

CubeSat Estandard para la fabricación de nanosatélites de bajo costo desarrollado por la Universidad Estatal Politécnica de California y la Universidad de Stanford. 1, 13, 15, 17, 18, 24, 28

digipeater Servicio de repetición de paquetes de datos recibidos a través de AX.25. 37

ECEF Earth Centered, Earth Fixed, Sistema de referencia no inercial utilizado comúnmente para expresar posiciones estáticas sobre la superficie terrestre. 18, 63, 74, 75

ECI Earth Centered Inertial, Sistema de referencia inercial utilizado comúnmente para expresar movimientos orbitales. 18–22, 59, 63, 74, 75

EMS Energy Management System (Módulo de Gestión de Energía). Es el módulo encargado de gestionar la alimentación energética de todo el satélite. 16, 24, 28, 29, 31, 32, 34–40, 42–52, 56, 59–61, 78, 83, 84, 87–91, 95

fecha juliana Escala de tiempo calculada como la cantidad de días y fracción transcurridos desde el inicio del periodo Juliano, el mediodía del 1º de enero del año 4713 A.C.. 20, 63

fotodiodo Dispositivo sensible a la luz que entrega una corriente de salida que varía con el ángulo de incidencia de la luz sobre su superficie. 16, 18, 20, 21, 23, 24, 57–59, 63

GPIO General Purpose Input/Output, Pin genérico en un chip cuyo comportamiento se puede controlar en tiempo de ejecución. 78

HAL Hardware Abstraction Layer, Capa de software encargada de la configuración y utilización de los distintos periféricos, brindando una interfaz de mas fácil uso al resto del sistema. 47–50, 60, 62, 77

I²C Inter-Integrated Circuit, bus de comunicaciones en serie diseñado por Phillips con el cual se comunican los módulos del satélite. Nombre del módulo que controla el bus I^2C. 16, 21, 22, 24, 30, 32, 33, 36, 37, 39–42, 46, 51, 60, 93

IAGA International Association of Geomagnetism and Aeronomy, Asociación no gubernamental dedicada al estudio de las propiedades eléctricas y magnéticas de la Tierra y la interacción con la atmósfera. 22

IAR IAR Systems, Empresa Suiza proveedora de herramientas para el desarrollo de software embebido, particularmente del IDE utilizado en este Proyecto. 23, 33, 49, 50, 52, 76, 79, 81, 83, 89, 93

IGRF International Geomagnetic Reference Field, Descripción matemática estándar del campo magnético Terrestre creado por IAGA. 21–23, 62, 63, 74, 76

IIE Instituto de Ingeniería Electrica de la Universidad de la República. 13, 83, 84, 86, 90, 91, 93

immp Inter Module Message Protocol, Protocolo de comunicaciones sobre I^2C mediante el cual se comunican los módulos del satélite. 42, 46, 60

JTAG Joint Test Action Group. Estándar de la IEEE que define un mecanismo para verificación y depuración de circuitos integrados ampliamente usado. 21

magnetorquer Dispositivo capaz de crear un campo magnético el cual al interactuar con el campo magnético terrestre es capaz de generar fuerzas de rotación en un satélite en orbita. 16, 54, 55, 59, 60, 63

magnetómetro Dispositivo capaz de medir la intensidad y dirección de un campo magnético. 18, 20, 21, 23, 24, 54, 55, 57

MCS Main Control System (Módulo de Control Principal). Es el módulo encargado de procesar la comunicación entre la Tierra y el resto de los módulos del satélite. 16, 17, 24, 29–32, 36–46, 51, 55, 57, 60, 61, 80, 89, 90

MISO Master Input Slave Output. Línea de control de SPI. Entrada de datos al Maestro y salida de datos del Esclavo. 24, 65, 78

morse Código Morse, es un sistema de representación de letras y números mediante el uso de señales intermitentes que representan puntos y rayas. Para cada letra se asocia una combinación determinada de puntos y rayas. 32, 35, 38, 41, 42, 44, 46, 48, 61, 89, 90

MOSI Master Output Slave Input. Línea de control de SPI. Salida de datos del Maestro y entrada de datos al Esclavo. 24, 65, 78

MPPT Maximum Power Point Tracking, Técnica utilizada para obtener la máxima potencia de los paneles solares. 30–33, 35, 39, 49–51

MSP430 Microcontrolador creado por Texas Instrument caracterizado por poseer una arquitectura RISC de bajo consumo. Es el principal tipo de microcontrolador utilizado en el satélite. 24, 33, 49, 50, 52, 55, 57–59, 63, 65, 74, 77–81, 83, 84, 93

nadir Nombre dado a la vertical entre un observador y el centro de la Tierra apuntando en la dirección de la fuerza de gravedad. 17, 54, 61

NED North East Down, Sistema de referencia comúnmente usado en aviación con su centro en el centro de gravedad del vehículo. 18, 22

NORAD North American Aerospace Defense Command, Organización conjunta de Estados Unidos y Canadá que provee defensa y control aéreo en Norteamérica. 18, 21

P-POD Poly-PicoSatellite Orbital Deployer, Mecanismo de transporte y despliegue para la puesta en orbita de CubeSats. 15, 28, 29, 31, 32, 35, 90

PAYLOAD Módulo responsable de contener y gestionar la carga científica del satélite. 29, 36, 37, 39–44, 61, 80, 84, 93

RISC Reduced Instruction Set Computer, estrategia de diseño de microprocesadores basada en la filosofía de mantener el conjunto de instrucciones relativamente reducido. 21

RSSI Received signal strength indication, medición de la potencia de una señal de radio. 37, 38, 45

RX Registro de desplazamiento para la recepción de datos a través de SPI en el MSP430. 65

RXBUF Buffer de recepción de datos a través de SPI en el MSP430. 65

RXBUF Cola de recepción de datos a través de SPI en el ARM. 66

SCLK Clock. Línea de control de SPI. Transmite la señal del reloj que marca la sincronización. Con cada pulso del reloj se lee o se envía un bit. 65, 78

SEL Single Event Latch-up, Estado anormal de alta corriente en un dispositivo electrónico causado por el pasaje de una partícula energética a través de regiones sensibles de la estructura del dispositivo, causando una perdida de funcionalidad. 28

SEU Single Event Upset, Cambio de estado provocado por iones o radiación electromagnética golpeando un nodo sensible de un dispositivo electrónico como un microcontrolador, causando el cambio del valor lógico de un bit o una compuerta. 28, 29

SGP4 Simplified Perturbations Models 4, Modelo matemático utilizado para calcular la posición y velocidad de satélites en orbita teniendo en cuenta las perturbaciones gravitacionales de los distintos cuerpos celestes. 18, 21, 23, 62, 63, 74

Sistema Orbital Sistema de referencia que consta de un eje que apunta hacia el centro de la Tierra, un eje que es tangencial a la órbita (paralelo a la velocidad) y un tercero que es el producto vectorial de los dos anteriores. 62, 63, 74

SPI Serial Peripheral Interface, bus de comunicaciones en serie con el cual se comunican el procesador principal y el coprocesador del módulo ADCS. 21, 24, 53, 55, 58, 61, 62, 64, 65, 67–69, 73, 76–81, 87, 90, 93

spimp SPI Message Protocol, Protocolo de comunicaciones sobre SPI mediante el cual se comunican los procesadores del módulo ADCS. 59, 62, 64, 72, 76, 77, 80, 81, 90, 95

spirpc SPI Remote Procedure Call, Interfaz de comunicación sobre spimp que permite el envío de mensajes entre los procesadores del módulo ADCS. 59, 64, 72, 76, 77, 79, 80, 90

SPIx_DR Registro de acceso a las colas de recepción y envío de datos por SPI en el ARM. 66

SS Slave Select. Línea de control de SPI. Selecciona el Esclavo a activar. 65

SSTV Slow Scan Television, Método usado en AntelSat para transmitir imágenes a Tierra a través de ondas de radio. 37

STM32F3 Chip contenedor del microcontrolador ARM Cortex M4 utilizado en el ADCS. 24, 62, 63

stub Pieza de código destinada a simular el comportamiento de un módulo de software durante el proceso de realización de pruebas. 40, 47–49, 60, 77, 84

TEME True Ecuator, Mean Equinox, Variación del sistema ECI utilizado en los TLE del NORAD. 18

TLE Two Lines Elements, Formato de datos utilizado para representar un conjunto de elementos orbitales que describen la orbita de un satélite. Es utilizado por el NORAD y la NASA. 18, 21, 22, 59, 61–63, 73, 74, 78, 90

TX Registro de desplazamiento para el envío de datos a través de SPI en el MSP430. 65

TXBUF Buffer de envío de datos a través de SPI en el MSP430. 65

TXBUF Cola de envío de datos a través de SPI en el ARM. 66

UART Universal Asynchronous Receiver/Transmitter, Componente de hardware que traduce información entre un medio paralelo y un medio serial. 21

watchdog Temporizador electrónico utilizado para detectar y recuperarse de fallas en un sistema electrónico. 21, 30–32