CEA Comité Español de Automática
XLVII JORNADAS DE AUTOMÁTICA CÓRDOBA 2026
UNIVERSIDADE DA CORUÑA

Jornadas de Automática

Diseño e implementación del software de a bordo para un CubeSat basado en arquitectura particionada

Lucía Alba[a], Andrea Nieto[a], Luis Ortiz[a], Marc Fontalba[a], Patricia Balbastre[a,*], Ana Guasque[a], José Simó[a], Alfons Crespo[a]

[a]*Instituto de Automática e Informática Industrial (ai2), Universitat Politècnica de València*

To cite this article: Alba, L., Nieto, A., Ortiz, L., Fontalba, M., Balbastre P., Guasque, A., Simó, J., Crespo, A. 2026. Design and implementation of onboard software for a CubeSat based on a partitioned architecture. Jornadas de Automática, 47. https://doi.org/10.17979/ja-cea.2026.47.13717

Resumen

Este trabajo presenta el diseño e implementación del software de a bordo (OBSW) de un nanosatélite CubeSat 1.5U educativo basado en una arquitectura IMA (*Integrated Modular Avionics*). El sistema se ejecuta sobre el hipervisor de tiempo real XtratuM XNG en una plataforma Zybo Z7 (Zynq-7000), garantizando el aislamiento temporal y espacial entre particiones conforme al estándar ARINC 653 y alineado con el estándar espacial ECSS. El software de vuelo se organiza en cuatro particiones funcionales: PCOM, responsable del enlace LoRa con la estación de tierra; OBCS, encargada de la coordinación y gestión de modos operacionales del satélite; PGS, que centraliza la adquisición, validación y distribución de datos de sensores y FDIR, que implementa la detección, clasificación y recuperación de fallos. La comunicación inter-partición se realiza exclusivamente mediante canales *sampling* y *queuing* conformes a ARINC 653. Los resultados obtenidos demuestran la viabilidad de aplicar arquitecturas IMA en nanosatélites educativos, aportando modularidad, determinismo temporal y mayor robustez frente a fallos.

Palabras clave: Arquitectura de computadores embebidos, Sistemas ciberfísicos, Planificación de sistemas de tiempo real.

Design and implementation of onboard software for a CubeSat based on a partitioned architecture

Abstract

This paper presents the design and implementation of the On-Board Software (OBSW) for a 1.5U educational CubeSat nanosatellite based on an IMA (Integrated Modular Avionics) architecture. The system runs on the XtratuM XNG real-time hypervisor on a Zybo Z7 (Zynq-7000) platform, ensuring temporal and spatial isolation between partitions in accordance with the ARINC 653 standard and aligned with the ECSS space standard. The flight software is organized into four functional partitions: PCOM, responsible for the LoRa link with the ground station; OBCS, in charge of satellite coordination and operational mode management; PGS, which centralizes sensor data acquisition, validation and distribution; and FDIR, which implements fault detection, classification and recovery. Inter-partition communication is carried out exclusively through ARINC 653-compliant sampling and queuing channels. The results demonstrate the feasibility of applying IMA architectures in educational nanosatellites, providing modularity, temporal determinism and improved fault robustness.

Keywords: Embedded computer architectures, Cyber-Physical Systems, Real-time algorithms, scheduling, and programming.

1. Introducción

Los CubeSats son nanosatélites basados en un estándar modular de 10×10×10 cm por unidad (1U), que ha democratizado el acceso al espacio en entornos académicos y de investigación (NASA, 2026). Su número de lanzamientos ha crecido de forma sostenida, superando los 3200 satélites hasta 2026 (Kulu, 2014). Sin embargo, la tasa de éxito de las misiones sigue siendo inferior a la de los satélites convencionales, siendo el software de vuelo uno de los factores más determinantes.

*Autor para correspondencia: patricia@ai2.upv.es

El software de vuelo en CubeSats ha sido desarrollado tradicionalmente mediante arquitecturas monolíticas o débilmente estructuradas, lo que dificulta el mantenimiento, la reutilización y la escalabilidad. En la industria aeroespacial se han adoptado arquitecturas más avanzadas como IMA, que establece la modularidad y el aislamiento entre funciones críticas (Gaska et al., 2015).

En paralelo, estándares como ARINC 653 (Aeronautical Radio, Inc. (ARINC), 2006) y ECSS (European Cooperation for Space Standardization (ECSS), 2025) definen modelos de particionado y requisitos para sistemas críticos, permitiendo garantizar propiedades fundamentales como el aislamiento temporal y espacial entre aplicaciones. Este enfoque resulta especialmente relevante en sistemas espaciales, donde la fiabilidad y el comportamiento determinista son esenciales.

El presente trabajo aborda el diseño e implementación de cuatro particiones del software de vuelo de un CubeSat 1.5U educativo: las particiones PCOM, OBCS, PGS y FDIR. Estas cuatro particiones conforman el núcleo funcional del satélite y constituyen el trabajo presentado en este artículo.

2. Estado del arte

2.1. *Arquitecturas software en CubeSats*

La evolución de las arquitecturas software en sistemas espaciales ha seguido una transición desde enfoques monolíticos hacia arquitecturas modulares y, más recientemente, hacia el paradigma IMA. En los diseños monolíticos, todo el software de vuelo se ejecuta en un único contexto sin separación formal entre componentes: un fallo puede propagarse al sistema completo, y la reutilización se ve dificultada por el fuerte acoplamiento al hardware (Araguz et al., 2018).

Las arquitecturas modulares introducen una descomposición en componentes independientes con interfaces bien definidas. Sin embargo, frameworks consolidados como NASA cFS, KubOS o FreeRTOS no proporcionan de forma nativa mecanismos de aislamiento fuerte entre componentes (Quiros-Jimenez and d'Hemecourt, 2019; Eshaq et al., 2025).

El particionado temporal y espacial entre aplicaciones puede ser implementado mediante técnicas de virtualización con hipervisor. En este modelo, cada función crítica se ejecuta en una partición con recursos estrictamente aislados, garantizando determinismo y contención de fallos. Este enfoque está siendo progresivamente adoptado en el ámbito CubeSat (Ciacchella et al., 2023).

2.2. *Hipervisor XtratuM y LithOS*

XtratuM (Masmano et al., 2009) es un hipervisor de tipo 1 para sistemas empotrados en tiempo real que permite la ejecución concurrente de múltiples particiones sobre una misma plataforma hardware. Su diseño se basa en una arquitectura particionada, alineada con ARINC 653, proporcionando aislamiento temporal y espacial entre particiones.

XtratuM XNG es una evolución orientada a plataformas modernas, con soporte para arquitecturas ARM Cortex-A9 y cualificación conforme a ECSS. Permite ejecutar particiones sobre LithOS (Masmano et al., 2010), un RTOS que cumple el standard ARINC-653 específicamente desarrollado para ejecutarse sobre XtratuM XNG. Esta combinación proporciona una solución coherente para el desarrollo de software de vuelo modular, robusto y determinista.

2.3. *Gestión de sensores y FDIR*

La gestión de sensores en CubeSats no suele abordarse como un problema independiente. En la arquitectura del 3Cat-1 (Araguz et al., 2018), los sensores se gestionan mediante módulos dependientes de hardware sin centralizar el acceso a los buses ni establecer mecanismos sistemáticos de validación. Otros enfoques como PHI-Demo (Eshaq et al., 2025) tampoco establecen una capa común que garantice la coherencia de los datos antes de su distribución.

En cuanto a los sistemas FDIR, los enfoques existentes basados en *watch points*, modos seguros o reinicios supervisados por watchdog operan típicamente sin las garantías de aislamiento que ofrece un entorno particionado (Eshaq et al., 2025; Araguz et al., 2018).

3. Arquitectura del sistema

3.1. *Plataforma hardware*

La plataforma hardware empleada es la Zybo Z7-10, basada en el SoC Zynq-7000 de Xilinx, que integra un procesador ARM Cortex-A9 de doble núcleo a 650 MHz y lógica programable FPGA. Dispone de 1 GB de memoria RAM DDR3. Esta arquitectura heterogénea permite implementar controladores hardware personalizados y adaptar los buses de comunicación al entorno particionado, además de soportar la ejecución de XtratuM XNG.

Para el subsistema de sensores se emplean una IMU MPU-9250 (acelerómetro, giroscopio y magnetómetro triaxiales, interfaz I2C) y un módulo GPS NEO-6M (interfaz UART, tramas NMEA-0183). Para las comunicaciones con tierra se usa un transceptor LoRa SX1278 (433 MHz, interfaz SPI). El diseño hardware se realizó mediante Vivado Design Suite, habilitando los bloques IP AXI IIC y AXI Uartlite para los sensores, y el controlador SPI1 del PS para el módulo LoRa.

3.2. *Modelo de particionado*

El sistema de vuelo se organiza en torno al concepto de partición. Cada partición constituye una unidad de ejecución aislada, con su propio espacio de memoria y ventana de tiempo asignada de forma estática en el plan de scheduling. Este aislamiento fuerte diferencia esta arquitectura de los enfoques más habituales en CubeSats.

La Figura 1 muestra el modelo de particiones del sistema con sus interacciones principales.

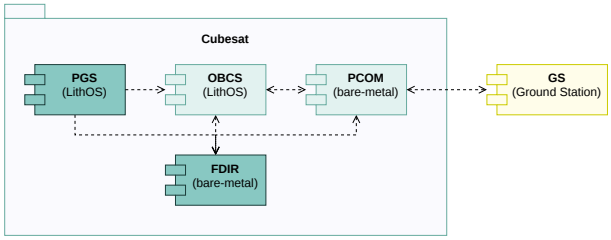

Figura 1: Modelo de particiones del sistema CubeSat.

El sistema cuenta con cuatro particiones principales y un componente externo:

- **PGS** (*Partición de Gestión de Sensores*): concentra el acceso exclusivo a los buses físicos de sensorización (I2C y UART). A través de ellos se realiza la lectura de acelerómetro, giróscopo y magnetómetro de la IMU MPU-9250 (I2C) y la posisión, velocidad y tiempo del GPS NEO-6M (UART). Ejecuta sobre LithOS.
- **FDIR** (*Fault Detection, Isolation and Recovery*): supervisa el estado del sistema y aplica acciones de recuperación. Funciona en modo bare-metal sobre el hipervisor.
- **OBCS** (*On-Board Computer System*): núcleo de coordinación y control del satélite. Ejecuta sobre LithOS.
- **PCOM** (*Partición de Comunicaciones*): gestiona el radioenlace LoRa con la estación de tierra. Funciona en modo bare-metal.
- **GS** (*Ground Station*): segmento de tierra, externo al satélite.

4. Particiones del sistema

4.1. Partición PCOM

La partición de comunicaciones (PCOM) gestiona el enlace bidireccional entre el satélite y la estación de tierra mediante tecnología LoRa en la banda de 433 MHz. Opera en modo bare-metal para minimizar la latencia. Su arquitectura interna se organiza en tres capas: acceso a hardware (drivers SPI y GPIO), protocolo de comunicaciones (PCOM protocol y adaptador LoRa-PCOM) y capa de control.

El protocolo PCOM distingue entre *mensaje* (unidad lógica de información) y *paquete* (unidad física de transmisión LoRa). La cabecera de mensaje (12 bytes) incluye identificador, tipo, marca temporal y tamaño de payload. La cabecera de paquete (11 bytes) añade CRC-16, número de secuencia y número total de paquetes para soporte de fragmentación. Cuando un mensaje supera los 217 bytes de payload útil por paquete, se fragmenta automáticamente en $N_p = \lceil T_m/217 \rceil$ paquetes.

Los mecanismos de integridad incluyen CRC-16 CCITT por paquete, confirmación ACK por paquete recibido y sistema de reintentos configurable. Si no se recibe ninguna comunicación durante más de 300 segundos, el sistema notifica al OBCS para activar el modo seguro.

La partición se organiza en torno a dos máquinas de estados finitos (FSM) que se ejecutan secuencialmente en cada slot: la FSM de transmisión, que fragmenta y envía los mensajes de telemetría leídos desde el queuing port de entrada, y la FSM de recepción, dirigida por eventos hardware sobre la línea DIO0 del transceptor. La planificación cíclica establecida en el fichero de configuración de XtratuM garantiza que la partición no supera su slot asignado.

4.2. Partición OBCS

La partición OBCS constituye el núcleo de coordinación del satélite. Opera sobre LithOS y gestiona los modos operacionales mediante una máquina de estados con cuatro modos: Inicialización, Nominal, Seguro y Emergencia. Se organiza en tres procesos con prioridades diferenciadas:

- **Process0** (prioridad máxima): recibe telecomandos desde PCOM, valida su estructura y los despacha al handler correspondiente según el tipo de mensaje.
- **Process1** (prioridad media): lee periódicamente los datos de sensores desde los sampling ports de PGS y los reenvía a PCOM para transmisión.
- **Process2** (prioridad mínima): monitoriza las condiciones del sistema y activa transiciones de modo cuando se superan los umbrales definidos.

Los tipos de telecomando soportados incluyen REBOOT, SET_TELEMETRY_RATE, SET_LORA_POWER, CHANGE_MODE, SET_PARAMETER, REQUEST_TELEMETRY y PING. La telemetría se compone con datos de sensores (PGS), estado del sistema y modo operacional actual, enviándose a PCOM mediante queuing port.

4.3. Partición PGS

La Partición de Gestión de Sensores (PGS) se ha diseñado siguiendo un principio de separación de responsabilidades, donde cada proceso asume una función específica dentro del ciclo de adquisición y publicación de datos. Al actuar como único punto de acceso al hardware de sensorización, elimina la posibilidad de accesos concurrentes desde otras particiones y garantiza que los datos que circulan por el sistema han sido validados previamente.

La PGS estructura su funcionamiento en seis procesos principales:

- **PROC_INIT**: inicializa y verifica todos los dispositivos hardware. Para cada sensor crea una instancia de clase a partir de la tabla de configuración `pgs_config`, configurando el bus, leyendo el registro de identidad (WHO_AM_I para MPU-9250) y aplicando la configuración inicial.
- **PROC_IMU_READ**: lee los registros hardware de la IMU en modo burst y actualiza el bloque compartido `pgs_sensor_block`.
- **PROC_GPS_READ**: lee el buffer UART del módulo GPS y actualiza el bloque de datos con la información de posición en formato NMEA.
- **PROC_VALIDATE**: verifica que los valores obtenidos de los sensores se encuentren dentro de los rangos operacionales definidos en la tabla estática `pgs_range_config`. Si un valor supera el umbral de errores consecutivos configurado (MAX_CONSEC_ERROR), genera un evento y lo almacena en `pgs_error_queue`.
- **PROC_PUBLISH**: publica los datos validados en los canales de *sampling* correspondientes, añadiendo el timestamp obtenido inmediatamente tras la lectura.
- **PROC_ERROR_RPT**: extrae los eventos de la cola de errores interna y los envía a la partición FDIR mediante el canal queuing QC_PGS_TO_FDIR.

Los drivers (IMU, I2C, GPS, UART) abstraen el acceso a los periféricos hardware siguiendo un modelo orientado a objetos en C, modelando cada protocolo y sensor como una clase mediante structs y funciones.

4.4. Partición FDIR

La Partición de Detección y Recuperación de Fallos (FDIR) opera como una capa de supervisión completamente desacoplada del resto del sistema. Al igual que la PGS, aplica un principio de separación de responsabilidades distribuyendo las etapas del proceso de gestión de fallos en procesos independientes: recepción, clasificación y recuperación.

La FDIR opera sobre bare-metal directamente sobre el hipervisor, gestionando de forma explícita la coordinación entre procesos mediante estructuras de datos compartidas. Se compone de cinco procesos:

- **PROC_INIT**: carga las tablas de configuración de watchdog, clasificación y recuperación antes de la operación nominal.
- **PROC_WDG_CHECK**: supervisa periódicamente el estado de las particiones mediante un mecanismo de watchdog. Cada partición renueva su señal de vida mediante un canal *sampling* dedicado (SC_*_TO_FDIR_WDG). Si una partición expira su timeout, se genera un evento de fallo con la acción correctiva correspondiente: reinicio de la partición (hasta 3 intentos) para particiones no críticas, o reset completo del sistema para particiones críticas (OBCS, PCOM o PGS).
- **PROC_EVT_RECV**: recibe los eventos de error enviados por PGS a través del canal QC_PGS_TO_FDIR y los encola en `fdir_evt_queue` para su clasificación.
- **PROC_EVT_CLASSIF**: extrae los eventos de la cola y determina su severidad y categoría consultando la tabla estática `fdir_classif_table`.
- **PROC_RECOVERY**: aplica la acción de recuperación asociada a cada evento clasificado, consultando `fdir_recovery_table` para determinar la respuesta adecuada en función del tipo y severidad del fallo.

5. Comunicación inter-partición

Toda la comunicación entre particiones se realiza exclusivamente mediante puertos y canales ARINC 653 proporcionados por XtratuM, sin ningún mecanismo de memoria compartida ni acceso directo entre espacios de ejecución. Este diseño garantiza el desacoplamiento entre particiones y mantiene el aislamiento fuerte que caracteriza a la arquitectura IMA. Se definen dos tipos de canal, cada uno adecuado a una semántica de comunicación diferente.

5.1. Canales sampling

Los canales *sampling* (SC) están concebidos para la transmisión de datos cuyo único valor relevante es el más reciente. Cada escritura sobreescribe el dato anterior, por lo que el receptor lee siempre la última muestra disponible sin necesidad de sincronizarse con el emisor. Este comportamiento los hace idóneos para señales de tipo sensor que se actualizan periódicamente, así como para variables de estado que otras particiones consultan en cualquier momento dentro de su ventana de ejecución.

En el sistema se utilizan canales SC para los siguientes flujos:

- **SC_PGS_IMU**: publica los valores de la IMU desde PGS hacia OBCS.
- **SC_PGS_GPS**: publica los valores del GPS desde PGS hacia OBCS.
- **SC_PCOM_LINK_STATUS**: informa a FDIR del estado del enlace LoRa (activo, perdido, en reconexión).
- **SC_*_TO_FDIR_WDG**: cada partición (PGS, OBCS, PCOM) renueva periódicamente su señal de vida hacia FDIR a través de su canal *watchdog* dedicado. FDIR comprueba que cada señal se renueva dentro del timeout configurado; si una partición deja de hacerlo, se genera un evento de fallo.

5.2. Canales queuing

Los canales *queuing* (QC) implementan una cola FIFO con capacidad limitada y garantía de entrega. Cada mensaje es consumido exactamente una vez y en el orden en que fue generado. Son adecuados para el intercambio de comandos, alarmas o eventos de error, donde perder un mensaje o alterar su orden podría comprometer el correcto funcionamiento del sistema. Si el receptor no consume los mensajes a tiempo existe riesgo de desbordamiento, lo que obliga a dimensionar las colas para el peor caso.

Los canales QC definidos en el sistema son:

- **QC_PGS_TO_FDIR**: PGS envía a FDIR los eventos de error detectados durante la adquisición (valores fuera de rango, fallos de bus, timeouts de sensor). FDIR los encola en `fdir_evt_queue` para su posterior clasificación y recuperación.
- **QC_OBCS_TO_PCOM**: OBCS envía a PCOM los mensajes de telemetría compuestos (datos de sensores, modo operacional, estado del sistema) para su transmisión hacia la estación de tierra.
- **QC_PCOM_TO_OBCS**: PCOM entrega a OBCS los telecomandos recibidos desde tierra, ya verificados mediante CRC y reensamblados si llegaron fragmentados.
- **QC_FDIR_TO_PCOM**: FDIR notifica a PCOM las alarmas y eventos críticos del sistema (fallos de partición, umbrales superados) para su retransmisión urgente a tierra.
- **QC_PCOM_TO_FDIR**: PCOM informa a FDIR de los errores de comunicación detectados (pérdida de enlace, CRC inválido, timeout de ACK).
- **QC_OBCS_TO_FDIR**: OBCS envía a FDIR eventos de sistema que requieren supervisión (transiciones de modo, anomalías detectadas en el procesamiento de telecomandos).

Tabla 1: Canales de comunicación inter-partición del sistema

Canal	Tipo	Emisor → Receptor	Contenido
SC_PGS_IMU	SC	PGS → OBCS	Valores de acelerómetro, giroscopio y magnetómetro (IMU)
SC_PGS_GPS	SC	PGS → OBCS	Posición, velocidad y tiempo (GPS)
SC_PCOM_LINK_STATUS	SC	PCOM → FDIR	Estado del enlace LoRa (activo, perdido, en reconexión)
SC_*_TO_FDIR_WDG	SC	Todas → FDIR	Tick de watchdog
QC_PGS_TO_FDIR	QC	PGS → FDIR	Eventos de error de sensores (fuera de rango, fallos de bus)
QC_OBCS_TO_PCOM	QC	OBCS → PCOM	Mensajes de telemetría para transmisión a tierra
QC_PCOM_TO_OBCS	QC	PCOM → OBCS	Telecomandos recibidos desde la estación de tierra
QC_FDIR_TO_PCOM	QC	FDIR → PCOM	Alarmas y eventos críticos para retransmisión urgente
QC_PCOM_TO_FDIR	QC	PCOM → FDIR	Errores de enlace (CRC inválido, timeout de ACK)
QC_OBCS_TO_FDIR	QC	OBCS → FDIR	Eventos de sistema (transiciones de modo, anomalías)

La Tabla 1 recoge el conjunto completo de canales con su tipo, emisor, receptor y contenido.

6. Resultados y validación

El sistema fue validado sobre la plataforma hardware Zybo Z7-10. Las pruebas funcionales verificaron el correcto funcionamiento de cada módulo de forma aislada mediante módulos de simulación (*mocks*) de los sensores IMU y GPS. Las pruebas de integración verificaron la correcta interacción entre particiones y el cumplimiento del aislamiento proporcionado por XtratuM.

Para las particiones PCOM y OBCS se midió el WCET de los procedimientos críticos. La función `run_tx_step()` de PCOM, que constituye la operación más costosa, demostró caber con margen dentro del slot de 12 ms. El análisis temporal de la PGS confirmó que la suma de WCETs de todos sus procesos (INIT: 0,3 ms, IMU_READ: 1,5 ms, GPS_READ: 2,0 ms, VALIDATE: 1,2 ms, PUBLISH: 1,8 ms, ERROR_RPT: 0,8 ms) es inferior al slot asignado de 8 ms.

Se validó el enlace LoRa completo mediante pruebas de *downlink* (transmisión de telemetría desde el satélite a una estación Arduino Uno R4) y de *uplink* (recepción de telecomandos). Los mensajes de telemetría de hasta 92 bytes se enviaron en un único paquete con CRC-16 correcto; los comandos se recibieron, verificaron y despacharon al handler correspondiente en OBCS. Las pruebas de integración confirmaron el correcto flujo de datos desde PGS hasta PCOM pasando por OBCS, y el correcto funcionamiento del mecanismo de watchdog de FDIR.

Los resultados demuestran la viabilidad de aplicar arquitecturas IMA en nanosatélites educativos, proporcionando:

- **Modularidad**: cada partición puede desarrollarse, validarse y actualizarse de forma independiente.
- **Determinismo temporal**: planificación cíclica estática con WCETs verificados en cada slot.
- **Contención de fallos**: el aislamiento espacial y temporal impide la propagación de errores entre particiones.
- **Gestión de fallos integrada**: la partición FDIR opera con las mismas garantías de aislamiento que el resto del sistema.

7. Conclusiones

Se ha presentado el diseño e implementación de una arquitectura software particionada para un nanosatélite CubeSat 1.5U educativo, organizada en cuatro particiones funcionales (PCOM, OBCS, PGS y FDIR) sobre el hipervisor XtratuM XNG. La arquitectura propuesta supera las limitaciones de los enfoques monolíticos y débilmente modulares predominantes en la literatura CubeSat, al incorporar aislamiento fuerte como base de ejecución.

Las principales contribuciones son: (1) la Partición de Gestión de Sensores como componente arquitectónico de primer nivel, que centraliza el acceso a los buses físicos y garantiza la validación de los datos antes de su distribución; (2) la integración de la partición FDIR dentro del entorno particionado, con mecanismos de watchdog, clasificación de eventos y acciones de recuperación; (3) las particiones PCOM y OBCS, que implementan respectivamente el enlace LoRa con la estación de tierra y la coordinación operativa del satélite. Todas ellas se comunican exclusivamente mediante canales ARINC 653 y se ejecutan dentro de ventanas temporales con WCETs verificados.

Como trabajo futuro se plantea ampliar el sistema con particiones adicionales (PWR, ADCS, SEC, Payload), completar el análisis temporal formal con validación de WCETs y márgenes de guardia, y avanzar hacia una validación en condiciones más próximas al entorno operativo real.

Agradecimientos

Esta publicación es parte del proyecto de I+D+i PID2024-155230OB-C42, financiado por MCIN/ AEI/10.13039/501100011033.

Referencias

Aeronautical Radio, Inc. (ARINC), Mar. 2006. Avionics application software standard interface, Part 1 – Required Services. Tech. rep., Aeronautical Radio, Inc., Annapolis, Maryland, USA.

Araguz, C., Marí, M., Bou-Balust, E., Alarcon, E., Selva, D., 2018. Design guidelines for general-purpose payload-oriented nanosatellite software architectures. Journal of Aerospace Information Systems 15 (3), 107–119.

Ciacchella, G., Mohammad, A., Zurawski, S., 2023. An open-source method for model-based development of embedded systems: Experience report from a CubeSat student project. In: 74th International Astronautical Congress (IAC 2023).

Eshaq, M., Zitouni, M. S., Atalla, S., Al-Mansoori, S., Macdonald, M., 2025. CubeSat flight software: Insights and a case study. Journal of Spacecraft and Rockets 62 (4), 1328–1345.

European Cooperation for Space Standardization (ECSS), Apr. 2025. Space engineering – software, ECSS-E-ST-40C Rev.1. Tech. rep., ECSS, Noordwijk, The Netherlands.

Gaska, T., Watkin, C. J., Chen, Y., Sep. 2015. Integrated modular avionics — past, present, and future. IEEE Aerospace and Electronic Systems Magazine 30 (9), 12–23.
DOI: 10.1109/MAES.2015.7321122

Kulu, E., 2014. Nanosats database. https://www.nanosats.eu, accedido: 24-mar-2026.

Masmano, M., Ripoll, I., Crespo, A., Metge, J.-J., Sep. 2009. XtratuM: A hypervisor for safety critical embedded systems. In: Proceedings of the 11th Real-Time Linux Workshop. Dresden, Germany.

Masmano, M., Valiente, Y., Balbastre, P., Ripoll, I., Crespo, A., 2010. ARINC-653 APEX based on XtratuM. In: Jornadas de Tiempo Real (JTR). Universidad Politécnica de Valencia.

NASA, 2026. What are smallsats and cubesats? https://www.nasa.gov/what-are-smallsats-and-cubesats/, accedido: 20-mar-2026.

Quiros-Jimenez, O. D., d'Hemecourt, D., 2019. Development of a flight software framework for student CubeSat missions. Tecnología en marcha 32 (6), 180–197.