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

Jornadas de Automática

LithOS Bare-Metal: an unusual path for an RTOS

Miguel Gotor-Ramos[a,*], Carlos Cuesta-Martínez[a], Manuel Muñoz-Alcobendas[a], Miguel Masmano-Tello[a]

[a] *FENTISS, Ciudad Politécnica de la Innovación, Edificio 9B Office 1.67 Camino de Vera s/n, 46022 Valencia España*

To cite this article: Gotor-Ramos, M., Cuesta-Martínez, C., Muñoz-Alcobendas, M., Masmano-Tello, M. 2026. LithOS Bare-Metal: an unusual path for an RTOS. Jornadas de Automática, 47. https://doi.org/10.17979/ja-cea.2026.47.13730

Abstract

This paper presents LithOS Bare-Metal (LithOS-BM), a Real-Time Operating System (RTOS) partially compliant with the ARINC 653 APEX standard, designed for small embedded systems where partitioning or virtualization is unnecessary or infeasible, and rapid response times are critical.

Integrated Modular Avionics consolidates software functions into shared computing resources, reducing hardware footprint, weight, and power consumption. ARINC 653 defines guidelines for Time and Space Partitioning (TSP), specifying a separation kernel and an APEX API for application software.

Originally, LithOS was developed as a guest RTOS for the XtratuM Next Generation hypervisor, which enforces strict TSP within a virtualized environment. However, certain embedded applications do not require complex hypervisor-managed partitioning, particularly when strict timing constraints or hardware limitations prevent virtualization.

LithOS-BM addresses this by replacing the hypervisor layer with a custom Board Support Package that interfaces directly with the hardware. This approach maintains ARINC 653 compliance, preemptive task scheduling, and temporal isolation while fully controlling the underlying hardware platform as a standalone RTOS.

Keywords: RTOS, Bare-Metal, Embedded systems, Hypervisor, ARINC 653, Time and Space Partitioning, Integrated Modular Avionics, ARM.

1. Introduction

1.1. IMA, TSP, and ARINC 653

In embedded systems, particularly avionics systems, **Integrated Modular Avionics** (IMA) is an architectural approach where multiple software modules are consolidated onto shared computing resources. Instead of running each function on dedicated hardware, IMA allows several applications to share the same platform, reducing the number of hardware units and weight, space, and power consumption.

The ARINC 653 standard is widely adopted in the aerospace domain to implement the **Time and Space Partitioning** (TSP) concept for systems based on the IMA architecture. To guarantee isolation, ARINC 653 defines guidelines for a **separation (or partitioning) kernel** in charge of the internal partition management mechanisms, and a standardized Application Executive (APEX) interface for Application software (AppSw).(Aeronautical Radio, Inc., 2010).

The specification is divided into three parts:

1. **Part 1**: Required Services
2. **Part 2**: Extended Services
3. **Part 3**: Conformity Test Specification

Although this standard does not prescribe how a hypervisor must operate (internal hypervisor operations are not explicitly described) its model closely aligns with the functionality typically provided by a hypervisor.

1.2. XtratuM Next Generation (XNG)

XNG is an embedded, ECSS category B qualified, Type I (bare-metal) hypervisor designed for safety-critical, real-time applications (Masmano et al., 2009). It enforces TSP

*Autor para correspondencia: mgotor@fentiss.com

to prevent error propagation between applications sharing the hardware by strictly isolating memory spaces and scheduling partition execution through deterministic CPU time slots based on a predefined cyclical schedule.

1.3. LithOS as XNG guest RTOS

While the XNG API and its internal operations conceptually and syntactically resemble the ARINC 653 standard, XNG is not natively an ARINC 653-compliant system. However, it supports the LithOS Real-Time Operating System (RTOS), which provides a runtime environment conformant to ARINC 653 Part 1 Version 3 (2010) (Aeronautical Radio, Inc., 2010).

ARINC 653-P1 § 3.1: *Service Request Categories* classifies APEX services into seven groups:

1. **Partition** Management services to control the transitions between states of an ARINC 653 partition (`NORMAL, IDLE, WARM START, COLD START`).
2. **Process** Management including configuration, creation, scheduling and execution to control the states of processes (`DORMANT, WAITING, READY, RUNNING`).
3. **Time** Management (periodic processes, timed waits, etc.)
4. **Memory** Management (MMU/MPU configuration for memory partitioning)
5. **Interpartition** Communication: Sampling and queuing ports
6. **Intrapartition** Communication: Buffers, blackboards, semaphores, and events for inter-process communication and synchronization.
7. **Health** Monitoring: Fault and failure detection, isolation and recovery thanks to a dedicated Health Monitoring (HM) module

In this layered architecture, the RTOS and hypervisor divide these responsibilities. LithOS implements *intra-partition* resource management, covering process and partition state control, software timers, intra-partition communication, and health monitoring. It also provides Part 2 extended services (Aeronautical Radio, Inc., 2012) and custom Non-Portable (NP) services for hardware interrupt (IRQ) management.

Conversely, *inter-partition* operations—such as memory partitioning (MMU/MPU configuration), inter-partition queuing, and system-level management—are beyond the reach of the RTOS and rely entirely on XNG, which assumes the role of the partitioning kernel.

Because LithOS delegates underlying TSP mechanisms to the hypervisor, it does not fulfill the entirety of ARINC 653 independently; rather, the layered RTOS/Hypervisor stack nominally constitutes a *single* ARINC 653 partition. Together, LithOS and XNG create a fully compliant, mature execution environment for industrial applications (Masmano et al., 2010), demonstrating their reliability in stringent aerospace missions such as SVOM (Space-based multi-band astronomical Variable Objects Monitor) or JUICE (Jupiter Icy Moons Explorer)(Losa et al., 2024).

2. The need for a LithOS bare-metal version

LithOS offers several critical benefits for resource-constrained, monocore embedded systems:

- It is a *safe-by-design* RTOS, since it implements the ARINC 653 standard.
- It provides a dedicated module for detection and mitigation of failures: the Health Monitor.
- LithOS is lightweight and does not include dispensable built-in features like a filesystem or network stack.
- It allows statically configuring partition resources through a standard configuration file. This ensures the memory allocated is tightly adjusted to the AppSw's actual needs, minimizing the waste of typically scarce resources.

However, some applications may not require a virtualized or partitioned environment and could instead benefit from a standalone LithOS RTOS that provides task prioritization and timing management without the overhead of a hypervisor layer. Partitioned architectures may also be unsuitable for certain real-time workloads with strict timing constraints, where hypervisor-induced context switching becomes non-negligible.

In other cases, virtualization is unnecessary because direct hardware access is sufficient. This limitation may also stem from hardware constraints: some microcontrollers lack a Memory Management Unit (MMU) or Memory Protection Unit (MPU), or do not implement distinct privilege levels, making paravirtualization infeasible.

Nevertheless, there remains a certain need to guarantee some level of compliance with the demanding standards mandated by the aerospace sector. To address these requirements, a standalone version of the LithOS operating system has been developed. This version is capable of running as a bare-metal application in a hypervisor-less environment while still fulfilling the majority of the requirements defined by the ARINC 653 standard, Part 1. This means that LithOS-BM offers the flexibility to operate without a hypervisor while maintaining the same interface (API) as its hypervisor-based version. This consistency allows easy software prototyping and development, enabling engineers to build and test system components in a standalone environment before integrating them into the final partitioned system.

Developing this new version of the LithOS software, called LithOS Bare-Metal (BM), required detaching LithOS from XNG to enable operation in a hypervisor-less scenario. Specifically, the adaptation for BM execution has targeted the ARM Cortex-R5F processor. The main development work has consisted of implementing a specialized Board Support Package (BSP) designed to interface with the aforementioned microprocessor architecture, acting as the intermediary layer for interaction between the LithOS and the hardware components of the board, replacing XNG's virtualization layer.

In its baseline version, the LithOS software is strongly dependent on the underlying hypervisor. LithOS delegates part of the functionality provided by the APEX interface services by implementing BSP wrapper functions which ultimately interact with the XNG hypercalls interface. Additionally, it entrusts to XNG those operations related to the board initialization and setup necessary to enable execution of user application code.

3. Background and hardware platform selection

The development of LithOS-BM was conducted within the context of the METAMORPHA H2020 project[1]. This EU project involves the development of an advanced digital laser machining platform that integrates various technologies contributed by different project partners (European Commission, 2022).

In this context, the LithOS-BM kernel has been developed in parallel to the application to meet project-specific requirements, providing a runtime support layer for a set of real-time tasks with hard deadlines. Specifically, it manages TCP/IP communication by integrating the lwIP stack and Xilinx XEmacPs driver to interface directly with the KR260 board's MAC and PHY hardware. The RTOS also coordinates with FPGA logic to acquire raw DAQ sensor data via shared RAM buffers. This data undergoes real-time processing such as feature extraction and Machine Learning (ML) inference, using ARM CMSIS DSP and custom math libraries. Finally, the kernel's Health Monitor (HM) performs sensor diagnostics and mitigates hardware or software failures.

Within this framework, the objective is not only to support resource-constrained systems, but also to validate the RTOS on representative, industrial-grade hardware platforms.

Although LithOS-BM is intended for deployment in constrained embedded systems, the selected development platform is the Kria KR260 Robotics Starter Kit, based on the Xilinx Zynq UltraScale+ EV Multi-Processor System-on-Chip (MPSoC). This choice reflects practical considerations in the context of the project and prioritizes documented, exploitable, mature technology for industrial application rather than the target deployment footprint.

It is important to distinguish between the hardware layers involved. The KR260 is a development board hosting the Zynq UltraScale+ MPSoC, which in turn integrates multiple processing units, including the Cortex-A53 Application Processing Unit (APU) and the Cortex-R5F Real-Time Processing Unit (RPU). While the overall platform is relatively powerful, LithOS-BM specifically targets the **Cortex-R5F cores within the RPU**, which are representative of deterministic, resource-constrained real-time processors, unlike Cortex-M based microcontrollers where such determinism is not guaranteed.

The dual-core Cortex-R5F RPU was selected over the Cortex-A53 APU because ARM Real-Time (R-profile) processors prioritize deterministic execution and low-latency responses over raw throughput. The R-profile architecture uses a Memory Protection Unit (MPU) instead of a Memory Management Unit (MMU). This lack of address translation removes Translation Lookaside Buffer (TLB) misses, a common source of non-deterministic latency in A-profile processors. Coupled with a simplified cache hierarchy, this contributes to a more predictable timing behavior.

Furthermore, the RPU supports a redundant CPU configuration that executes the instructions simultaneously on both cores and continuously compares the results to detect discrepancies. This mechanism known as *lock-step* or safety mode provides hardware redundancy and fits the monocore design of LithOS-BM.

4. Software architecture and design

To maintain compatibility with its hypervisor-based predecessor while operating autonomously, LithOS-BM is structured around five major modular components: CORE, APEX, LIBC, LTCF, and the BSP. Since the APEX remains unmodified, user applications achieve full source-level compatibility across both LithOS versions (BM/XNG).

- **CORE**: Implements the fundamental kernel functionalities. It manages the creation and startup of ARINC 653 processes, treating them internally as independent execution threads. It features a fixed-priority preemptive scheduler utilizing a FIFO-within-priorities policy. Furthermore, the CORE unit manages the software timers (`kTimers`) to enforce process deadlines and implements the underlying communication channels for Inter-Process Communication (IPC).

- **APEX**: Provides the AppSw with an interface strictly compliant with ARINC 653 Part 1. It exposes services for process management, time management, and intra-partition communication utilizing blackboards, buffers, semaphores, and events. It also includes the Health Monitoring (HM) system for detecting and recovering from hardware and software faults.

- **LIBC**: A lightweight, customized subset of the standard C library. It provides basic string functions, mathematical integer operations, and services for managing linked lists. These linked lists are created and maintained by the kernel to manage active software timers, processes, and message queues.

- **LTCF** (LithOS Configuration File): To avoid the unpredictability and risks associated with dynamic memory allocation at runtime, LithOS relies exclusively on static memory allocation. The LTCF component parses configuration data at compile time, allocating a fixed memory pool for all necessary resources (processes, blackboards, buffers, etc.) based on the specific constraints of the AppSw.

- **BSP** Component: The architecture-specific layer that replaces the virtualization layer previously provided by XNG. It is responsible for all low-level hardware initializations, including setting up the CPU, FPU, L1 caches, UART, and the system clock. It also manages the Generic Interrupt Controller (GIC) and Triple Timer Counters (TTC) to support the RTOS's event-driven task scheduler and time-based services.

5. Implementation of the bare-metal BSP

This section is centred on the development of a specialized BSP to interface with the Cortex-R5F ARMv7R architecture. Being the intermediary layer, it enables direct interaction between the RTOS and the hardware components of the board, replacing the virtualization layer created by XNG. This task involves identifying and replacing both explicit functionalities provided by XNG to LithOS through services, and implicit

functionalities related to the board initialization and setup necessary to enable the proper execution of user application code, i.e., the AppSw component.

5.1. Explicit functionalities

Explicit functionalities refer to specific capabilities that LithOS actively delegates to XNG through explicit requests to services provided by the hypervisor to partitions, known in this context as *hypercalls*.

ARMv7-R processors implement two main privilege levels to support a basic security model: Privileged (PL1), used by the operating system/kernel, and Unprivileged (PL0), used for applications. XNG runs in a *privileged* processor mode (i.e., `System`, `Undefined`, `Supervisor`, `Abort`, `IRQ` or `FIQ` mode), while partitions execute in the *unprivileged* `User` mode. XNG's hypercall mechanism uses the Supervisor Call (`SVC`) assembly instruction to trigger a privilege escalation protocol. When a hypercall is made from the partition, the processor transitions to a privileged (`system`) mode, where the request is handled as an `SVC` exception and dispatched accordingly. As a result, both LithOS and AppSW execute at the same privilege level within a partition (user mode). In contrast, LithOS-BM runs directly at the highest privilege level, replacing the hypervisor. While moving application software to the remaining unprivileged level to enforce separation is theoretically possible, it falls outside the scope of this paper.

LithOS delegates part of the functionality provided by the APEX interface services by implementing BSP wrapper functions which ultimately interact with the XNG hypercall interface. These wrappers translate between the RTOS and the hypervisor time units and data types, such as return codes.

LithOS-BM eliminates these wrappers and the hypervisor's hypercall dispatcher by interacting directly with the hardware, flattening the function call tree for APEX interface services.

5.1.1. Hardware management

Real-time scheduling and event management directly depend on proper control over hardware interrupts and timers. LithOS-BM implements device drivers for the GICv1 and TTCs in the ZUS+ MPSoC.

- GICv1 Configuration: The Cortex-R5F utilizes a GICv1 architecture based on the ARM PL390. Because the processor operates in lock-step mode acting as a single logical CPU, the notion of core-private interrupts is discarded. The bare-metal BSP's interrupt driver is specifically configured to operate entirely on Shared Peripheral Interrupts (SPIs).
- TTC Implementation: XNG previously provided virtual interrupts (vIRQs) to manage partition scheduling and timekeeping. In LithOS-BM, this mechanism is replaced by control of the hardware timers, configured to operate in one-shot (match) mode at the hardware level.

5.1.2. Implementation of new services

To extend IRQ and TTC control to the application software, several non-portable ARINC-like services have been developed:

- A service to **configure the specified TTC timer** to operate in either periodic (interval) mode or one-shot (match) mode. It also disarms the timer when a non-positive load value is provided, eliminating the need for a separate service to reset the hardware timer.
- A service to **install an IRQ handler** for the selected hardware IRQ, linking a custom C function to a specific interrupt source.
- A service to **unmask** (enable) the selected hardware IRQ in the GIC register.
- A service to **mask** (disable) the selected hardware IRQ in the GIC register
- Additionally, several services were introduced to provide **fine-grained control** of transitions to specific states within the IRQ state machine (active, pending).

To maintain API consistency, these services are designed as *ARINC-like* extensions. They use standard APEX base types and append the `_NP` (Non-Portable) suffix. Additionally, the functions adhere to ARINC 653 C naming and parameter-passing conventions.

5.2. Implicit functionalities

These are functionalities or configurations that LithOS expects to be in place but does not actively request or interact with. Specifically, it assumes that certain devices are already configured and operational. These startup operations are implicitly dependent on XNG, which prepares a virtualized execution environment ready for use. Given this dependency, it is important to identify any additional operations performed by XNG that might not be explicitly reflected in the LithOS source code. For instance, this hardware architecture lacks hardware-assisted virtualization support. Consequently, XNG for the ARMv7R architecture employs Para-virtualization techniques to implement the partitioned system, including a mechanism for invoking system instructions from partition mode using the UDF (Undefined) ARM assembly instruction. Similar to the SVC instruction, UDF accepts an additional parameter (the ID) to identify the requested operation.

However, since LithOS-BM has full access to the CPU, including its special registers and coprocessors, assembly code has been modified to directly manipulate the processor state using the corresponding ARM specialized instructions, instead of their para-virtualized counterparts.

5.2.1. Exception handling and the software trampoline

LithOS-BM requires the ability to handle asynchronous events (interrupts) to manage periodic timer events that trigger a context switch, and to manage errors (diagnose and recover from failures).

Due to the para-virtualized architecture, when an exception occurs while LithOS is running on top of XNG, three exception vector tables are considered. The processor jumps to: ARM's exception vector, then XNG's exception vector, and finally LithOS's exception vector.

The bare-metal porting eliminates XNG's intermediate table and requires bridging the gap between the ARMv7 default exception vector and the one implemented by the BSP

component in LithOS-BM. This is achieved using a software trampoline mechanism that injects specific branch assembly instructions directly into the memory addresses corresponding to the ARM default exception vector entries. This setup dynamically redirects the execution flow to the relocatable LithOS-BM executable image using an indirect branch. It allows a flexible deployment since the system entry point is not tied to a specific memory address.

5.2.2. Partition scheduling

In the original implementation, XNG sends a Virtual Interrupt Request (vIRQ) to the LithOS partition to signal the partition "Release Point", indicating that its time slot has started. This release point marks the specific moment when the LithOS partition (or any other partition) is released from the idling state and begins its timed execution. To prevent reducing its time capacity, periodic processes become eligible at the beginning of the next slot, in conformity with the ARINC 653-P1 specification.

However, in the proposed environment, the AppSw partition occupies all available time ($PartitionPeriod = PartitionDuration = MajorFrame$), rendering the concept of partition period and duration irrelevant. This setup enforces a "single-partition schedule" on the task group, required for continuous process execution. This arrangement represents the simplest form of partition management, featuring a single active "partition" without idle time and eliminating the need for any kind of scheduler intervention.

6. Software Validation and performance results

To assure compliance with the ARINC 653 standard, LithOS-BM underwent a comprehensive validation campaign. The testing framework utilized a host/target approach, where the test suite was compiled and managed on an x86-64 GNU/Linux host PC and deployed to the Zynq UltraScale+ MPSoC target via the Xilinx System Debugger (XSDB).

Tests relying on inter-partition communication and hypervisor system management were inherently incompatible with the standalone nature of LithOS-BM, so test cases validating the sampling and queuing port ARINC 653 services were filtered out.

Finally, the kernel was tested using a subset of the standard ARINC653 validation suite, plus proprietary performance, stress, and robustness tests. Of the 402 original validation tests, 314 were applicable to the single-application bare-metal context, and all 314 executed successfully, achieving a 100% pass rate.

Additionally, the validation campaign included benchmarking tests for APEX services and internal kernel operations to evaluate kernel performance and ensure that LithOS-BM remains within real-time margins. Results are provided in Table 1, which compares the profiling of LithOS-BM against a baseline execution of LithOS for XNG on the ZUS+ platform on the Cortex-R5F core running at approximately 500 MHz with both instruction and data caches enabled. It provides hardware interrupt latency (via the Rhealstone benchmark methodology), resource shuffle times (the context switch overhead when releasing locked events, semaphores, buffers, or blackboards), the overhead of active Floating-Point Unit (FPU) context switching, and the execution overhead for the main ARINC 653 services.

A consistent performance improvement is observed across all services, with mean execution times reduced by several microseconds compared to the hypervisor-based baseline.

Because LithOS is non-reentrant, it enforces critical sections around every APEX service call by masking interrupts. This design imposes a baseline overhead on all service invocations, which explains why performance differences are observed across all services, including those not directly dependent on hypercalls.

However, it is necessary to contextualize this reduction by examining the underlying measurement function. The benchmarking macro calculates the execution cost of a given service by timestamping its start and completion using the `GET_TIME` APEX service. To calculate the execution time of the target code, the macro subtracts the duration of a consecutive pair of `GET_TIME` calls. Although this macro compensates for the direct cost of timestamp acquisition by subtracting the duration of one `GET_TIME` interval, this compensation only removes the first-order timing overhead. In the LithOS/XNG version, `GET_TIME` requests a hypercall to retrieve the system time from the hypervisor. This introduces additional variability directly into the measurement pipeline that is not fully canceled by the subtraction. In LithOS-BM, this hypercall has been replaced by direct hardware access to the memory-mapped clock registers. Consequently, part of the observed performance improvement in LithOS-BM can be attributed not only to the elimination of hypervisor mediation in service execution but also to the reduction of residual measurement artifacts associated with timestamp acquisition.

Comparisons with industry alternatives (FreeRTOS, Zephyr, VxWorks, PikeOS) have been omitted as these systems either are not ARINC-653 compliant, or do not publish equivalent API-level microbenchmarks comparable with the obtained ARINC-653 APEX latency tables.

Beyond execution latency, memory footprint is a critical indicator of a kernel's suitability for constrained environments. The LithOS kernel object maintains a highly compact profile, with a `.text` (code) size of roughly 30 KB and a `.bss` (uninitialized data) section of only 1.6 KB. While LithOS-BM is inherently heavier than a rudimentary RTOS kernel due to the strict requirements of the ARINC 653 APEX interface, it remains vastly lighter than commercial separation kernels such as VxWorks or PikeOS.

7. Summary and conclusions

7.1. On the work conducted

This paper presented LithOS-BM, a new version of the LithOS RTOS designed for embedded systems with limited computing resources.

LithOS-BM followed an unusual development path. It was originally designed for the XNG environment, and later adapted for bare-metal execution, whereas most RTOS solutions are initially conceived as bare-metal systems, and then modified for execution as guest OS within a specific hypervisor.

Table 1: LithOS-BM and LithOS for XNG performance results

Service Group	Service / Metric	BM [μs]	XNG [μs]	Service Group	Service / APEX API	BM [μs]	XNG [μs]
Latency & Shuffle Times	Process Preemption Time	4	39	Buffer	CREATE_BUFFER	14	15
	Asynchronous Event Proc.	3	28		GET_BUFFER_ID	11	13
	FPU Context Switch Overhead	2	9		GET_BUFFER_STATUS	1	6
	Semaphore Shuffle Time	4	32		SEND_BUFFER	1	6
	Event Shuffle Time	3	31		RECEIVE_BUFFER	1	6
	Buffer Shuffle Time	4	32	Blackboard	CREATE_BLACKBOARD	6	8
	Blackboard Shuffle Time	4	32		GET_BLACKBOARD_ID	4	8
	Hardware IRQ Latency	12	76		GET_BLACKBOARD_STATUS	1	6
Partition	GET_PARTITION_STATUS	< 1	5		DISPLAY_BLACKBOARD	1	6
	SET_PARTITION_MODE	< 1	5		READ_BLACKBOARD	1	6
Time	TIMED_WAIT	< 1	5		CLEAR_BLACKBOARD	< 1	5
	REPLENISH	2	9	Semaphore	CREATE_SEMAPHORE	6	8
	GET_TIME	2	9		GET_SEMAPHORE_ID	5	7
	PERIODIC_WAIT	< 1	5		SIGNAL_SEMAPHORE	< 1	5
Process	CREATE_PROCESS	12	15		WAIT_SEMAPHORE	< 1	5
	GET_PROCESS_ID	10	17		GET_SEMAPHORE_STATUS	< 1	5
	GET_PROCESS_STATUS	2	8	Event	CREATE_EVENT	5	8
	SET_PRIORITY	13	19		GET_EVENT_ID	4	7
	START	18	27		SET_EVENT	1	5
	STOP	3	9		WAIT_EVENT	< 1	5
	DELAYED_START	17	26		RESET_EVENT	< 1	5
	GET_MY_ID	< 1	6		GET_EVENT_STATUS	1	6
	SUSPEND	2	8	Health Monitor	CREATE_ERROR_HANDLER	< 1	5
	RESUME	15	22		GET_ERROR_STATUS	< 1	4
	LOCK_PREEMPTION	1	7		REPORT_APPLICATION_MSG	1	5
	UNLOCK_PREEMPTION	14	20		RAISE_APPLICATION_ERROR	4	42
				NP IRQ Services	CREATE_IRQ_EVENT_NP	11	12
					GET_IRQ_EVENT_ID_NP	6	8
					GET_IRQ_EVENT_STATUS_NP	1	5
					WAIT_IRQ_EVENT_NP	< 1	5
					RESET_IRQ_EVENT_NP	< 1	5

In this standalone configuration, LithOS-BM implements the capabilities previously handled by the hypervisor. Such autonomy enhances efficiency and reduces overhead for certain internal operations, but it also requires incorporating all necessary system management mechanisms directly into the RTOS kernel.

The transition to a bare-metal architecture preserves ARINC 653 API, while eliminating the hypervisor layer to produce a deterministic, lightweight runtime environment for bare-metal execution of standalone applications.

This approach unlocks low-level hardware control and demonstrates that core ARINC 653 functionalities can extend to deeply embedded edge nodes without hypervisors or middleware.

While designed to fulfill the requirements of embedded systems targeting the aerospace sector, LithOS-BM extends to applications beyond this domain such as the METAMORPHA EU project (section 3).

7.2. Future work

This work presents a fully functional proof-of-concept prototype that successfully meets hard real-time execution constraints, though it currently operates with certain architectural limitations.

Given that the system runs a single application with full ownership of the hardware, there is no strict requirement to enforce memory isolation. However, defining regions with specific permissions is necessary to protect critical code or data within the application. Therefore, a potential future improvement is the extension of the LTCF component to allow partitioning the memory into regions of specific attributes through the MPU.

As noted earlier, LithOS-BM and AppSw run at the same privilege level. A potential improvement to ensure the HM is unaffected by AppSw errors would be relocating the user code to a lower privilege level to handle application exceptions at a higher privilege level. This privilege separation is a fundamental architectural principle to guarantee system integrity in future iterations of this version.

Acknowledgements

This project has received funding from Horizon Europe, the European Union's Framework Programme for Research and Innovation, under Grant Agreement No. 101057457 (METAMORPHA).

References

Aeronautical Radio, Inc., 2010. ARINC specification 653P1-3 — Avionics application software standard interface. Part 1 - Required Services. Aeronautical Radio, Inc., 2551 Riva Road, Annapolis, Maryland 21401-7435.

Aeronautical Radio, Inc., 2012. ARINC specification 653P2-2 — Avionics application software standard interface. Part 2 - Extended Services. Aeronautical Radio, Inc., 2551 Riva Road, Annapolis, Maryland 21401-7435.

European Commission, 2022. Made-to-measure micromachining with laser beams tailored in amplitude and phase (metamorpha). EU Funding & Tenders Portal, Grant Agreement ID: 101057457.
URL: https://cordis.europa.eu/project/id/101057457

Losa, A., et al., 2024. Hardware-assisted virtualization extensions for leon processors in mixed-criticality systems. Microprocessors and Microsystems 112, 105130.
DOI: 10.1016/j.micpro.2024.105130

Masmano, M., Ripoll, I., Crespo, A., Metge, J. J., 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., Metge, J. J., 2010. Lithos: a arinc-653 guest operating for xtratum. In: Proceedings of the 12th Real-Time Linux Workshop (RTLWS). Nairobi, Kenya.