Reference Architectures for Mixed-Criticality Virtualization in Next-Generation Space Systems

SOFTWARE PRODUCT ASSURANCE CONFERENCE 2025
22-25 September | ESA ESTEC | The Netherlands

Gran Sasso Science Institute &Thales Alenia Space Italia
Daniele Masti, Ester Maio

Date: 22/09/2025
Ref:
Template: 83230347-COM-TAS-EN-012

INTRODUCTION : MOTIVATION

/// The evolution of satellite systems has introduced a need for flexible and modular software architectures that support both safety-critical and non-critical applications on shared hardware.

/// The aim of this work is to explore advanced software architectures capable of meeting these demands, focusing on integrating hypervisors, nested virtualization, containerization, and security mechanisms such as firewalls to ensure controlled interactions between software zones

/// The collaboration between Thales Alenia Space and GSSI brings together the fields of research and industry to introduce a first proposal for Software Architectures for New Generation Virtualization Infrastructures in Space. This initiative aims to balance safety, flexibility, and ECSS compliance in mixed-criticality systems.

Date: 22/09/2025
Ref:
Template: 83230347-COM-TAS-EN-012

SUMMARY

INTRODUCTION:SW CRITICALITY & APPLICABLE STANDARDS

/// Software Product Assurance (ECSS-Q-ST-80C Rev.1) &Software Engineering (ECSS-E-ST-40C)

/// Dependability (ECSS-Q-ST-30C Rev.1)

- Focuses on reliability, availability, and maintainability.
- Implements fault tolerance and recovery mechanisms.
- Implementation Strategies

Date: 22/09/2025
Ref:
Template: 83230347-COM-TAS-EN-012

INTRODUCTION:SW CRITICALITY & APPLICABLE STANDARDS

/// Software Product Assurance (ECSS-Q-ST-80C Rev.1) &Software Engineering (ECSS-E-ST-40C)

/// Dependability (ECSS-Q-ST-30C Rev.1)

- Focuses on reliability, availability, and maintainability.
- Implements fault tolerance and recovery mechanisms.
- Implementation Strategies

/// Software Dependability and Safety Handbook (ECSS-Q-HB-80-03A Rev.1)

- Provides guidelines for achieving software dependability and safety.
- Includes best practices for risk assessment and mitigation.

INTRODUCTION:SW CRITICALITY & APPLICABLE STANDARDS

/// Software Product Assurance (ECSS-Q-ST-80C Rev.1) &Software Engineering (ECSS-E-ST-40C)

/// Dependability (ECSS-Q-ST-30C Rev.1)

- Focuses on reliability, availability, and maintainability.
- Implements fault tolerance and recovery mechanisms.
- Implementation Strategies

/// Software Dependability and Safety Handbook (ECSS-Q-HB-80-03A Rev.1)

- Provides guidelines for achieving software dependability and safety.
- Includes best practices for risk assessment and mitigation.

/// ARINC 653

- Specifies an operational environment for application software in IMA modules.
- Ensures deterministic behavior and resource partitioning.
- Supports mixed criticality by isolating software components with different criticality levels.

Date: 22/09/2025
Ref:
Template: 83230347-COM-TAS-EN-012

INTRODUCTION:SW CRITICALITY & ECSS

/// According to ECSS-Q-ST-80C:

- A software dependability and safety analysis of the software products, using the results of system-level safety and dependability analyses, to determine the criticality of the individual software components
- ECSS methodology shall be applied to evaluate the severity of undesirable scenarios Uses a 4-level grading system (Catastrophic - Critical - Major - Minor)
- If it cannot be prevented that software components cause failures of higher criticality components, due to failure propagation or use of shared resources, then all the involved components shall be classified at the highest criticality category among them.

/// Cross-domain similarity – Railway Software Standards

- Similar principles apply in the **railway domain**, where the **EN 50128** standard defines **Design Assurance Level A (DAL-A)** as requiring strict separation to prevent fault propagation from lower to higher criticality components.
- Just like ECSS, this includes enforced partitioning, isolation of memory and communication, and architectural safeguards.

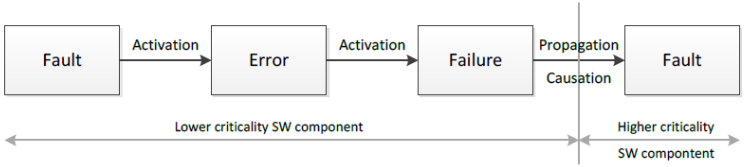
Date: 22/09/2025
Ref:
Template: 83230347-COM-TAS-EN-012

SOFTWARE CRITICALITY AND FAILURE PROPAGATION

/// The propagation of a failure to other software components or systems is a major threat to system dependability and safety.

- Suitable measures need to be put in place to prevent this from happening

/// Software failure propagation has a direct influence on the possibility of classifying software components at different criticality categories.



/// **There exists many technical solutions to prevent propagation:**

/// **Partitioning**: provides isolation between functionally independent software components to restrain and/or isolate faults. Segregation is often, but not necessarily, implemented by using different computers. A set of hardware and software approaches can minimize failure propagation

/// **Safety Monitoring:**This technique protects against specific failure conditions by directly monitoring a function which would contribute to the failure condition

Date: 22/09/2025
Ref:
Template: 83230347-COM-TAS-EN-012

FROM ARINC-653 TO MIXED-CRITICALITY ARCHITECTURES

/// **ARINC 653** is a recognized standard for space and time partitioning in real-time operating systems, originally developed for avionics but increasingly adopted in the **space domain**. ARINC 653 provides a robust framework for hosting multiple software applications—each with different criticality levels—on a single computing platform, while ensuring strict isolation and deterministic behavior

/// **How does it work?**

/// **1)Dividing the software as a whole into partitions**

- Partition will include all the software that provide a specific function.
- A most common method to implement separation between partitions is virtualization, but is it always the right solution?

Date: 22/09/2025
Ref:
Template: 83230347-COM-TAS-EN-012

FROM ARINC-653 TO MIXED-CRITICALITY ARCHITECTURES

/// **ARINC 653** is a recognized standard for space and time partitioning in real-time operating systems, originally developed for avionics but increasingly adopted in the **space domain**. ARINC 653 provides a robust framework for hosting multiple software applications—each with different criticality levels—on a single computing platform, while ensuring strict isolation and deterministic behavior

/// **How does it work?**

/// **1)Dividing the software as a whole into partitions**

- Partition will include all the software that provide a specific function.
- A most common method to implement separation between partitions is virtualization, but is it always the right solution?

/// **2)Inter-Partition Communication (IPC):**

- One of the main challenges addressed in these hypervisors is supporting ARINC-653's inter-partition communication mechanisms. ARINC-653 defines two specific IPC port type

Date: 22/09/2025
Ref:
Template: 83230347-COM-TAS-EN-012

FROM ARINC-653 TO MIXED-CRITICALITY ARCHITECTURES

/// **Inter-Partition Communication (IPC) ports:**

/// Sampling Ports: Only one message is held at any time, effectively providing the latest snapshot of data.

/// Queuing Ports: Can buffer multiple messages in a queue for sequential processing.

/// These IPC mechanisms differ significantly from conventional network or inter-process communication protocols used in industrial systems (e.g., Ethernet-based protocols like Ether CAT or PROFINET). Because of this difference, hypervisors need to handle these ARINC-specific communication requirements carefully and often design IPC support from the ground up.

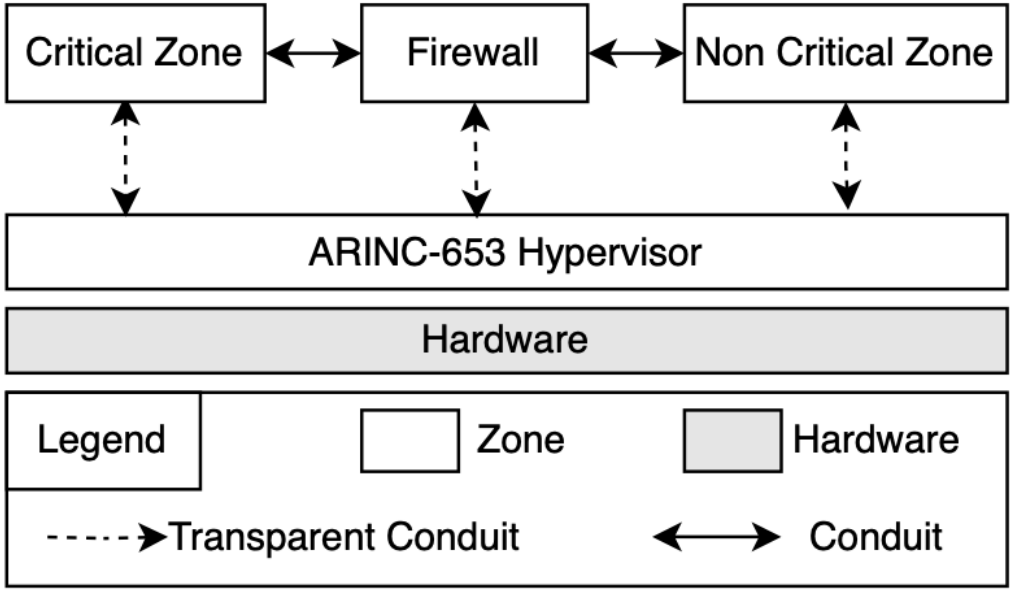
Date: 22/09/2025
Ref:
Template: 83230347-COM-TAS-EN-012

OVERVIEW OF THE THREE REFERENCE ARCHITECTURES

- A most common method to implement separation between partitions is virtualization, **is it also the right solution to integrate non-critical components?**
- Different vendors have proposed different idea. Which is the right one? **We answer such a question using a conduits and zones approach**
- Which are the zones?
  - **VMM/Hypervisor**
  - **Critical software zone(s)**
  - **Firewall/Router**
  - **Non-critic / service oriented**
- These macro-components are found in basically all commercial and open source mixed criticality platforms

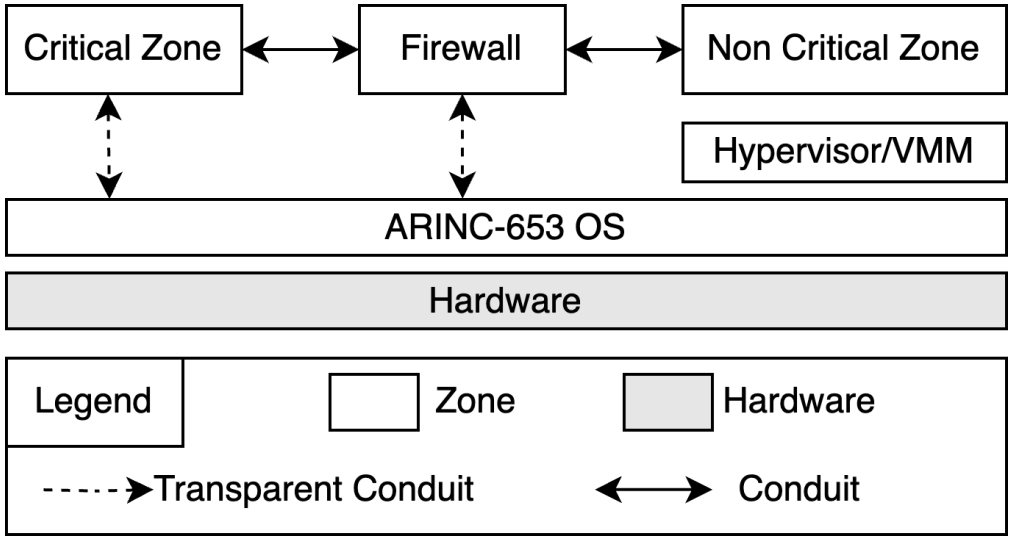
ARCHITECTURE A: STRONG ISOLATION VIA HYPERVISOR

All components are virtualized. High degree of separation and flexibility, but with added overhead



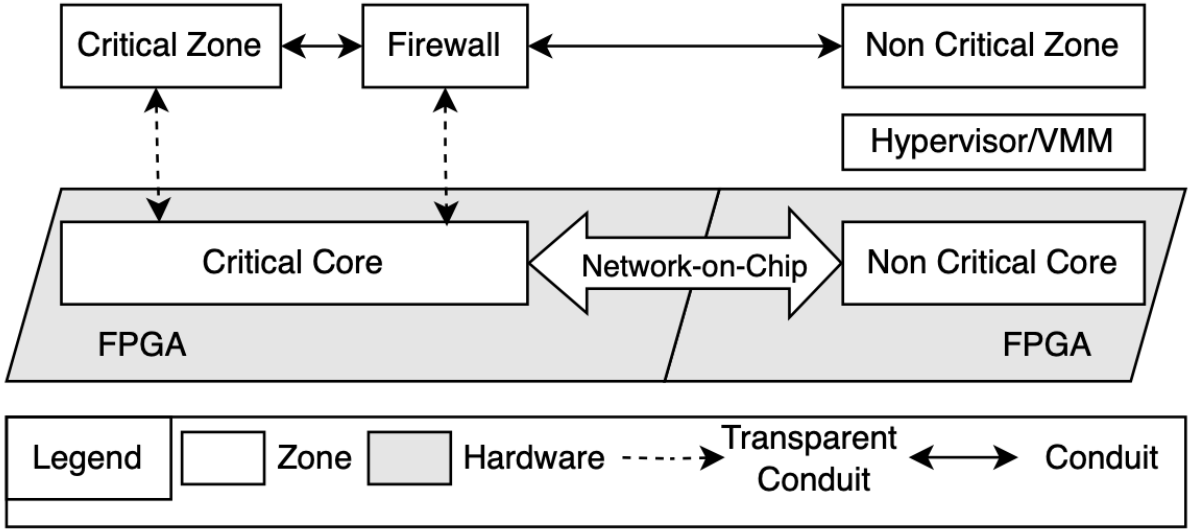
ARCHITECTURE B: SIMPLER, LESS OVERHEAD

Critical software runs directly on the OS. This simplifies the architecture but provides less isolation



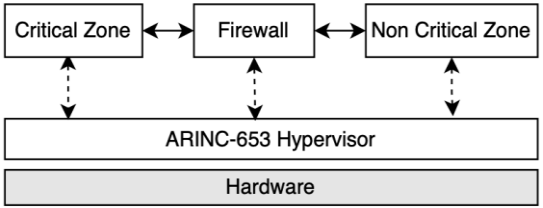
ARCHITECTURE C: PHYSICAL SEPARATION WITH FPGA

Hardware-based separation using FPGA cores. Highest robustness, but the most complex and expensive

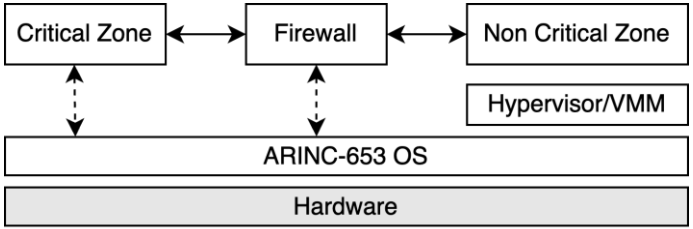


COMPARISON: RELIABILITY, AVAILABILITY, MAINTAINABILITY

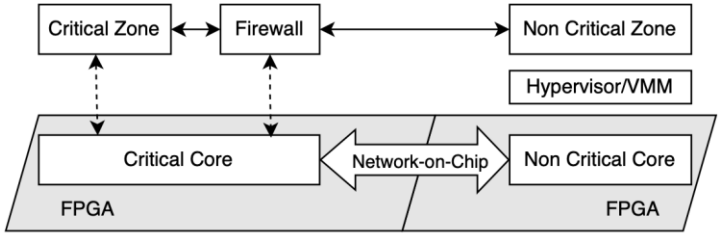
The best choice depends on the mission profile. There is no universally superior solution



Arch. A: Balanced and modular



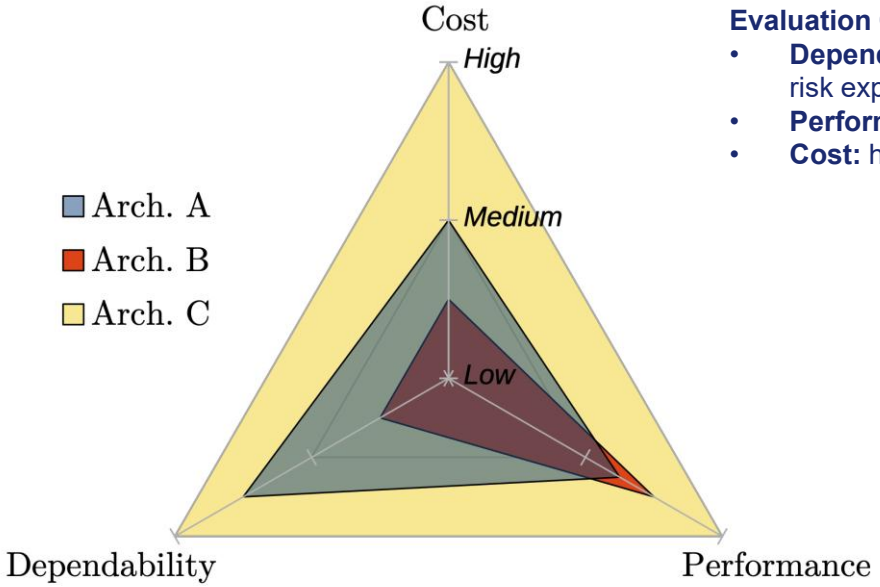
Arch. B: Low-cost, legacy friendly



Arch. C: Most robust, but costly and complex

COMPARISON: RELIABILITY, AVAILABILITY, MAINTAINABILITY

We evaluated all three architectures based on ECSS attributes: fault propagation, recoverability, OTA support, etc



Evaluation Criteria:

- **Dependability:** fault propagation, fault containment, recovery capability, security risk exposure
- **Performance:** virtualization overhead, real-time responsiveness, data latency
- **Cost:** hardware complexity, certification effort, maintenance burden

Architecture A (Balanced)

Moderate cost and performance overhead. Good isolation through full virtualization. Suitable for systems needing flexibility and OTA updates.

Architecture B (Legacy-friendly)

Lowest cost and minimal overhead, but **lower dependability** due to **shared execution context** between critical and non-critical zones. (**Security risk** from weak partitioning)

Architecture C (Robust)

Physically separates zones using FPGA cores. Highest dependability and security, but at the cost of higher development complexity and energy footprint.

THANK YOU !

email: daniele.masti@gssi.it
email: ester.maio@thalesaleniaspace.com

- Special thanks to
  - Luigia Ambrosio
  - Gianluca Caruso
  - Franco Raimondi
  - Patrizio Pelliccione
  - Alberto Petrucci
  - Francesco Basciani