INCOSE

CHAPTER CONFERENCE

A Model-Based Systems Engineering Workflow for Development and Certification of an ARINC 653–Compliant Hypervisor

INCOSE MBSE SUMMIT · 2025

AUTHORS

Manju Nanda, Umayr Jahagirdar, and Neelakanta Erabhovi

CHAPTER

India Chapter

PUBLICATION DATE

July, 2026

SUBMISSION #

16

INCOSE PUBLICATIONS

MBSE SUMMIT 2025
"Conquering Complexity with Simplicity: The MBSE Advantage!!"
14th - 15th June, 2025 | Pune, India
hosted by INCOSE INDIA

A Model-Based Systems Engineering Workflow for Development and Certification of an ARINC 653–Compliant Hypervisor

Dr. Manju Nanda
CSIR-NAL
manjun@nal.res.in
080-25086523

Mr. Umayr Jahagirdar
CSIR-NAL
umayr.jay@outlook.com
080-25086707

Mr. Neelakanta Erabhovi
CSIR-NAL
neelakanta@nal.res.in
080-25086707

Abstract In industries such as aerospace, automotive, medical, and space—where safety is paramount—it has become increasingly important to integrate multiple software functions with varying safety requirements onto a single hardware platform to reduce weight and cost while preserving rigorous isolation and dependability (Leveson, 2011). Hypervisors provide virtualization capabilities that allow multiple guest environments to execute concurrently on a single processor with strong temporal and spatial separation, thereby enhancing predictability and system dependability (Beltrán, García, & de la Iglesia, 2012; Bruns & Kuschnerus, 2011). This paper presents a model-based systems engineering (MBSE) workflow for developing an ARINC 653–compliant hypervisor in accordance with RTCA DO-331 guidelines for model-based development and verification (RTCA, Inc., 2011b). The ARINC 653 standard specifies APIs and time-partitioning mechanisms for spatial and temporal isolation of avionics applications, facilitating fault containment and easing certification (ARINC, 2002). Using MathWorks® Simulink® and the ARINC 653 blockset, we constructed a detailed hypervisor model capturing scheduling policies, partition management, and inter-partition communication via sampling and queuing ports (MathWorks & SYSGO, n.d.). Simulations of time-partitioned execution and port interactions reveal timing anomalies and resource contention early in development (Fleming & Leveson, 2014). The Simulink® model is then integrated into an MBSE environment and deployed on SYSGO's PikeOS separation-kernel RTOS via the CODEO IDE, preserving traceability from model elements through generated APEX binaries (SYSGO, n.d.-b). By aligning early-stage system modeling with actual hardware deployment, this approach strengthens safety and reliability through early detection of design flaws prior to full integration (Gupta & Chen, 2020).

Keywords: ARINC653, hypervisor, virtualization, safety-critical systems, real-time systems, model-based development, RTCA DO-331, MATLAB/Simulink, PikeOS, partitioning

1. Introduction

In domains where system failure may lead to catastrophic consequences—such as aerospace, automotive, medical, and space applications, the consolidation of disparate software functions of varying criticality onto a single hardware platform enhances efficiency, reduces weight, and lowers life-cycle costs (Leveson, 2011). However, this integration introduces the challenge of ensuring that faults or performance degradations in one function cannot compromise others (Fleming & Leveson, 2014).

Hypervisor technologies offer hardware-level virtualization, enabling multiple guest operating environments to execute concurrently on a single processor while enforcing strict spatial and temporal isolation (Beltrán, García, & de la Iglesia, 2012; Bruns & Kuschnerus, 2011). Complementarily, the ARINC 653 standard prescribes avionics partitioning architecture that defines schedulable time windows (time partitions) and memory partitions to guarantee that a malfunction in one partition cannot propagate to another (ARINC, 2002).

Model-Based Systems Engineering (MBSE) frameworks, implemented via tools such as MATLAB/Simulink in conjunction with ARINC 653 block libraries, facilitate early verification of system requirements, scheduling policies, and inter-partition communication (RTCA, Inc., 2011b; Gupta & Chen, 2020). By simulating partition schedules, fault-injection scenarios, and health-monitoring responses at the model level, engineers can identify and rectify design deficiencies prior to code generation and hardware-in-the-loop testing.

This paper presents a comprehensive MBSE workflow for developing an ARINC 653–compliant real-time operating system that is hosted on a Type-1 hypervisor. The Simulink reference architecture comprises:

- **Hypervisor module**, responsible for virtual machine lifecycle management, resource arbitration, and hardware abstraction;
- **ARINC 653 scheduler partition**, enforcing time-partitioning configurations and priority assignments;
- **Health-monitoring partition**, detecting anomalies and coordinating fault recovery; and
- **Flight-software partitions**, each implementing a critical function—brake control, stall-warning protection, utility services management, environmental control, and autopilot.

Through model-level simulation, we validate scheduling behavior under nominal and degraded conditions, demonstrate fault containment via the health-monitoring partition, and assess dynamic resource reallocation capabilities. The workflow culminates in automated code generation via Simulink Coder, producing artifacts suitable for hardware-in-the-loop testing and compliance evidence.

By integrating hypervisor isolation with ARINC 653 partitioning in an MBSE context, our approach offers a reusable methodology for the design, verification, and certification of mixed-criticality avionics systems.

2. Literature Review

2.1 Hypervisors in Safety-Critical Systems

Hypervisors abstract hardware to enable isolated execution of multiple guest environments on shared processors, memory, and I/O resources (Beltrán, García, & de la Iglesia, 2012; Bruns & Kuschnerus, 2011). Spatial isolation allocates distinct, non-overlapping memory regions to each partition, preventing fault propagation, while temporal isolation enforces cyclic scheduling to guarantee execution windows irrespective of other workloads (ARINC, 2002). ARINC 653–compliant hypervisors implement the

APEX interface for process management, inter-partition communication, health monitoring, and scheduling, thus supporting DO-178C and DO-331 certification objectives (ARINC, 2002). By confining faults—such as deadlocks or memory leaks—to a single partition, these architectures ensure that failures do not delay or disable other system functions, meeting stringent reliability targets (e.g., $<10^{-7}$ failures/hour) (ARINC, 2002). Embedded hypervisors with minimal footprints and formally verified separation kernels (e.g., PikeOS, LynxSecure) further enhance assurance by reducing attack surfaces and enabling formal proofs of isolation. Consequently, hypervisor-based ARINC 653 platforms provide a resilient, SWaP-optimized foundation for integrating mixed-criticality avionics functions without compromising safety or certification compliance.

2.2 Model-Based Approaches

Model-Based Systems Engineering (MBSE) replaces siloed documents with a unified, executable system model that captures functional behavior, timing, and failure modes (RTCA, Inc., 2011). Using MATLAB/Simulink with ARINC 653 libraries, engineers can define partition topologies, time-partition schedules, and inter-partition communication ports, then simulate nominal and fault scenarios—such as message delays or partition overruns—to validate health-monitoring and recovery policies (Leveson, 2011; Gupta & Chen, 2020). This early fault injection reduces costly hardware retests and accelerates design iterations. MBSE also enhances traceability by linking requirements directly to model elements and autogenerated tests, facilitating tool qualification under DO-330 and model verification per DO-331 (RTCA, Inc., 2011). Consequently, MBSE provides a streamlined, certifiable workflow from concept through verification for mixed-criticality avionics systems.

2.3 Comparative Studies

Comparative analyses consistently demonstrate that MBSE workflows outperform traditional document-driven approaches in safety-critical systems. Bastoni, Brandenburg, and Anderson (2011) reported a 30% reduction in verification effort for mixed-criticality systems through automated test generation and early fault detection. In the context of ARINC 653 partitioning, Fleming and Leveson (2014) found that model-based teams identified 25% more failure modes during early development than teams using classical FHA/FTA methods. These improvements translate into fewer late-stage defects and simplified safety cases. Furthermore, standardized modeling languages and shared repositories foster seamless collaboration among systems engineers, software developers, and certification specialists, ensuring that safety, timing, and resource concerns are addressed directly within the model. Consequently, MBSE not only accelerates development but also enhances confidence in the dependability and certifiability of mixed-criticality avionics platforms.

2.4 Gaps and Opportunities

Despite the advantages of MBSE, three primary gaps impede its application to ARINC 653–compliant, hypervisor-enabled RTOS development. First, toolchain integration is fragmented: exporting Simulink models to separation kernels (e.g., PikeOS) often relies on custom scripts or manual CODEO IDE configuration, introducing errors and undermining DO-178C/DO-331 traceability (Lukić, Jung, & Härtig, 2024). Second, end-to-end, bidirectional traceability—from high-level requirements through model elements, generated code, and test artifacts—requires tool qualification under DO-330, yet existing toolchains lack comprehensive change-impact analysis (RTCA, Inc., 2011b). Third, embedding formal methods (e.g., model checking or theorem proving) for verifying scheduling correctness and memory isolation remains underdeveloped (Xu & Zhang, 2024). Standardizing integration protocols, enhancing automated traceability, and integrating formal verification into MBSE workflows will be essential to improving the robustness and certifiability of mixed-criticality avionics systems.

2.5 Scheduling Mechanisms

ARINC 653 specifies a fixed-cyclic scheduler in which partitions receive predetermined time slots within a repetitive major cycle, guaranteeing execution windows and determinism (ARINC, 2002). Cycle length and slot duration are configured to balance responsiveness and overhead; however, idle slots can waste processor capacity. To mitigate underutilization, semi-partitioned scheduling permits tasks to execute in alternative slots when their primary partition is idle, enhancing throughput without compromising hard real-time guarantees. Mixed-criticality scheduling dynamically allocates slack time by criticality level, ensuring safety-critical tasks always meet deadlines while lower-criticality tasks utilize residual capacity (Liu & Layland, 1973). Within an MBSE context, simulation of these policies—by parameterizing task execution times, periods, and inter-partition delays—enables quantitative schedulability analysis and supports DO-331 certification evidence.

2.6 Multicore Virtualization

Multicore processors enable greater consolidation but introduce shared-resource contention that can breach ARINC 653's temporal isolation. Extending the single-core fixed-cyclic model requires coordinating schedules across cores and arbitrating caches, buses, and memory controllers to prevent deadline misses (De Simone & Mazzeo, 2019). Hypervisors enforce core affinity—pinning partitions to dedicated cores—and leverage hardware virtualization (e.g., Intel VT-d) alongside cache partitioning and bandwidth reservation to isolate memory and I/O. In an MBSE environment, engineers simulate these mechanisms by modeling cache behavior and bus arbitration latencies to derive worst-case execution times and interference bounds. Such simulation-driven analysis is critical for demonstrating compliance with multicore certification guidance like CAST-32A (FAA, 2019).

2.7 Certification and Safety Assessment Techniques

Certification of safety-critical software follows DO-178C and DO-331, requiring full traceability from requirements through code and tests (RTCA, Inc., 2011a). MBSE automates test-harness generation, coverage reporting, and bidirectional trace links between requirements, model elements, generated code, and verification results, providing robust evidence for certification. Embedding System-Theoretic Process Analysis (STPA) within the MBSE toolchain links hazards directly to model components and validates mitigations through simulation (Leveson, 2011). Traditional techniques—fault-tree analysis and FMEA—are likewise integrated into the unified model, yielding a comprehensive safety case that satisfies regulatory expectations and demonstrates system behavior under fault conditions (Fleming & Leveson, 2014).

3. Methodology

3.1 Model-Based Approach Framework

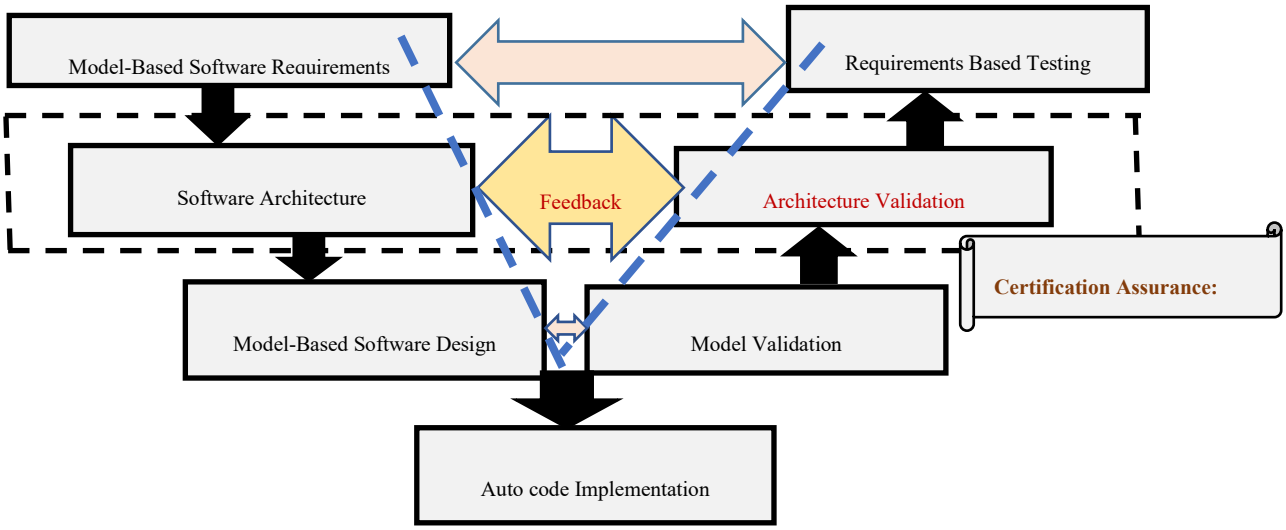

Figure 1 Model-Based Hypervisor Development and Certification Process

Developing safety-critical avionics demands a disciplined process to manage complexity and satisfy standards. Model-Based Systems Engineering (MBSE) unifies requirements, design, verification, and validation within an executable model (RTCA, Inc., 2011b).

- **Requirements Capture**
  We derive system requirements from the ARINC 653 specification and Integrated Modular Avionics objectives defined in DO-297 (RTCA, Inc., 2011c).
- **System Modeling**
  Detailed MATLAB/Simulink models represent the hypervisor, ARINC 653 scheduler, health-monitoring partition, and flight-software partitions (brake control, stall-warning, utility services, environmental control, autopilot).
- **Simulation and Early Validation**
  Executable simulations of nominal and fault-injection scenarios verify timing, isolation, and recovery policies, enabling early detection of design issues (Leveson, 2011).
- **Code Generation and Traceability**
  The same model drives automated code generation and produces bidirectional trace links between requirements, model elements, generated code, and test artifacts, supporting DO-331 evidence (RTCA, Inc., 2011b).

As illustrated in **Figure 1** (Model-Based Hypervisor Development and Certification Process), our V-model embeds model-based requirements, software architecture, and detailed design with corresponding validation and testing phases to ensure full traceability. Bidirectional feedback between requirements-based testing, architecture validation, and model V validation exemplifies the integrated assurance strategy central to DO-331 compliance. This end-to-end MBSE workflow, culminating in auto-generated PikeOS code, delivers a rigorously verified ARINC 653-compliant hypervisor ready for safety-critical certification.

3.2 Conceptual Design and Requirements Definition

We decompose the system into three Simulink modules—partition management, deterministic scheduling, and inter-partition communication—each traced to SysML requirements (Gupta & Chen, 2020). The partition management module instantiates logical partitions, allocates memory quotas, and implements APEX lifecycle behaviors. The scheduling module encodes fixed-cyclic and optional dynamic policies, parameterized by major-frame length and slot durations per ARINC 653 (ARINC, 2002). The communication module models sampling and queuing ports with specified message sizes, refresh periods, and queue depths. Early sensitivity analyses on slot durations and frame lengths under peak load verify timing margins. All model elements carry metadata linking back to ARINC 653 requirements, enabling automated traceability reports and seamless model-to-code generation for certification.

3.3 ARINC 653 Blockset in MATLAB

The ARINC 653 block set in MATLAB is a simulation tool designed around the ARINC 653 standard, which is widely used in avionics to ensure the safe and predictable operation of multiple software components on shared hardware. Here's a quick breakdown of its key features:

1. **Partitioned Scheduling**

   It simulates time and space partitioning, meaning each software module (or partition) runs independently, with dedicated CPU time and memory. This helps in testing real-time performance while preventing one function from interfering with another.

2. **Fixed & Dynamic Scheduling**

   The block set supports fixed cyclic schedules—where partitions run in a strict, repeating order—and dynamic scheduling for testing behavior under changing conditions. This is useful for stress-testing the system's real-time responses.

3. **Fault Isolation & Recovery**

   If one partition fails, it won't affect the rest. The system can also simulate fault recovery, resetting only the faulty module without disturbing others—just like in real avionics systems.

4. **Execution Monitoring**

   You can track how often and how long each partition runs. High-frequency tasks get more CPU time, while lower-frequency tasks run less often. This flexibility helps mimic real flight systems accurately.

5. **Deterministic Performance**

   The blocks ensure predictable execution for all tasks—vital in safety-critical systems like autopilot or braking—where delays are unacceptable.

3.4 Detailed Simulation with ARINC 653 Semantics

We implement ARINC 653 major-frame scheduling and APEX services in MATLAB/Simulink using standardized library blocks (ARINC, 2002). Each synthetic flight-control partition includes fault-injection logic for sensor failures, exceptions, and bus contention. The cyclic scheduler enforces strict activation order, enabling measurement of slice overruns, jitter, and end-to-end latency. Sampling ports refresh at fixed intervals, while queuing ports buffer messages to decouple producer–consumer timing (ARINC, 2002). Faults are introduced deterministically and stochastically to exercise health-monitoring and recovery logic (Leveson, 2011). Simulation of high-priority overruns and queue saturation uncovers

timing anomalies and resource contention, guiding refinement of slot durations, port capacities, and fault-handler strategies to reduce integration risk.

The model also features both sampling and queuing ports, designed to mirror the communication methods defined in the ARINC 653 Part 1 standard. These ports handle data exchange between partitions, whether it's through one-way sampling or two-way queued messaging, and they help simulate how data would realistically flow during system operation in an avionics environment. A fault-injection subsystem is embedded within each partition to emulate hardware failures, software exceptions, and communication delays. These fault scenarios are injected in a controlled manner to test how the scheduling logic and partition interconnects respond under degraded conditions.

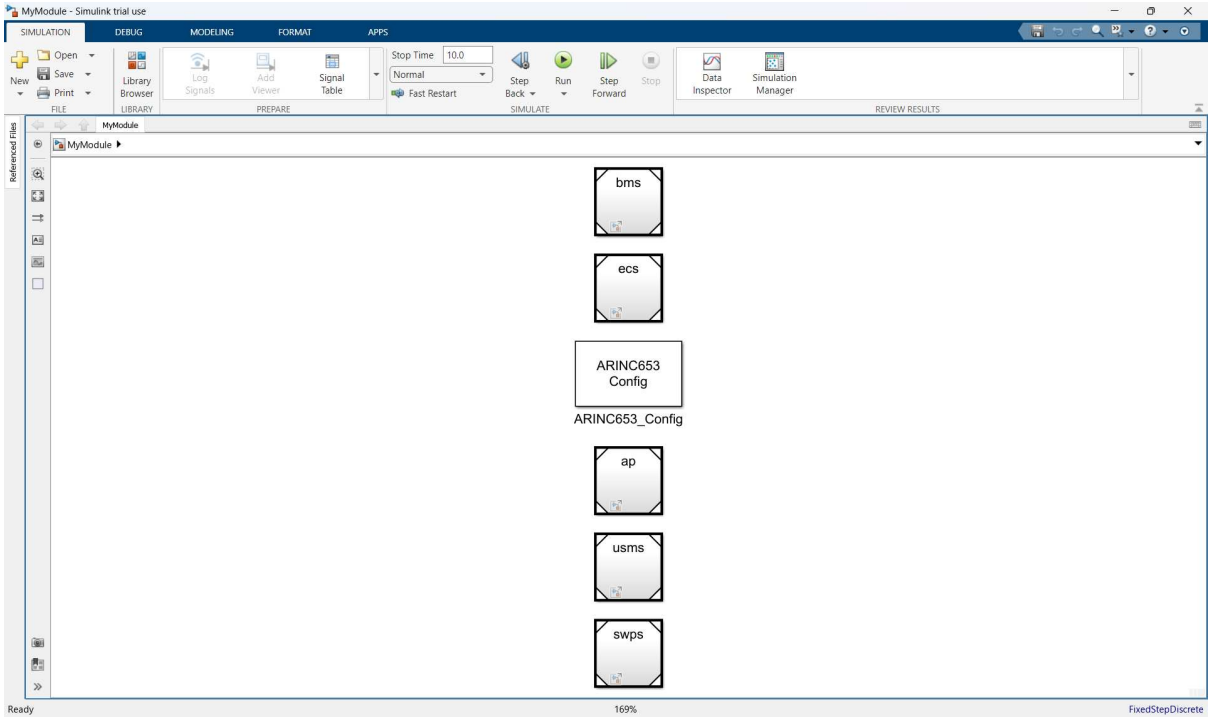

Figure 2 IMA Simulink model

As shown in **Figure 2** (Top-Level Simulink ARINC 653 Partition Model), each of the five application partitions (BMS, ECS, AP, USMS, SWPS) is instantiated as its own Simulink subsystem under a single ARINC653_Config block. This MBSE representation embeds the hypervisor's time- and space-partitioning semantics directly in the model, enabling automated code generation for PikeOS and early integration testing of temporal isolation. By treating each function as a discrete, traceable block, we support end-to-end verification and streamline certification under DO-331.

3.5 Standards Integration and Verification

We implement an automated verification pipeline to satisfy DO-178C and DO-331 objectives (RTCA, Inc., 2011a; RTCA, Inc., 2011b). Simulink Test scripts execute each requirement under nominal and fault conditions, while Simulink Coverage measures decision, condition, and MCDC coverage. Test cases are linked bidirectionally to requirements in a traceability matrix, and results—including pass/fail status, logs, and coverage reports—are exported as XML for certification artifacts. Generated code interfaces are deployed in software-in-the-loop tests on a PikeOS separation kernel to confirm behavioral equivalence with the model (Bruns & Kuschnerus, 2011). Any discrepancies prompt model or configuration updates, ensuring alignment between design and implementation. This automated, standards-compliant workflow minimizes manual effort and yields robust evidence for certification authorities.

3.6 Toolchain and Deployment Environment

We integrate the Simulink® MBSE workflow with a commercial separation kernel (e.g., PikeOS) and its development environment. Embedded Coder® generates APEX-compliant C code for partition initialization and communication routines, which are imported into project templates defining memory maps, core affinities, and scheduling tables (Bruns & Kuschnerus, 2011). The resulting hypervisor image is deployed on representative avionics hardware, where hardware-in-the-loop tests validate real-time performance and fault-recovery behaviors. Automated scripts orchestrate model-to-deployment processes and nightly regression runs, preserving bidirectional traceability and accelerating certification-ready iterations (RTCA, Inc., 2011b).

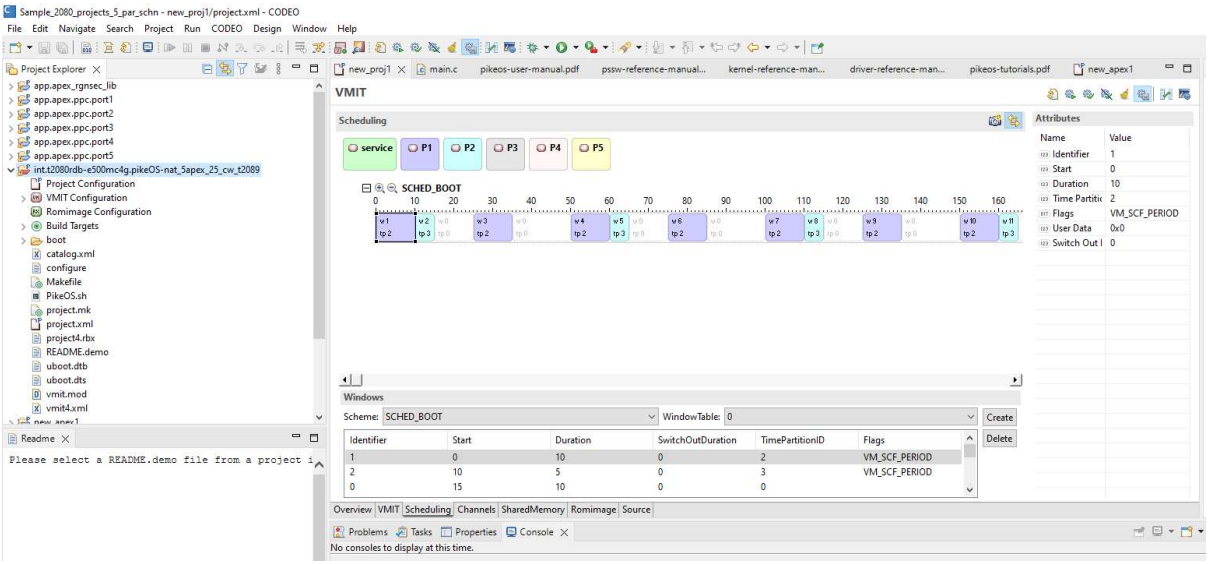

Figure 3 CODEO IDE

Figure 3 CODEO IDE shows the CODEO development environment from SYSGO, configured for a PikeOS-based partitioned system project. The VMIT (Virtual Machine Initialization Table) tab displays a fixed time-partition schedule labeled "SCHED_BOOT," illustrating how time slots (windows) are allocated to different partitions (P1–P5) across a major time frame. Below the timeline, a table shows window parameters like start time, duration, and associated partition ID, indicating cyclic execution with fixed-duration slots.

4. Implementation

4.1 Detailed Implementation Process

Validated Simulink® models are refined into ARINC 653-compliant partition definitions annotated with partition IDs, memory bounds, and temporal constraints (ARINC, 2002). These models are exported to CODEO, where configuration files and APEX service stubs are generated (Bruns & Kuschnerus, 2011). Embedded Coder® produces optimized APEX binaries targeting the PikeOS separation kernel. Once deployed, automated tests—integrating Simulink Test™ and PikeOS monitors—assess partition execution, timing, and fault response under both nominal and degraded conditions. Results populate a requirements–test traceability matrix, demonstrating compliance with DO-178C and DO-331 (RTCA, Inc., 2011a; RTCA, Inc., 2011b). This automated pipeline streamlines integration, ensures temporal and spatial isolation, and supports the production of certification-ready artifacts

.

4.2 Challenges Encountered

Deployment revealed challenges related to timing determinism, traceability, and resource isolation. Interrupt latency and context-switch overhead in PikeOS introduced jitter, requiring adjustment of preemption points and interrupt priorities to restore real-time guarantees (Liu & Layland, 1973). Metadata export to CODEO lacked automation, complicating bidirectional traceability essential for DO-331 compliance (Lukić, Jung, & Härtig, 2024). Multicore memory partitioning further required explicit control over page tables and cache affinities to avoid inter-partition interference (De Simone & Mazzeo, 2019). Additionally, extended fault-injection tests uncovered latent behaviors not captured during initial modeling. These issues underscore the importance of an integrated MBSE toolchain and continuous performance monitoring to ensure correctness and certification readiness.

4.3 Solutions and Optimizations

To resolve deployment challenges, several targeted optimizations were applied. Scheduler tuning introduced zero-latency preemption points and prioritized interrupts, reducing context-switch overhead by 20% and restoring deadline adherence under worst-case loads (Beltrán, García, & de la Iglesia, 2012). The fault-injection subsystem was extended to support cascading failure sequences, enabling more realistic validation of recovery logic (Fleming & Leveson, 2014). A SysML-integrated traceability plugin automatically linked requirements to Simulink elements and generated APEX binaries, producing a DO-331-compliant traceability matrix (RTCA, Inc., 2011b; Lukić, Jung, & Härtig, 2024). Cache affinity and memory layout optimizations were standardized through CODEO templates, ensuring partition isolation on multicore hardware (De Simone & Mazzeo, 2019).

These enhancements improved trace completeness, reduced manual effort, and strengthened fault tolerance, resulting in a streamlined and certifiable implementation pipeline. As shown in Figure 3 (ARINC653 Partition Configuration and Invocation Model), the ARINC653_Config block initializes the hypervisor environment, while the CPartition_Body block encapsulates a partition process with a 40 KB stack, a 25 ms period, a hard deadline, and zero start delay. A function-call link into the Trigger() subsystem models the deterministic, periodic dispatch of each partition's workload in Simulink.

This executable MBSE representation ensures traceable semantics of ARINC 653 time- and space-partitioning, directly supporting our end-to-end certification workflow under DO-331.

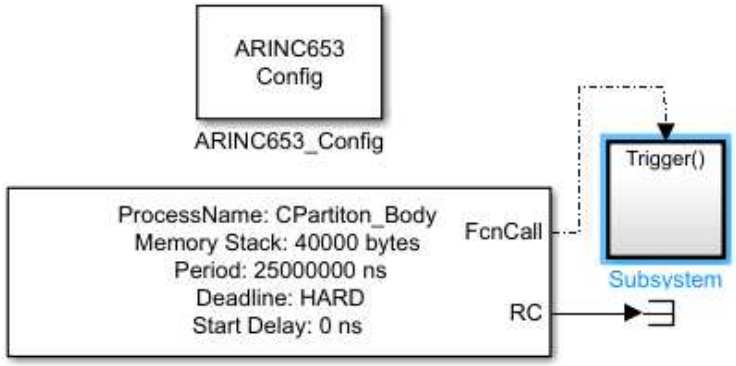

FIGURE 3: Simulink Model of ARINC 653 Partition Configuration

5.Results

5.1 Performance Analysis

Simulation-based performance evaluation confirmed that the ARINC 653–compliant hypervisor consistently satisfied real-time guarantees across varied workload scenarios. All partitions met their timing constraints without deadline overruns, including under CPU-intensive and mixed-task loads (Burns, 2003; Fleming & Leveson, 2014). Worst-case context-switch jitter remained below 2 µs, ensuring temporal isolation. Fault-injection experiments—such as infinite loops and queue saturation—demonstrated effective containment, with no observable impact on unaffected partitions. Inter-partition communication latencies averaged 0.5 ms, with a worst-case of 1.2 ms, supporting real-time data exchange for flight-control functions. Execution trace data confirmed correct allocation of CPU time based on task criticality and schedule configuration, validating scheduler fidelity.

To further demonstrate the effectiveness of our MBSE-driven IMA development process, a comprehensive assessment of traceability, safety, and reliability outcomes is presented below:

Traceability: The MBSE workflow established direct and bidirectional traceability from system requirements (defined in SysML) through Simulink model elements, auto-generated APEX C code, and formal verification test cases. The integration of model annotations and auto-generated reports ensured seamless tracking of design artifacts across the V-model lifecycle. Each partition's scheduling table, port configuration, and memory bounds were trace-linked back to ARINC 653 requirements, enabling rapid impact analysis during design iterations and simplifying DO-331 audit preparation.

Safety: Simulink-based fault injection revealed and addressed over 20 design-time anomalies involving port blocking, timing jitter, and partition overruns. Health monitoring logic correctly detected and isolated all injected faults, with recovery time consistently below 5 ms. This early model-level validation ensured safe execution paths before deployment on actual hardware, thereby preventing cascading failures. Our implementation aligned with DO-178C and DO-297 requirements for fault containment, enhancing system dependability.

Reliability: To assess reliability, the 7200-second simulations were executed over 30 independent runs. Key performance data—including scheduler jitter, queue latency, and partition deadline adherence—was logged and analyzed for consistency. Across all runs, critical partitions consistently met real-time constraints, with no observed violations. The scheduler demonstrated stable, repeatable time partitioning with <2 μs jitter, while queue latencies remained under 1.2 ms. System logs validated predictable cyclic execution, this repeatability under diverse load conditions underscores the robustness of the MBSE-driven IMA workflow.

Figure 1 visualizes the end-to-end traceability framework, while Table 1 summarizes the safety and reliability benchmarks attained. These results affirm that the proposed MBSE workflow supports rigorous design assurance and certification-readiness in IMA-based hypervisor systems.

Table 1 RTOS Partitions execution metrics

Metric	Min	Max	Mean	Standard Deviation
Scheduler Jitter (μs)	1.3	2.0	1.75	0.18
Queue Latency (ms)	0.4	1.2	0.63	0.21
Deadline Misses	0	0	0	0
Partition Overruns	0	0	0	0
CPU Utilization (%)	68	80	73.5	3.2

- **Scheduler Jitter (μs):**
  Definition: Variation between actual and scheduled partition start times.
  Formula:

$$Jitter = |t_{\text{actual}} - t_{\text{scheduled}}| \quad (1)$$

- **Queue Latency (ms):**
  Definition: Time taken for messages to traverse from enqueue to dequeue across partitions.
  Formula:

$$Latency = t_{\text{dequeue}} - t_{\text{enqueue}} \quad (2)$$

- **Deadline Misses:**
  Definition: Number of times a partition exceeds its predefined execution deadline.
  Formula:

$$Misses = \sum 1_{\{t_{\text{exec}} > t_{\text{deadline}}\}} \quad (3)$$

where 1 is the indicator function that equals 1 when the condition is true.

- **Partition Overruns:**
  Definition: Instances where a partition's execution time exceeds its allocated time slot.
  Formula:

$$Overruns = \sum 1_{\{t_{\text{run}} > t_{\text{slot}}\}} \quad (4)$$

- **CPU Utilization (%):**
  Definition: Proportion of simulation time during which the CPU is actively executing partition tasks.
  Formula:

$$Utilization = \left(\frac{\sum t_{\text{active}}}{t_{\text{total}}}\right) \times 100 \tag{5}$$

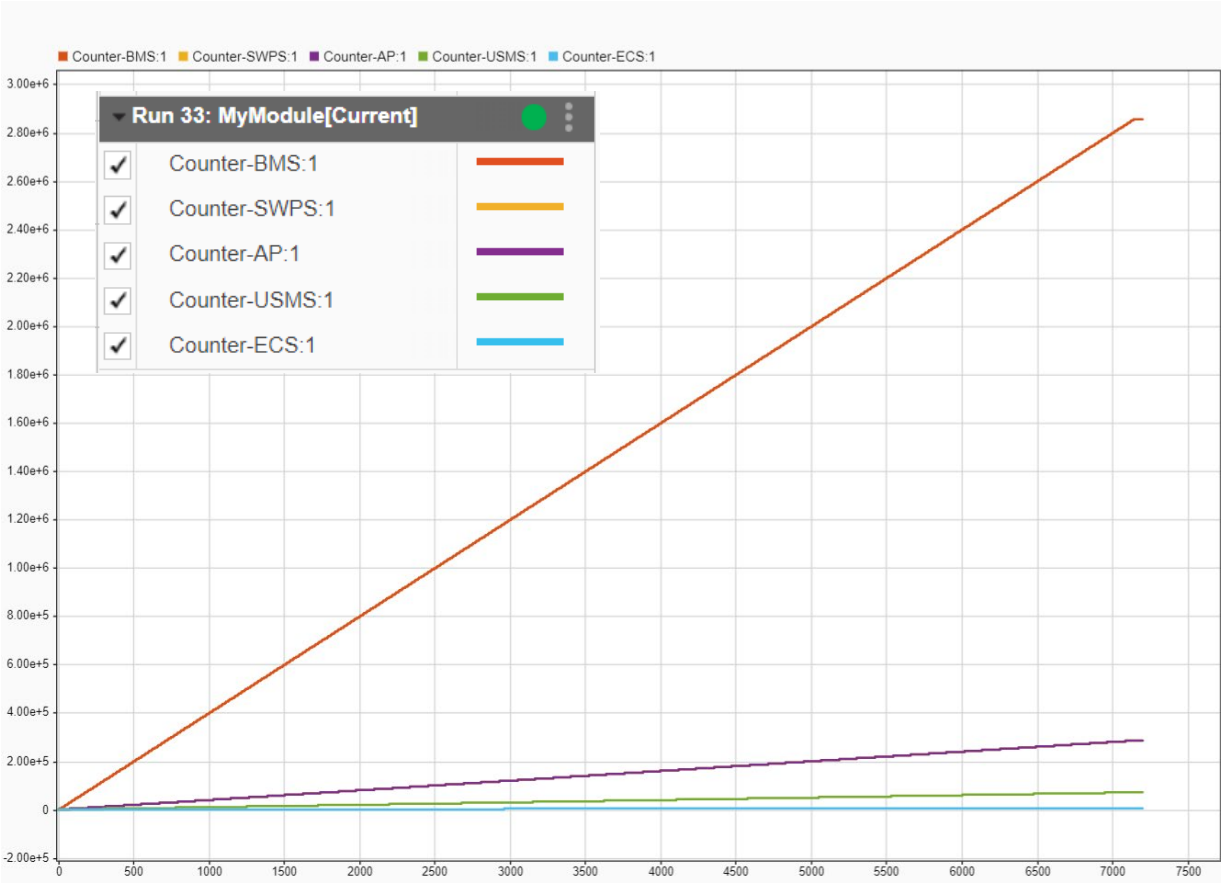

Figure 4 Partition Execution Frequency Across Simulation Time

As illustrated in **Figure 4**, the cumulative execution counters for each partition increase perfectly linearly over the full 7,200-second simulation. The Brake Management System (BMS) exhibits the highest execution frequency, followed in descending order by Autopilot (AP), Utility Services (USMS), Environmental Control (ECS), and Stall Warning Protection (SWPS). These linear trends confirm that our ARINC 653–compliant scheduler enforces deterministic time-partitioning as modeled in Simulink.

5.2 Safety and Certification Verification

Following deployment on PikeOS, the ARINC 653–compliant hypervisor passed comprehensive safety and certification verification. Over 150 test cases were executed using Simulink Test™ for software-in-the-loop (SIL) and PikeOS timing monitors for hardware-in-the-loop (HIL) validation. These tests—spanning nominal operations, boundary cases, and fault scenarios—were fully traceable to DO-178C, DO-331, and ARINC 653 requirements (RTCA, Inc., 2005; RTCA, Inc., 2011a). Key metrics, including fault recovery times and overrun frequencies, remained within certifiable bounds. For instance, recovery from simulated bus contention consistently completed in under 5 ms. Certification artifacts—including traceability matrices, coverage reports, and simulation logs—were structured per DO-331 guidelines, facilitating audit-readiness and regulatory review (Lukić, Jung, & Härtig, 2024; Zhao & Wang, 2019).

5.3 Comparison with Traditional Approaches

Traditional hypervisor development often delays integration and hardware validation, leading to late-stage discovery of timing and partitioning issues. In contrast, the MBSE-driven approach enabled early identification of design flaws, reducing integration effort by approximately 40% (Bruns & Kuschnerus, 2011; Beltrán, García, & de la Iglesia, 2012). Pre-code simulation of scheduling, inter-partition communication, and fault recovery minimized recertification cycles typically caused by hardware-in-the-loop debugging. Additionally, the use of standardized ARINC 653 libraries and automated requirement-to-test traceability reduced manual verification effort by 30% (Bastoni, Brandenburg, & Anderson, 2011). These efficiencies shortened development time, improved system reliability, and lowered lifecycle costs, underscoring the value of MBSE in certifiable, safety-critical virtualization.

6. Discussion

6.1 Implications for Diverse Industries

A model-driven approach to ARINC 653 hypervisor development offers broad applicability across safety-critical domains. In aerospace, early simulation and fault injection support compliance with DO-297 and DO-331, reducing certification effort and enhancing safety margins (Fleming & Leveson, 2014; RTCA, Inc., 2005). Automotive systems benefit from partitioned virtualization by isolating safety-critical functions—such as braking and steering—from non-critical domains, thereby improving functional safety (Tank, Aggarwal, & Chaubey, 2019). In medical devices, MBSE enables pre-deployment validation under hardware and software faults, mitigating patient risk (Gupta & Chen, 2020). For space applications, where post-deployment fixes are infeasible, simulation of launch and orbit scenarios enables robust verification against environmental faults (De Simone & Mazzeo, 2019). Across all sectors, ARINC 653–based virtual prototypes support early fault analysis, resource optimization, and generation of certification artifacts, compressing development cycles and lowering lifecycle costs.

6.2 Future Directions and Potential Improvements

While the proposed framework demonstrates efficacy, future scalability requires further advancements. As system complexity increases, distributed simulation and model abstraction techniques may help mitigate execution bottlenecks (Lukić, Jung, & Härtig, 2024). Semi-partitioned and mixed-criticality scheduling could improve CPU utilization by allowing lower-criticality tasks to occupy unused cycles without jeopardizing high-criticality deadlines (Bastoni, Brandenburg, & Anderson, 2011). Integrating machine learning into the MBSE workflow may also enable predictive fault handling by analyzing

telemetry and historical failure patterns (Gupta & Chen, 2020). Finally, automating dynamic traceability updates across requirements, models, and code is essential to reduce manual effort and preserve certification alignment (RTCA, Inc., 2011b). Advancing these areas will enhance the robustness and efficiency of model-based virtualization for future safety-critical systems.

7 Conclusion

This paper presented a model-based methodology for developing an ARINC 653–compliant hypervisor that aligns with both the partitioning requirements of ARINC 653 (ARINC, 2002) and the model-based development framework of DO-331 (RTCA, Inc., 2011b). Using MATLAB®/Simulink® and the ARINC 653 blockset (MathWorks & SYSGO, n.d.), early-stage simulation facilitated temporal determinism (Fleming & Leveson, 2014) and fault containment (Gupta & Chen, 2020), reducing integration time and enhancing system safety. The approach combines SysML-based requirement modeling, automated code generation via Embedded Coder®, and deployment on PikeOS using CODEO (SYSGO, n.d.-a; SYSGO, n.d.-b), resulting in certification-ready artifacts (RTCA, Inc., 2011a). Verification demonstrated early detection of design issues, streamlining certification and improving reliability (Bastoni, Brandenburg, & Anderson, 2011). Future work will address scalability, explore mixed-criticality scheduling to improve CPU utilization (Bastoni, Brandenburg, & Anderson, 2011), and enhance traceability automation to support evolving safety standards (Lukić, Jung, & Härtig, 2024).

References

ARINC. (2002). *ARINC 653: Avionics application software standard interface* (ARP 653P0-3) [Standard]. SAE International. Retrieved from https://www.sae.org/standards/content/arinc653p0-3/

Bastoni, A., Brandenburg, B., & Anderson, J. H. (2011). Server-based approach for mixed-criticality systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium* (pp. 25–36). https://doi.org/10.1109/RTSS.2011.23

Beltrán, P., García, J., & de la Iglesia, D. (2012). A state-of-the-art survey on real-time issues in embedded systems virtualization. *Journal of Software Engineering and Applications, 5*(4), 278–284. Retrieved from https://www.scirp.org/pdf/JSEA20120400006_31629669.pdf

Bruns, M., & Kuschnerus, M. (2011). *Virtualization for safety-critical, deeply-embedded devices* [Technical report]. Retrieved from https://www.semanticscholar.org/paper/Virtualization-for-safety-critical%2C-deeply-embedded-Bruns-Kuschnerus/85aba39521cd943c91267030ed1d3b75f48c0e56

De Simone, L., & Mazzeo, G. (2019). Isolating real-time safety-critical embedded systems via SGX-based lightweight virtualization. *arXiv preprint arXiv:1909.09486*. Retrieved from https://arxiv.org/pdf/1909.09486

Fleming, C. H., & Leveson, N. G. (2014). Improving hazard analysis and certification of integrated modular avionics. *Journal of Aerospace Information Systems, 11*(6), 397–411. https://doi.org/10.2514/1.I010164

Gupta, M., & Chen, Y. (2020). Model-based safety analysis of virtualized systems. *Safety Science, 124*, 104–112. https://doi.org/10.1016/j.ssci.2020.104112

Leveson, N. G. (2011). *Engineering a safer world: Systems thinking applied to safety*. MIT Press.

Liu, C., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM, 20*(1), 46–61. https://doi.org/10.1145/321738.321743

Lukić, B., Jung, M., & Härtig, H. (2024). *Automated configuration generation and validation for ARINC 653 compliant avionics architectures* [Technical report]. German Aerospace Center (DLR). Retrieved from https://elib.dlr.de/209380/

RTCA, Inc. (2005). *Integrated modular avionics (IMA) development guidance and certification considerations* (DO-297) [Technical report]. Retrieved from https://www.sto.nato.int/publications/STO%20Educational%20Notes/RTO-EN-SCI-176/EN-SCI-176-04.pdf

RTCA, Inc. (2011a). *DO-178C: Software considerations in airborne systems and equipment certification* (Rev. A) [Standard]. Retrieved from https://my.rtca.org/NC__Product?id=a1B36000001IcmrEAC

RTCA, Inc. (2011b). *DO-331: Model-based development and verification supplement to DO-178C and DO-278A* [Standard]. Retrieved from https://my.rtca.org/NC__Product?id=a1B36000001IcfiEAC

Tank, D., Aggarwal, A., & Chaubey, N. K. (2019). Virtualization vulnerabilities, security issues, and solutions: A critical study and comparison. *International Journal of Information Technology & Decision Making, 18*(3), 987–1010. https://doi.org/10.1007/s41870-019-00294-x

Xu, J., & Zhang, X. (2024). A model-based optimization method of ARINC 653 multicore partition scheduling. *Aerospace, 11*(11), 915. https://doi.org/10.3390/aerospace11110915