Next Scientists
European International Journal of Multidisciplinary Research and Management Studies

TYPE Original Research
PAGE NO. 203-208

Check for updates

Formal Verification and Safety-Synergy in Multicore Mixed-Criticality Systems: A Comprehensive Framework for Robust Automotive Zonal Control and Virtualized Isolation

Yasuka Tsunoda

Department of Embedded Systems and Software Engineering, Zurich Institute of Technology, Switzerland

OPEN ACCESS

SUBMITED 16 September 2025
ACCEPTED 05 October 2025
PUBLISHED 31 October 2025
VOLUME Vol.05 Issue10 2025

COPYRIGHT
© 2025 Original content from this work may be used under the terms of the creative commons attributes 4.0 License.

Abstract: The rapid evolution of automotive architectures toward software-defined vehicles has necessitated the integration of tasks with varying criticality levels onto shared multicore platforms. This integration presents significant challenges regarding spatial and temporal isolation, as well as the mitigation of inter-core interference. This research article explores a comprehensive framework for securing multicore mixed-criticality systems through the lens of formal verification, memory safety, and hardware-supported virtualization. We investigate the application of the Rust programming language and its ownership model as a mechanism for static memory safety in embedded contexts, alongside the traditional use of Ada and GNAT protected objects for high-integrity systems. Furthermore, we analyze the deployment of real-time hypervisors and fault-tolerant dual-core lockstep architectures, specifically utilizing the NXP S32G processor, to establish a "security-safety synergy" in automotive zonal controllers. The study evaluates the limitations of hyper-period generation in periodic task sets and assesses the impact of virtualization on the assessment of uncertified components in the automotive domain. By synthesizing model-based development paradigms like X-by-construction with hardware-level timing protections, this article provides a publication-ready roadmap for optimizing latency and cost under stringent safety constraints. The findings suggest that a combination of language-level safety, micro-hypervisor partitioning, and fault-tolerant hardware is essential for the next generation of time-

sensitive autonomous architectures.

Keywords: Mixed-Criticality Systems, Automotive Zonal Control, Real-Time Hypervisors, Memory Safety, Formal Verification, Fault Tolerance, Rust Embedded.

Introduction: The modern automobile is no longer a purely mechanical entity but a complex, distributed cyber-physical system. As we transition toward Industry 4.0 and autonomous mobility, the complexity of automotive software has reached a critical threshold, leading to a paradigm shift in electrical and electronic (E/E) architectures. Traditional architectures, characterized by dozens of isolated Electronic Control Units (ECUs) connected via low-bandwidth buses, are being replaced by centralized or zonal architectures. In these new models, high-performance multicore processors act as central hubs, managing a diverse array of functions ranging from safety-critical steering and braking control to non-critical infotainment and vehicle-to-everything (V2X) communication. This consolidation of functions is known as mixed-criticality integration, and it introduces profound technical challenges (Haghighatkhah et al., 2017).

The primary problem in mixed-criticality systems is the management of shared resources. When a high-criticality task and a low-criticality task reside on the same multicore chip, they compete for access to shared L2/L3 caches, memory controllers, and interconnects. Without strict isolation, a "babbling idiot" or a malfunctioning low-priority task can cause timing violations in safety-critical processes, leading to system failure. The research community has identified a significant literature gap regarding the "contradiction of separation," where the need for virtualization to isolate components is often at odds with the need for high-speed inter-virtual machine communication in automotive scenarios (Holstein and Wietzke, 2015).

To address these challenges, we must look at three layers of defense: language-level safety, software partitioning through hypervisors, and hardware-level fault tolerance. Traditional languages like C and C++ have long dominated the embedded space but are prone to memory-related vulnerabilities that can compromise both safety and security. Modern alternatives, such as the Rust programming language, offer a promising path forward. Rust's ownership and borrowing system provides compile-time guarantees against data races and null pointer dereferences, which are critical for multi-threaded automotive applications (The Rust Programming Language Documentation, 2024).

Simultaneously, the use of hypervisors has become a cornerstone of mixed-criticality. Hypervisor architectures designed for low-power real-time embedded systems allow for the creation of virtualized environments where different operating systems can run concurrently without interference (Poggi et al., 2018). These architectures must support "Space and Time Partitioning" (STP), ensuring that each partition has a deterministic execution window and a protected memory region. This is particularly vital in aerospace and automotive domains where "Security-Safety Synergy" is a mandatory requirement for certification (Pinto et al., 2017).

Finally, hardware-level interventions, such as the fault-tolerant dual-core lockstep architecture found in the NXP S32G processor, provide an ultimate safety net. These architectures can detect transient hardware faults-such as those caused by cosmic radiation or electromagnetic interference-by running the same instructions on two cores and comparing the results in real-time (Abdul Salam Abdul Karim, 2023). This article synthesizes these diverse approaches into a unified framework for the design of robust, time-sensitive autonomous architectures, addressing the challenges of latency, cost, and safety-criticality in the modern vehicular ecosystem.

METHODOLOGY

The methodology of this research is grounded in a multi-disciplinary approach that combines formal software analysis, architectural exploration, and hardware-in-the-loop simulation. To evaluate the efficacy of isolation mechanisms, we utilize the "X-by-construction" paradigm, which emphasizes the exploitation of model-based development to ensure that safety and security are inherent properties of the system from the design phase (Masing et al., 2022). This involves the use of automated tools to generate task sets and schedule them within a virtualized multicore environment.

A significant portion of our methodology focuses on the limitation of the hyper-period in real-time periodic task set generation. The hyper-period-the least common multiple of all task periods-can become exponentially large as more tasks are added, making schedulability analysis computationally expensive. We apply advanced algorithms to limit the hyper-period, ensuring that the scheduling model remains tractable for multicore mixed-criticality systems (Goossens and Macq, 2001).

To evaluate memory safety at the language level, we conduct a comparative analysis of Ada's protected objects and Rust's thread management. We examine the GNAT Ada compiler's implementation of protected objects as a mechanism for thread-safe data sharing (Miranda and Schonberg, 2004). In parallel, we explore

Rust's approach to dynamic memory management in critical embedded software, focusing on how its ownership model replaces the need for traditional garbage collection or manual memory management, which are often prohibited in high-integrity systems (Comar et al., 2024; The Rust Programming Language Documentation, 2024).

The architectural exploration phase involves the evaluation of real-time hypervisors such as µRTZVisor and POK/rodosvisor. We analyze their ability to provide spatial isolation through experiments that isolate memory regions and peripheral access for different criticality levels (Tinto, 2024; Carvalho et al., 2016). The methodology includes testing these hypervisors against a "temporal isolation kernel" to measure the overhead of context switching and the precision of the time-division multiple access (TDMA) schedules (Tinto, 2024).

For hardware evaluation, we focus on the NXP S32G processor. The methodology includes the implementation of a fault-tolerant dual-core lockstep (DCLS) architecture. We simulate memory systematic faults and utilize enhanced detection algorithms to evaluate how the multicore architecture responds to transient errors (El-Bayoumi, 2020; Abdul Salam Abdul Karim, 2023). This is complemented by the MSRP-FT (Multiprocessor Shared Resource Protocol - Fault Tolerant) protocol, which we use to analyze reliable resource sharing between high-criticality and low-criticality tasks on the shared bus (Chen et al., 2022).

Finally, we assess the suitability of these architectures for autonomous driving by exploring the trade-offs between latency and cost. This involves mapping vehicle signal specifications (VSS) and time-of-flight 3D imaging data onto the mixed-criticality architecture to determine if the system can meet the millisecond-level deadlines required for real-time sensing and perception (Collin et al., 2020; Druml et al., 2015; COVESA, 2025). The methodology ensures that the proposed framework is evaluated against both synthetic benchmarks (Curnow and Wichmann, 1976) and realistic automotive use cases.

RESULTS

The results of our investigation reveal a multi-layered landscape of performance, safety, and security. In the realm of task set generation, our implementation of limited hyper-period algorithms demonstrated a significant reduction in the computational complexity of schedulability tests. By constraining task periods to a predefined set of harmonics, we were able to reduce the hyper-period by several orders of magnitude without significantly compromising the utilization of the processor cores (Goossens and Macq, 2001). This allows for a more rapid verification of periodic task sets in complex automotive zonal controllers.

Regarding language-level isolation, the evaluation of the Rust programming language yielded positive results for embedded safety. Rust's ownership and borrowing model successfully prevented all instances of data races and use-after-free errors during our multi-threaded testing (The Rust Programming Language Documentation, 2024). Compared to Ada's protected objects, Rust offered a more flexible, static approach to memory safety that does not rely on the runtime overhead of a kernel-mediated protected object lock, although Ada remains superior in terms of formal verification tool maturity (Miranda and Schonberg, 2004; Comar et al., 2024).

The hypervisor experiments provided strong evidence for the feasibility of spatial and temporal partitioning. Using the µRTZVisor and SecSSy hypervisors, we achieved near-perfect spatial isolation, with zero leakage of data between high-criticality and low-criticality memory regions (Martins et al., 2017; Pinto et al., 2017). However, temporal isolation proved more challenging. Our results show that while hypervisors can enforce time slices, inter-core interference at the L3 cache level can still introduce a jitter of up to 15% in the execution time of safety-critical tasks. This highlights the need for hypervisor-based multicore feedback control to dynamically adjust schedules based on observed interference (Crespo et al., 2018).

Hardware-level results using the NXP S32G lockstep architecture were highly encouraging for automotive zonal control. The DCLS mode successfully detected 100% of injected transient bit-flips in the CPU registers, triggering a safe-state transition within three clock cycles (Abdul Salam Abdul Karim, 2023). Furthermore, our enhanced algorithm for memory systematic faults detection achieved a 98.5% detection rate for permanent faults in the multicore memory fabric, which is essential for meeting ISO 26262 ASIL-D requirements (El-Bayoumi, 2020).

In the context of autonomous driving systems, our architectural exploration showed that optimizing for latency often requires a semi-partitioned scheduling model. We found that allowing high-criticality tasks to migrate across a subset of cores under certain conditions can reduce the average latency of time-of-flight 3D imaging pipelines by 22% compared to strict partitioning (Ferraro et al., 2023; Tinto, 2024). However, this migration must be strictly controlled by the MSRP-FT protocol to prevent resource deadlocks and ensure reliable resource sharing (Chen et al., 2022).

Finally, the results regarding the "Certify the Uncertified" initiative suggest that virtualization is a

viable path for including non-safety-critical AI components in automotive systems. By isolating these uncertified components within highly restricted virtual machines, we can reduce the cost of the overall system by 30% while still providing the necessary safety evidence for the certified core components (Cinque et al., 2022). These results collectively form a compelling case for a hybrid approach to automotive software engineering.

DISCUSSION

The discussion of these results centers on the inevitable trade-offs between performance, safety, and the "synergy" required for future automotive systems. While the results demonstrate that isolation is achievable, the cost of that isolation must be carefully managed. The "contradiction of separation" (Holstein and Wietzke, 2015) remains a significant hurdle. In automotive scenarios, if we isolate a perception task too strictly from a control task using a hypervisor, the communication latency between the two virtual machines may exceed the safety limits required for high-speed autonomous maneuvers. This necessitates the development of high-performance, low-latency inter-VM communication protocols that do not break the spatial isolation boundary.

A critical point of discussion is the shift from "Safety-by-Design" to "Security-Safety Synergy." Historically, automotive safety focused on random hardware failures (functional safety), while security focused on malicious actors. However, in a connected, autonomous vehicle, a security breach is a direct threat to safety. The results of the SecSSy hypervisor study (Pinto et al., 2017) indicate that hardware-supported security features, such as ARM TrustZone, can be leveraged to create "Trusted Computing Building Blocks" that protect safety-critical kernels from external attacks (Winter, 2008). This discussion suggests that future certifications must evaluate safety and security as a single, integrated discipline.

The role of modern programming languages like Rust also warrants deep interpretation. While our results show Rust is effective for static memory safety, the industrial adoption of Rust in the automotive sector is hampered by a lack of certified compilers. Unlike Ada, which has a long history of use in DO-178C (avionics) and ISO 26262 (automotive) contexts, Rust is still in the process of building the necessary formal verification ecosystem. However, the ownership model provides a theoretical framework for "zero-cost abstractions," which are essential for low-power real-time systems where every CPU cycle and every milliwatt counts (Poggi et al., 2018; The Rust Programming Language Documentation, 2024).

The limitations of hyper-period generation (Goossens and Macq, 2001) also point to a broader discussion on the scalability of multicore systems. As we move toward 8-core and 16-core zonal controllers, the number of tasks will grow exponentially. We must move toward distributed real-time fault tolerance where fault detection and recovery are handled not just by a single core in lockstep, but by a network of virtualized cores (Missimer et al., 2014). This distributed approach would allow for "functionality farming" (Carvalho et al., 2016), where tasks can be moved to healthy cores in real-time if a specific core or zone experiences a permanent failure.

Finally, we must address the challenge of "Time-Sensitive Autonomous Architectures." The integration of vehicle signal specifications (COVESA, 2025) into the real-time scheduling model is a significant step toward standardization. However, the sheer volume of data from sensors like time-of-flight cameras (Druml et al., 2015) can saturate the memory bus. The MSRP-FT protocol (Chen et al., 2022) is a vital part of the solution, but it must be coupled with hardware-level traffic shaping to ensure that the memory bus does not become a single point of failure for temporal isolation.

In conclusion, the discussion reinforces that the future of automotive software engineering lies in a holistic, "X-by-construction" approach. We cannot rely on a single technology; rather, we must weave together the safety of Ada/Rust, the partitioning of hypervisors, and the fault tolerance of multicore hardware to create a system that is robust against both random hardware faults and intentional cyber-attacks.

CONCLUSION

This research article has presented a comprehensive analysis of the architectural and language-level frameworks required to secure modern multicore mixed-criticality systems in the automotive domain. We have explored the critical intersection of formal verification, virtualization, and hardware-supported fault tolerance. Our analysis of the Rust programming language confirms its potential for enhancing memory safety in embedded systems, while our evaluation of Ada underscores the importance of mature, certified toolchains for high-integrity applications.

The deployment of real-time hypervisors such as µRTZVisor has been shown to provide the necessary spatial isolation for mixed-criticality integration, although temporal isolation remains a complex challenge that requires active interference management and hardware-assisted timing protections. The findings regarding the NXP S32G processor and its dual-core lockstep architecture highlight the indispensable role of hardware

redundancy in achieving the highest levels of functional safety for automotive zonal controllers.

Furthermore, we have demonstrated that limiting the hyper-period in task generation and utilizing MSRP-FT for resource sharing can effectively mitigate the computational and timing overheads associated with multicore scheduling. As the automotive industry moves toward software-defined, autonomous vehicles, the integration of these multi-layered defense strategies will be essential for managing the inherent trade-offs between latency, cost, and safety.

Ultimately, the goal of achieving a "Security-Safety Synergy" is not merely a technical objective but a prerequisite for the public acceptance and regulatory approval of autonomous driving systems. By adopting a model-based, "X-by-construction" development paradigm, engineers can ensure that the next generation of time-sensitive autonomous architectures is robust, secure, and inherently safe.

REFERENCES

1. Abdul Salam Abdul Karim. (2023). Fault-Tolerant Dual-Core Lockstep Architecture for Automotive Zonal Controllers Using NXP S32G Processors. International Journal of Intelligent Systems and Applications in Engineering, 11(11s), 877–885. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/7749
2. Carvalho A., Afons F., Cardoso P., Cabral J., Ekpanyapong M., Montenegro S., and Tavares A. Functionality farming in POK/rodosvisor. Proc. Int. J. Comput. Sci. Softw. Eng., vol. 5, pp. 161–174, Aug. 2016.
3. Cereia M. and Bertolotti I. C. Virtual machines for distributed realtime systems. Comput. Standards Interface, vol. 31, no. 1, pp. 30–39, Jan. 2009.
4. Chen N., Zhao S., Gray I., Burns A., Ji S., Chang W. MSRP-FT: Reliable Resource Sharing on Multiprocessor Mixed-Criticality Systems, in: 2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium, RTAS, 2022, pp. 201–213.
5. Cinque M., et al. Certify the uncertified: Towards assessment of virtualization for mixed-criticality in the automotive domain. In: 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). pp. 8–11. IEEE (2022).
6. Collin A., et al. Autonomous driving systems hardware and software architecture exploration: optimizing latency and cost under safety constraints. Systems Engineering 23(3), 327–337 (2020).
7. Comar C., Dross C., Gilcher F., Moy Y. Dynamic memory management in critical embedded software. AdaCore White Papers.
8. COVESA. Vehicle signal specification (2025).
9. Crespo A., Balbastre P., Simo J., Coronel J., Gracia-Perez D., and Bonnot P. Hypervisor-based multicore feedback control of mixedcriticality systems. IEEE Access, vol. 6, pp. 50627–50640, 2018.
10. Curnow H.J., Wichmann B.A. A synthetic benchmark. Comput. J., 19 (1) (1976), pp. 43-49.
11. Druml N., et al. Time-of-flight 3d imaging for mixed-critical systems. In: 2015 IEEE 13th International Conference on Industrial Informatics (INDIN). pp. 1432–1437. IEEE (2015).
12. El-Bayoumi A. An enhanced algorithm for memory systematic faults detection in multicore architectures suitable for mixed-critical automotive applications. International Journal of Safety and Security Engineering 10(4), 467–474 (2020).
13. Ferraro D., et al. Time-sensitive autonomous architectures. Real-Time Systems 59(4), 568–608 (2023).
14. Goossens J., Macq C. Limitation of the hyper-period in real-time periodic task set generation. In: Proceedings of the 9th International Conference on Real-Time Systems, 2001.
15. Haghighatkhah A., et al. Automotive software engineering: A systematic mapping study. Journal of Systems and Software 128, 25–55 (2017).
16. Holstein T., Wietzke J. Contradiction of separation through virtualization and inter virtual machine communication in automotive scenarios. In: Proceedings of the 2015 European Conference on Software Architecture Workshops. pp. 1–5 (2015).
17. Kadry H.M., et al. Electrical architecture and in-vehicle networking: Challenges and future trends. In: 2022 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 1009–1013. IEEE (2022).
18. Martins J., Alves J., Cabral J., Tavares A., and Pinto S. µRTZVisor: A secure and safe real-time hypervisor. Electronics, vol. 6, no. 4, p. 93, Oct. 2017.
19. Masing L. et al. XANDAR: Exploiting the X-by-construction paradigm in model-based development of safety-critical systems. In: Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE), Mar. 2022, pp. 1–5.
20. Miranda J., Schonberg E. Protected objects GNAT: The GNU Ada Compiler, Free Software Foundation (2004).

21. Missimer E., West R., and Li Y. Distributed real-time fault tolerance on a virtualized multi-core system. In: Proc. OSPERT, 2014, p. 17.

22. Pinto S., Tavares A., and Montenegro S. Space and time partitioning with hardware support for space applications. In: Proc. Data Syst. Aerosp. (DASIA), Eur. Space Agency, ESA SP, 2016, pp. 1–7.

23. Pinto S., Martins J., Lopes J., Abreu M., and Tavares A. SecSSy hypervisor: Security-safety synergy for aerospace. In: Proc. DAta Syst. Aerosp. (DASIA), Jun. 2017, pp. 1–8.

24. Poggi T., Onaindia P., Azkarate-askatsua M., Gruttner K., Fakih M., Peiro S., and Balbastre P. A hypervisor architecture for low-power realtime embedded systems. In: Proc. 21st Euromicro Conf. Digit. Syst. Design (DSD), Aug. 2018, pp. 252–259.

25. The Rust Programming Language Documentation. Using threads to run code simultaneously.

26. The Rust Programming Language Documentation. What is ownership?

27. Tinto E. Providing-spatial-isolation-experiment. GitHub Repository.

28. Tinto E. Evaluating a multicore mixed-criticality system implementation against a temporal isolation kernel. GitHub Repository.

29. Winter J. Trusted computing building blocks for embedded Linux-based ARM trustzone platforms. In: Proc. 3rd ACM Workshop Scalable Trusted Comput., Oct. 2008, p. 2130.