Elektrobit

Tech paper

Enhancing automotive HPC safety with Linux: a comprehensive overview

Summary of white paper contents

Enhancing automotive HPC safety with Linux: a comprehensive overview

Synopsis: Architecture, stage of development, and concept of **EB corbos Linux for Safety Applications**. The reader will learn how Linux and safety for HPC in vehicles works.

Author: Joachim Schlosser

1. Introduction

1.1 Functional safety matters

Since the 1990s, the automotive industry has witnessed a proliferation of electronic control units (ECUs) in vehicles, driven by increasing system complexity. Features such as anti-lock braking systems, traction control, and advanced driver assistance systems (ADAS) demanded dedicated ECUs, allowing for optimization, and ensuring efficient performance and reliability.

This modular approach to ECU design facilitated easier integration of new features without overhauling the entire system, offering flexibility and cost-effectiveness in sourcing, assembly, and maintenance. Moreover, dedicated ECUs for critical functions such as braking and airbag deployment significantly improved vehicle safety and reliability by isolating these functions and reducing the risk of system failures.

Regulatory requirements and industry standards have further propelled the adoption of multiple ECUs in vehicles. Mandates for emissions control, safety, and cybersecurity often stipulated the use of dedicated ECUs for specific functions. Additionally, industry standards and protocols facilitated interoperability and compatibility between different ECUs and vehicle systems, ensuring seamless integration and operation. With more and more software functionality, high-performance computing (HPC) systems now aim to bundle functionality of many ECUs into a single system-on-a-chip (SoC) or SoC cluster within less, but larger ECUs.

Functional safety remains paramount in automotive high-performance computing (HPC) systems, due to their integral role in managing both safety-critical and non-safety-critical applications. Compliance with the applicable standards and regulations, such as ISO 26262 [1] for functional safety is required for HPC systems to achieve the required reliability, functional safety, security and especially cybersecurity, privacy, and data protection.

1.2 Automated driving increases the need for safety-relevant software

Automated driving, also known as (partially) autonomous driving or self-driving technology, significantly increases the amount of safety-relevant software in vehicles for several main reasons:

Complexity of control systems. The intricate interactions among components such as complex sensors – cameras or radars – and control units drive the increasing complexity of automated driving systems.[1] Substantial data volumes from complex sensors require advanced and intensive data processing. The integration of diverse complex sensors and advanced data processing require the co-operation of many skilled interdisciplinary teams. Making complex control systems functionally safe is inherently complex, too.

Object recognition and situation interpretation. AD/ADAS systems need to first build a model of an uncertain and changing environment (the road, other drivers, pedestrians, the vehicle itself …). This, in turn, requires collecting and processing information from different sensors and aggregating them into that model, and then using the model to make decisions about how to drive the vehicle. This process requires extensive and intensive data acquisition and processing, including images and signals.

Redundancy and fail-operational mechanisms. Safety-critical systems in automated vehicles often incorporate redundancy and fault detection and negation mechanisms to ensure availability and functional safety.

For example, redundant sensor arrays and redundant processing units are employed to cross-validate data and detect/negate/mitigate potential failures.[2] Sophisticated fault detection, isolation, and recovery algorithms are often used to detect anomalies, thus ensuring safe operation.[3–5]

In ADAS systems up to SAE Level 2,[6] the driver is always the ultimate supervisor and holds final responsibility for the vehicle, even if the system fails. From SAE Level 3, the system bears the full responsibility for the safety of the vehicle, while the driver is not required to supervise its operations. For this reason, systems are required to be able to perform an emergency function even after a fault, at least for a limited time (for instance, until the vehicle has been parked). While for fail-safe systems, redundancy can be used to detect faults, for fail-operational systems it also allows the availability required to perform an emergency function in case of a fault or failure. And of course, this applies to our example of a digital cockpit, where we also need to have a safe state and/or fallback mechanisms.

Real-time decision-making. Automated driving systems must quickly make a decision in dynamic and unpredictable driving environments to ensure safe driving. This requires highly responsive and accurate software capable of processing, in real time, vast amounts of data through complex algorithms.[7] The software must account for factors such as road conditions, traffic patterns, weather conditions, and the behavior of other road users to make appropriate driving decisions and avoid accidents.[8]

Verification and validation. The development of safety-critical software for automated driving requires proper verification and validation activities to ensure the risk of residual errors complies with the applicable norms and regulations. This includes testing in simulated and real-world environments, involving many different scenarios with both virtual [12-14] and real [11] driving data.

Automated driving technology brings various safety-related software components into vehicles. These components include everything from perception algorithms to fail-operational mechanisms. Additionally, there are verification/validation processes to ensure that all these components work correctly. Ensuring the safety and reliability of these software systems remains paramount as the automotive industry advances towards autonomous driving. [9]

1.3 High-performance computing and applicability to AD/ADAS and cockpit

While HPC systems support safety-critical functions such as autonomous driving and collision avoidance, cockpit/in-vehicle infotainment (IVI) systems as use cases for HPC [10] primarily focus on non-safety-critical functions such as entertainment and GPS navigation, but also perform safety-related functionalities. Regardless of their level of criticality, both cases share the same challenges related to system complexity, interconnectivity, and potential failure modes, necessitating robust functional safety measures to ensure reliable operation.

Fig. 1. Edge-to-edge digital cockpit as shown at CES 2024 [10]

A cockpit system is a good example. Intensive computation resources are required for all the screen real estate, and might lead towards needing an HPC. It is worth noting that the cockpit can also be used to show safety-critical information, which then requires to take functional safety into account. In vehicle technology, both automated driving and advanced driver assistance systems (ADAS) involve the cockpit, which constitutes the human-machine interface and as such is critical for the overall vehicle control.

2. The role of Linux in automotivesystems

2.1 Understanding and defining the Automotive OS

One of the most common terms currently used by carmakers is Automotive OS (automotive operating system). [11] It gives the impression that a car actually has a single operating system, like your mobile phone or your PC, and is the incarnation of the 'software defined vehicle' – another term broadly used in the automotive world currently. Yet, we all know that a car in fact contains dozens of individual ECUs, all running their own software, held together via multiple networks such as CAN or Ethernet using common communication standards mostly defined by AUTOSAR [12]. How should such a collection of ECUs act as a single automotive operating system? And how should such a scattered infrastructure be safe and secure [13]?

To address these questions, we need to start with a definition of the term 'Automotive OS'. Elektrobit uses three statements to define the properties of an Automotive OS:

“The Automotive OS is a software platform that abstracts the complex vehicle network of ECUs as one device.”[14]

This means that for the Automotive OS we are not talking about a single operating system. In fact, it is a software layer to hide the complex collection of ECUs with its communication matrices using a defined interface. This allows applications such as the infotainment and cluster systems to interact with drivers and passengers, but also enables cloud connectivity to access the car's functionality without knowing the vehicle's network structure. This increases the portability between car models and different generations of the software platform.[15, 16]
“The Automotive OS manages, supervises, and updates the device.” [14]

Of course, the software platform that makes up the Automotive OS knows about the internal network of the vehicle. It makes sure that individual components are configured correctly, operating in the right state, and acts as an interface to the outside world. It also makes sure that all components are up to date and run on compatible software versions.

“The Automotive OS enables an ecosystem by harmonizing the APIs against which functions are developed.” [14]

While the first two statements already improve the maintainability of a carmaker's software platform significantly, this statement summarizes the full power of the Automotive OS concept: “We need to provide an open and defined interface to applications that can function between different car models, brands, and even generations – such as your mobile phone apps that run on multiple OS versions and various devices.”[17]

2.2 Advantages of using Linux in automotive systems

GNU/Linux, an open-source operating system, offers numerous advantages for automotive systems. It provides customizable features and functionalities, reliability, and stability across various use cases such as server environments, cloud computing, HPC, and Android devices.

Large developer pool. One major advantage of GNU/Linux is its large developer pool. Together with the large number of users and user organizations, this contributes to continuous improvement and providing valuable resources, documentation, and support. Well-known APIs simplify development and integration, reducing time and effort. Additionally, the open-source nature eliminates vendor lock-in, enabling organizations to choose from multiple suppliers and ensuring flexibility even if the original supplier becomes unavailable.

Rapid evolution cycle. GNU/Linux benefits from a rapid evolution cycle, with regular updates, bug fixes, and feature enhancements keeping the operating system up to date.

Compatibility. Compatibility with a wide range of hardware architectures allows automotive manufacturers to select components based on specific requirements without immediate software compatibility issues.

Source code transparency. Moreover, complete source code transparency in GNU/Linux facilitates efficient debugging and troubleshooting, potentially leading to faster bug fixes and improved system stability. This transparency enhances the overall maintainability of automotive systems built on GNU/Linux.

2.3 Advantages of using a specialized Linux distribution in automotive over enterprise distributions or embedded Linux build system

Choosing a Linux distribution over an embedded Linux build system such as Yocto [18] for automotive applications warrants consideration due to several key factors, with maintenance standing out as a critical consideration. While Yocto offers flexibility and customization options for embedded systems, it requires substantial expertise and resources to maintain and update it effectively. In contrast, a Linux distribution tailored for automotive use streamlines maintenance processes by providing pre-configured components and support for automotive-specific features, reducing the burden on development teams, and ensuring system security over the vehicle's life cycle.

It also needs to be considered that a general-purpose Linux distribution does not match the specific needs of the automotive domain because of its excessive storage and memory requirements. Until now this has been addressed by simply minimizing the number of packages, which is not an effective solution because it adversely affects the functionality.
Moreover, a dedicated automotive Linux distribution offers targeted optimizations and enhancements for automotive use cases, automotive-grade security features and should comply with the applicable standards and regulations such as ISO 26262. This specialized solution **EB corbos Linux for Safety Applications** comes equipped with software stacks and tools tailored for automotive development, enabling faster time to market and reducing development costs.

3. Functional safety standards and regulations in the automotive industry

3.1 Overview of relevant safety standards and regulations

Relevant automotive safety standards and regulations play a crucial role in ensuring the safety and reliability of vehicles on the road. One of the most prominent standards is ISO 26262, which outlines the functional safety requirements for electrical and electronic systems in road vehicles.[19] ISO 26262 provides a comprehensive framework for the development, implementation, and verification of safety-critical systems throughout the automotive supply chain. It covers various aspects, including hazard analysis and risk assessment, safety goals definition, hardware and software development processes, and verification and validation activities, all aimed at minimizing the risk of system failures and ensuring vehicle safety.

On a software source code level, for automotive this usually includes adhering to MISRA-C and derivatives. [20] While there are coding guidelines for GNU/Linux [21, 22] available, those are not geared towards embedded let alone functional-safety-relevant systems.

In addition to ISO 26262, automotive manufacturers must comply with regional safety regulations and standards specific to their target markets. If companies want to put an electronic or software system on the road, they must take functional safety into account. Product liability laws, even when not explicitly demanding compliance to certain functional safety standard, hold OEMs accountable for ensuring their systems are developed with state-of-the-art technology including verification and validation.

3.2 Impact of functional safety standards on Linux-based automotive systems

“The Linux kernel, as a prominent open-source component, attracts significant attention from security researchers and the broader developer community. Its transparent nature allows for in-depth analysis, leading to a higher reporting rate of security vulnerabilities compared to closed-source software.” [23]

This makes for a solid cybersecurity performance but does not yet talk about functional safety. Functional safety is another axis of consideration. The Linux kernel was not designed for vehicle systems to operate reliably and safely, even in the presence of malfunctions or errors, to prevent accidents and protect passengers and road users. Even more, the GNU system around the kernel was even less designed for this purpose. This makes for a solid cybersecurity performance but does not achieve the required functional safety. The Linux kernel was not designed to operate reliably and safely and even less to handle malfunctions or errors. The GNU system around the kernel was even less designed for safety-critical and mission-critical applications. As a consequence, it would not be realistically possible to specify and demonstrate the functional safety of GNU/Linux using the methods, techniques, and procedures that are commonly applied to achieve and demonstrate the compliance with the applicable functional safety norms and standards.

Making GNU/Linux suitable for functional-safety-relevant applications requires a very innovative approach and thinking 'out of the box'.

4. Safety solution for Linux in automotive

Let's take a closer look at **EB corbos Linux for Safety Applications** as the solution for safety-relevant systems automotive HPC system based on open source.

A TÜV assessed safety architecture for an HPC system with Linux for automotive applications

The safety concept involves the integration of safety measures within the software of the ECU to ensure the validity of critical systems in automotive applications.

In order to show the potential and benefits of its solution, Elektrobit has developed an example system. The example system displays a video stream generated by the vehicle's sensors or cameras, providing visual information about the vehicle's surroundings. The video stream and other relevant data are shown on display screens, showing the software behavior and system responses, allowing viewers to understand the system's behavior. The software includes a safety monitoring system that continuously analyzes a video feed for errors or anomalies. If any issues are detected, the system takes appropriate action, such as shutting down the system to prevent potential safety hazards. The demo includes a mechanism for inducing error scenarios by injecting faults into the video feed or the Linux kernel, emulating real-world situations where the system must detect and respond to anomalies.

An exemplary architecture for a system is illustrated in *Fig. 2*. In this example, there is another operating system partition integrated in the HPC, using an Android framework serving special display use cases for which the OEM might want to use existing software modules. The shaded, outlined areas show the safety integrity level for which the components are being developed and run.

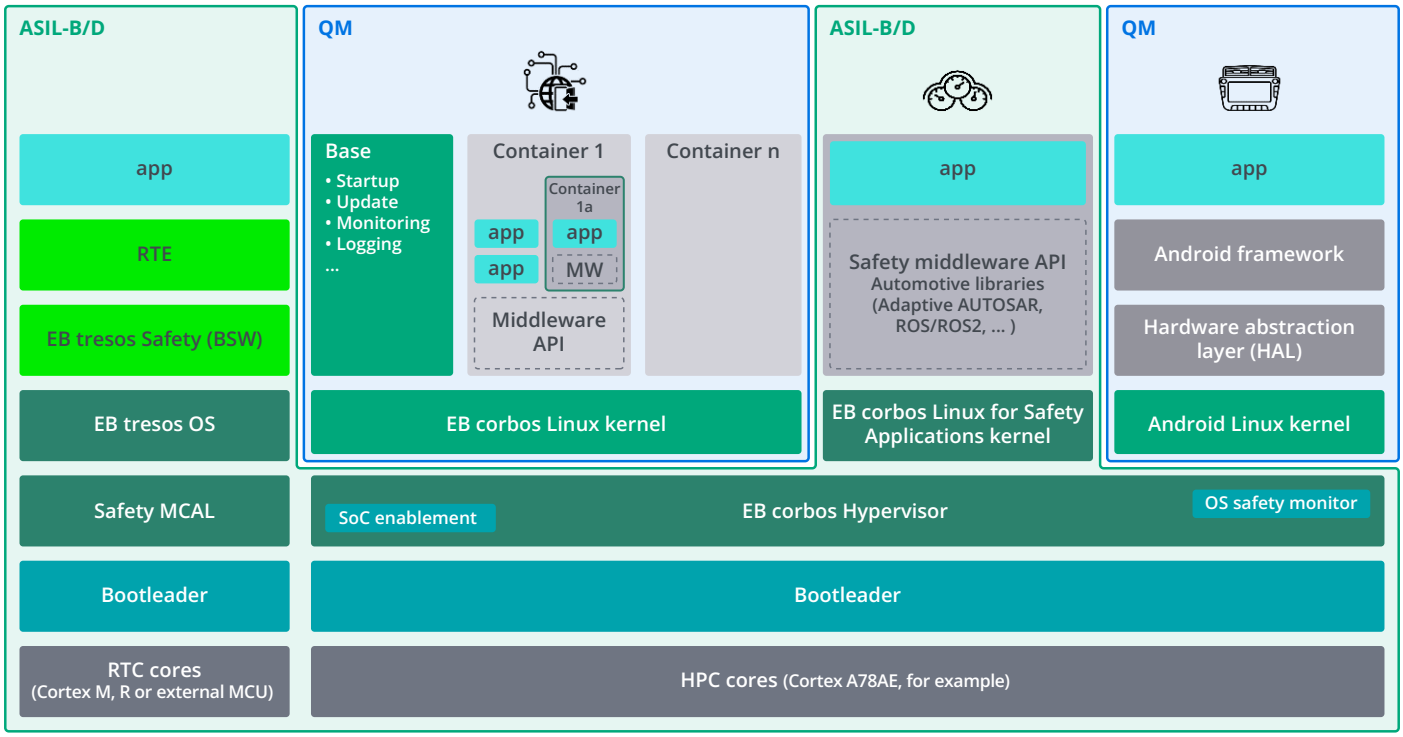

Fig. 2. Example integration cockpit + IVI [24

The illustration of an exemplary digital cockpit architecture shows the different hardware and software layers. Starting from bottom to top, the system-on-a-chip (SoC) uses one or more RTC cores such as the ARM Cortex M, R, or an external MCU. The HPC cores will usually comprise ARM Cortex A cores, such as the Cortex A78AE, for example. RTC and HPC cores each have their own bootloader.

In this example, the RTC section lays the safety-enabled Classic AUTOSAR operating system **EB tresos Safety OS** [25] with its Safety MCAL on top of the bootloader, but could also have the dedicated real-time **EB tresos Embedded Hypervisor** [26] below, in case multiple Classic AUTOSAR instances are needed.

The dedicated HPC hypervisor **EB corbos Hypervisor** [27] with the specific SoC enablement and the OS safety monitor serves as the basis for multiple HPC operating system instances. The hypervisor and OS safety monitor are safety components and developed in full accordance with relevant quality [28] and safety standards [19, 29]. A non-safety (quality-managed (QM)) Linux partition runs all regular non-safety-related functions, e.g. as performance applications in a containerized environment. [30, 31] In the example of the digital cockpit, the Android framework also runs as a non-safety-related partition, for use cases as mentioned above. Finally, the **EB corbos Linux for Safety Applications** kernel [32] is a separate virtual machine on a hypervisor, with the safety middleware API, such as **EB corbos Adaptive** [33] or ROS [34], as basis for performance safety applications that are ISO 26262 ASIL-A/B enabled.

This concept encompasses several key aspects:

1. **Error monitoring**: The system continuously monitors for errors, particularly in the example's video stream, which is critical for various automotive functionalities.

2. **High-integrity partition**: A high-integrity partition within the ECU is responsible for generating and monitoring the video feed. It ensures that the video stream promptly detects any errors or anomalies.

3. **OS safety monitor:** The OS safety monitor, aka supervisor, oversees the functions of the high integrity partition, ensuring its proper operation. It also monitors for faults within the partition and triggers shutdown procedures if necessary.

Overall, this safety concept aims to create a robust system that can detect and respond to potential hazards or malfunctions, enhancing the safety and reliability of automotive systems. Higher integrity levels (ASIL) are well possible by functional decomposition [35, 36] or multichannel approaches. In ISO 26262 context, functional decomposition involves breaking down ASIL-D functions, which have the highest safety requirements, e.g., into redundant ASIL-B(D) requirements.[37Ch. 5]. These ASIL-B(D) requirements then can be implemented in ASIL-B components with sufficient freedom from interference.[37Ch. 6–7]

EB corbos Linux for Safety Applications is protected by patents due to its technological approach which decouples OSS from the supervisor.

4.1 Functional safety-relevant failure scenarios

The failure scenarios demonstrated with the example system introduced above involve inducing errors in the video stream or directly in the monitor.

Nominal use case: A performance application runs on our Linux distribution [32] and processes image data. A safety-critical application runs on the safety derivative of the Linux distribution, in a second hypervisor virtual machine, and supervises the performance application. The supervisor protects the data integrity of the safety-critical application.

Fault within the performance application: In the first scenario, an error is injected into the video feed, causing the video stream generator to transmit an invalid image to the safety application. The system detects the error in the video feed, leading to the display of a red indicator and subsequent system shutdown.

Kernel fault corrupting the safety-critical application: When the kernel of the safety-enabled Linux partition, for example, performs an access to an unauthorized memory area, a dedicated safety layer of our operating system solution, called supervisor, detects this, and notifies the external user interface, running on the PC, via GPIO. It should be noted that functional safety is a system property, not an inherent feature of any of its components. Each function by its very nature results in specific functional safety requirements for the chain of influence of this function throughout the complete system, and thus creates requirements on the system architecture, resulting in requirements on the hardware as well as software and software architecture.

While in the classic ECU world as mentioned above, this led to specialized hardware for many functions, fulfilling specific functional safety needs amongst others, the strong need and desire to reduce variants and increase centralization as well as the reduction of hardware complexity, drive the E/E architecture evolution towards architectures that allow for various functional safety concepts.

Both scenarios highlight the system's ability to detect errors and initiate shutdown procedures, highlighting the robustness of safety measures integrated into the system.

4.2 Integration of safety measures into the Linux-based automotive software stack

The goal is for the safety application to execute correctly and provide appropriate notification otherwise. Safety extensions within **EB corbos Linux for Safety Applications** incorporate aspects of keeping the system safe:

Supervise and legitimate user-space initialization with its processes. When a Linux system boots up, various user-space components need to be initialized to enable the system to function properly. This initialization process involves loading necessary libraries, setting up environment variables, and executing startup scripts or binaries. Legitimate in this context implies that the initialization process follows established conventions, respects security mechanisms, and adheres to best practices. In particular, specialized startup and service initialization modules such as crinit [38] and cominit [39] replace non-automotive ready script-based solutions.

Separate read/write/exec rights on memory regions in between kernel and applications. In Linux, the kernel does not have write or execute rights on the memory allocated to the applications, which means that the memory allocated to the application and its content cannot be modified by the Linux kernel, which in turns protects the application.

Supervise and legitimate any read/write/exec attempt on the application memory. This means in addition to the different write and execute rights described above, any attempt to access the memory allocated to the application is supervised and allowed only if it is legitimate. The goal here is to detect any undue or improper access to the memory allocated to the application and give notice if such an attempt happens.

Supervise and allow or prevent updates on registers holding the processor state throughout context switches. This ensures proper handling of process switching and prevents unauthorized modifications, enhancing system stability and security.

With these extensions, **EB corbos Linux for Safety Applications** ensures there will be no improper access to memory sections that belong to a safety application by the Linux kernel, and there will be no uncontrolled access to memory sections that belong to a safety application via memory shared with other partitions.
The safety-assessed hypervisor partitions resources for safety and non-safety workloads. The hypervisor’s operating system safety monitor is responsible for the external monitoring of the **EB corbos Linux for Safety Applications** kernel.

5. Best practices and recommendations

Contrary to ECU design in earlier decades, where after SOP of the vehicle there were little to no updates ever to an ECU's software, an HPC system design is not a one-time activity, but a long-term continuous development project.

All software development for **EB corbos Linux for Safety Applications** virtual machine partitions must follow the Safety Manual. This does not apply to non-safety EB corbos Linux, which has no restrictions. The supervision-concept does scale; further allowed functionalities will be added throughout the upcoming versions.

The adoption of open-source software in the automotive industry brings both opportunities and challenges. In this section, we draw conclusions based on the discussions presented throughout the paper, highlighting the key points to consider when evaluating the use of GNU/Linux and open-source software in automotive systems.

It is important to acknowledge that open-source software is not a drop-in replacement for proprietary software, particularly when considering the implications of long-term maintenance. While open-source software offers numerous advantages such as flexibility, transparency, and a large developer pool, it also requires careful evaluation and consideration of specific application requirements.

There are no easy answers or 'one size fits all' solutions when it comes to selecting between proprietary software and GNU/Linux. Each application needs to undergo a thorough analysis, considering the pros and cons of adopting GNU/Linux, particularly in view of the long-term implications. Factors such as functional requirements, compliance regulations, and industry standards must be considered to determine the suitability of open-source software for a specific application.[23]

While open-source software is often associated with cost savings, it is essential to consider that the total cost of ownership may vary depending on the application's needs [15]. Free software obtained directly from open-source projects may lack warranties and guarantees, which can impact its suitability for use in critical automotive systems. Therefore, sourcing from organizations dedicated to GNU/Linux as a product can provide additional support and ensure long-term maintenance, potentially saving significant resources and mitigating risks. Companies with deep experience in automotive software and Linux such as Elektrobit can help them, offering EB corbos Linux [16], based on Canonical Ubuntu [17, 18].

Thoroughly evaluate the needs and priorities of each application when considering the adoption of GNU/Linux. Consider the long-term implications, compatibility with existing systems, and the availability of dedicated GNU/Linux suppliers. Engage with suppliers that have a proven track record in delivering long-term maintained GNU/Linux systems suitable for the automotive industry, such as Elektrobit. This will help ensure ongoing support, bug fixes, and security updates.

6. Conclusion

Keeping an HPC system functionally safe is by no means an easy task and shall not be underestimated. This applies to GNU/Linux as operating system for the HPC, too. Effectively, **EB corbos Linux for Safety Applications** achieves the required level of functional safety without placing any reliance on the Linux kernel itself, and so using the latest kernel is possible. This allows for future-proof architecture and performance in your vehicles' and vehicle platform's architecture.

Going HPC, you need to seriously consider functional safety aspects early on. **EB corbos Linux for Safety Applications** is the way to go.

7. References

[1] S. Pruisken, “An end-to-end approach for mastering rising software complexity,” presented at the Autonomous Driving Meetup Munich, Munich, Sep. 19, 2019. Available: https://autonomous-driving.org/wp-content/uploads/2019/10/Elektrobit_Mastering-Software-Complexity__ADM11.pdf. [Accessed: Feb. 15, 2024]

[2] T. Ishigooka, S. Honda, and H. Takada, “Cost-Effective Redundancy Approach for Fail-Operational Autonomous Driving System,” in 2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC), Singapore: IEEE, May 2018, pp. 107–115. doi: 10.1109/ISORC.2018.00023. Available: https://ieeexplore.ieee.org/document/8421154/. [Accessed: Feb. 15, 2024]

[3] AUTOSAR, “Explanation of Error Handling on Application Level.” Nov. 23, 2023. Available: https://www.autosar.org/fileadmin/standards/R23-11/CP/AUTOSAR_CP_EXP_ApplicationLevelErrorHandling.pdf

[4] Lucian Badescu, “Software for fail-operational systems in autonomous vehicles,” Elektrobit, Jun. 30, 2023. Available: https://www.elektrobit.com/blog/software-for-fail-operational-systems-in-autonomous-vehicles/. [Accessed: Feb. 15, 2024]

[5] T. Schmid, S. Schraufstetter, and S. Wagner, “An Approach for Structuring a Highly Automated Driving Multiple Channel Vehicle System for Safety Analysis,” in 2018 3rd International Conference on System Reliability and Safety (ICSRS), Barcelona, Spain: IEEE, Nov. 2018, pp. 362–367. doi: 10.1109/ICSRS.2018.8688859. Available: https://ieeexplore.ieee.org/document/8688859/. [Accessed: Feb. 15, 2024]

[6] “J3016:2018: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International.” Available: https://www.sae.org/standards/content/j3016_201806/. [Accessed: Nov. 17, 2020]

[7] J. Ren and D. Xia, “Autonomous Driving Operating Systems,” in Autonomous driving algorithms and Its IC Design, Singapore: Springer Nature Singapore, 2023, pp. 245–261. doi: 10.1007/978-981-99-2897-2_11. Available: https://link.springer.com/10.1007/978-981-99-2897-2_11. [Accessed: Feb. 15, 2024]

[8] A. K. Jägerbrand and J. Sjöbergh, “Effects of weather conditions, light conditions, and road lighting on vehicle speed,” SpringerPlus, vol. 5, no. 1, p. 505, Dec. 2016, doi: 10.1186/s40064-016-2124-6

[9] J. Schlosser and J. Petersohn, “Considering Linux for functional safety relevant system architecture: Pitfalls and Potential,” in ATZlive Automatisiertes Fahren 2024, Frankfurt am Main: Springer Nature, Mar. 2024.

[10] “Digital Cockpits Solutions for Automotive,” Elektrobit. Available: https://www.elektrobit.com/services/software-engineering/hmi/cockpit-system-solutions/. [Accessed: Feb. 15, 2024]

[11] A. Mattausch, J. Schlosser, and M. Neukirchner, “E/E Architectures and the Automotive OS,” in 23rd Stuttgart International Symposium - Automotive and Engine Technology Documentation, Wiesbaden: Springer Vieweg, 2023, pp. 175–183. Available: https://link.springer.com/chapter/10.1007/978-3-658-42048-2_13

[12] “Standards AUTOSAR.” Available: https://www.autosar.org/standards. [Accessed: Apr. 03, 2024]

[13] United Nations Economic Commission for Europe, “UN Regulation No. 155 - Cyber security and cyber security management system, E/ECE/TRANS/505/Rev.3/Add.154.” 4. Available: https://unece.org/transport/documents/2021/03/standards/un-regulation-no-155-cyber-security-and-cyber-security

[14] M. Neukirchner, “Defining the Automotive OS,” LinkedIn Post, Apr. 08, 2022. Available: https://www.linkedin.com/posts/dr-moritz-neukirchner-6a957b19_defining-the-automotive-os-activity-6918185360971722752-rnFI/. [Accessed: Apr. 11, 2023]

[15] M. Neukirchner, “Automotive OS Fulfills Its Promise,” EE Times, Mar. 06, 2023. Available: https://www.eetimes.com/automotive-os-fulfills-its-promise/

[16] J. Schlosser, A. Mattausch, M. Neukirchner, and R. Holve, “Adaption des Software-Qualitätsmanagements im Automotive-Bereich für eine Nutzung von Fremdkomponenten,” in Software Engineering 2023 Workshops, I. Groher and T. Vogel, Eds., Bonn: Gesellschaft für Informatik e.V., Feb. 2023, pp. 78–91. doi: 10.18420/SE2023-WS-10. Available: http://dl.gi.de/handle/20.500.12116/40195

[17] M. Neukirchner, “There is no software-defined vehicle without an Automotive OS. Period.,” The Automotive OS perspective, Dec. 15, 2022. Available: https://www.linkedin.com/pulse/software-defined-vehicle-without-automotive-os-period-neukirchner/. [Accessed: Apr. 11, 2023]

[18] Linux Foundation, “The Yocto Project,” The Yocto Project. Available: https://www.yoctoproject.org/. [Accessed: Jan. 30, 2024]

[19] “ISO 26262:2018: Road Vehicles – Functional Safety. Part 1-12,” International Organization for Standardization, Geneva, ISO 26262:2018, Dec. 2018.

[20] Motor Industry Software Reliability Association, Ed., MISRA-C:2004: guidelines for the use of the C language in critical systems, 2. ed, [Stand]: October 2004. Nuneaton: MIRA, 2008.

[21] “GNU Coding Standards,” Jul. 01, 2021. Available: https://www.gnu.org/prep/standards/standards.html. [Accessed: Mar. 06, 2024]
[22] “Linux kernel coding style,” The Linux Kernel documentation, 2016. Available: https://www.kernel.org/doc/html/v4.10/process/coding-style.html. [Accessed: Mar. 06, 2024]
[23] J. Schlosser and J. Petersohn, “Maintaining Open-Source based Software or What is the true cost of free?,” in ELIV 2023: Electric / Electronics for Commercial Vehicles 2023, VDI Wissensforum, Ed., in VDI-Berichte, vol. 2423. Düsseldorf: VDI Verlag, 2023, pp. 267–280. doi: 10.51202/9783181024232-267. Available: https://doi.org/10.51202/9783181024232-267
[24] J. Petersohn and M. Wiegand, “EB corbos Linux – built on Ubuntu. Product Description,” Erlangen, Dec. 29, 2023.
[25] “Functional Safety ISO 26262 for Automotive - EB tresos Safety,” Elektrobit. Available: https://www.elektrobit.com/products/ecu/eb-tresos/functional-safety/. [Accessed: Apr. 20, 2023]
[26] “EB tresos Embedded Hypervisor,” Elektrobit. Available: https://www.elektrobit.com/products/ecu/eb-tresos/embedded-hypervisor/. [Accessed: Apr. 20, 2023]
[27] “Adaptive AUTOSAR Hypervisor: EB corbos Hypervisor,” Elektrobit. Available: https://www.elektrobit.com/products/ecu/eb-corbos/hypervisor/. [Accessed: Apr. 20, 2023]
[28] “Automotive SPICE Process Reference Model / Process Assessment Model.” VDA Quality Management Center, Nov. 29, 2023.
[29] “IEC 61508-1:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements.” International Electrotechnical Commission, Geneva, Switzerland, 2010.
[30] M. Schäfer, “Invisible Container Fortress with EB corbos Linux – built on Ubuntu.” Elektrobit, 2023. Available: https://www.elektrobit.com/tech-corner/invisible-container-fortress-with-eb-corbos-linux-built-on-ubuntu/
[31] The invisible container fortress - isolating apps in software instead of distinct ECUs, (Jul. 26, 2023). Available: https://mobex.io/webinar-library/the-invisible-container-fortress-isolating-apps-in-software-instead-of-distinct-ecus/
[32] “Linux OS for Automotive: EB corbos Linux,” Elektrobit. Available: https://www.elektrobit.com/linux/. [Accessed: Apr. 20, 2023]
[33] “Adaptive AUTOSAR Architecture: EB corbos,” Elektrobit. Available: https://www.elektrobit.com/products/ecu/eb-corbos/. [Accessed: Feb. 16, 2024]
[34] “ROS: Home,” Robot Operating System (ROS). Available: https://www.ros.org/. [Accessed: Feb. 16, 2024]
[35] A. Frigerio, B. Vermeulen, and K. Goossens, “Component-Level ASIL Decomposition for Automotive Architectures,” in 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Portland, OR, USA: IEEE, Jun. 2019, pp. 62–69. doi: 10.1109/DSN-W.2019.00021. Available: https://ieeexplore.ieee.org/document/8806012/
[36] C. Lidstrom, C. Bondesson, M. Nyberg, and J. Westman, “Improved Pattern for ISO 26262 ASIL Decomposition with Dependent Requirements,” in 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Sofia, Bulgaria: IEEE, Jul. 2019, pp. 28–35. doi: 10.1109/QRS-C.2019.00019. Available: https://ieeexplore.ieee.org/document/8859482/
[37] “ISO 26262:2018: Road Vehicles – Functional Safety. Part 9: Automotive safety integrity level (ASIL)-oriented and safety-oriented analyses,” International Organization for Standardization, Geneva, ISO 26262:2018-9, Dec. 2018.
[38] “Elektrobit/crinit.” Elektrobit Automotive GmbH, Dec. 06, 2023. Available: https://github.com/Elektrobit/crinit. [Accessed: Feb. 15, 2024]
[39] “Elektrobit/cominit.” Elektrobit Automotive GmbH, Sep. 11, 2023. Available: https://github.com/Elektrobit/cominit. [Accessed: Feb. 15, 2024]

About the author

Prof. Dr. Joachim Schlosser
Senior Manager
Elektrobit

About Elektrobit

Elektrobit is an award-winning and visionary global vendor of embedded and connected software products and services for the automotive industry. A leader in automotive software with over 35 years of serving the industry, Elektrobit's software powers over five billion devices in more than 600 million vehicles and offers flexible, innovative solutions for car infrastructure software, connectivity & security, automated driving and related tools, and user experience. Elektrobit is a wholly-owned, independently-operated subsidiary of Continental.

For more information, visit us at **elektrobit.com**

Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany

Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333

sales@elektrobit.com