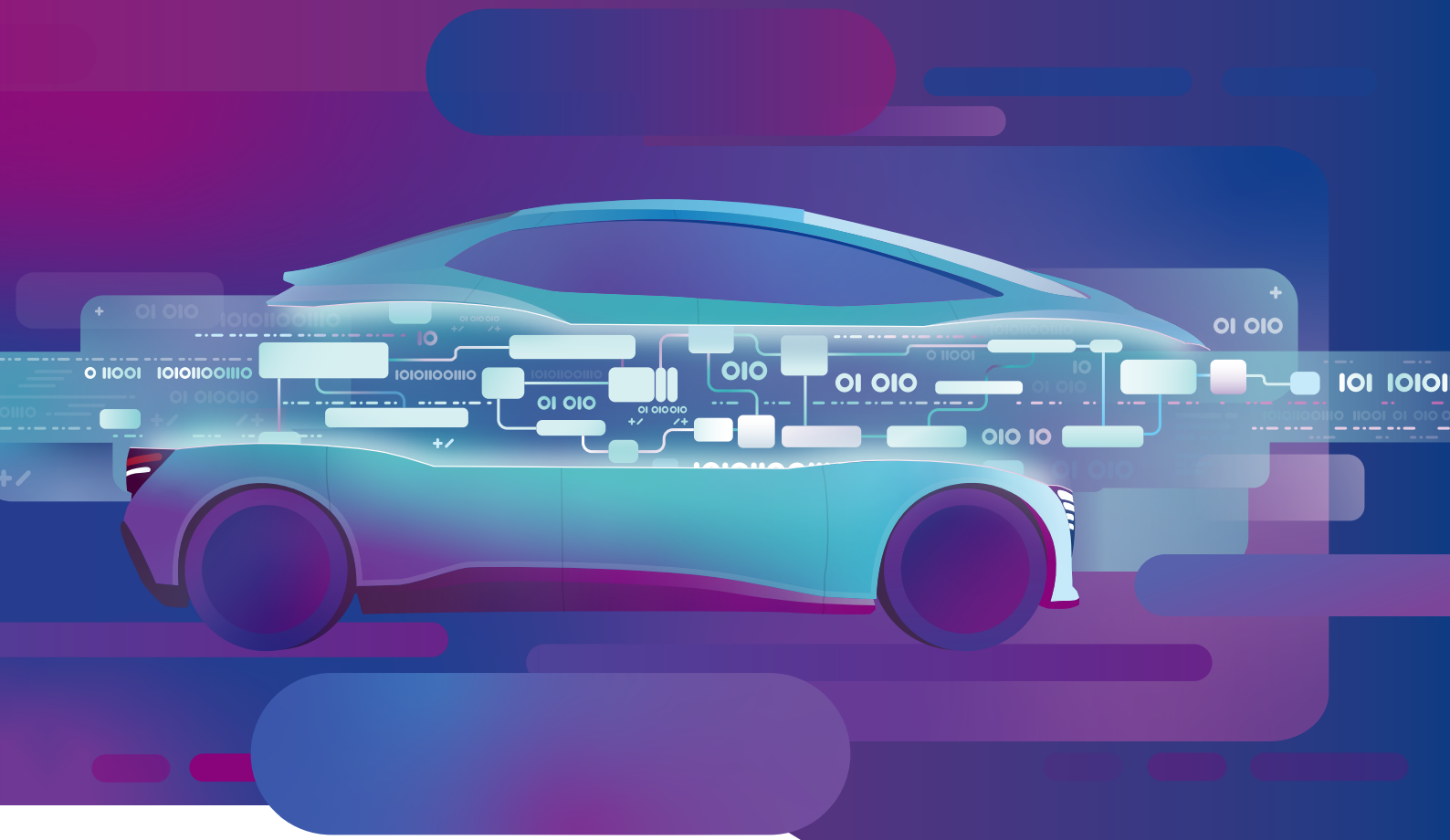

Optimizing AUTOSAR multicore distributions

Practical considerations with real-life automotive examples

Empowering Tomorrow's Automotive Software
www.etas.com

ETAS

Abstract

A few years after desktop computing, the automotive sector is facing the challenge of a software shift towards multicore. This shift requires software to go through significant changes to gain full advantage of new state-of-the-art chip technologies, i.e. microcontrollers (MCUs) and systems-on-chip (SoCs) with homogeneous or heterogeneous multicore systems. More and more developers in the automotive sector are confronted with configuring and optimizing such a system, relying on best practices from experienced partners.

ETAS took up this challenge by developing the world's first multicore AUTOSAR stack for use in vehicle series production back in 2009 and has been developing it further ever since, building on many years of experience in ECU performance optimization in single core systems. As of today, over 4 billion ECUs worldwide rely on ETAS RTA-CAR (RTA-Classic AUTOSAR) to run application software, with an increasing trend towards multicore applications.

This white paper provides a short introduction into how and why multicore has become common. It moves on to three real-life use cases, describing how ETAS supports automotive customers in optimizing their core distribution, followed by a further theoretical example. The use cases are based on typical starting situations, namely the optimization of an existing and the build-up of a new multicore system. The paper ends with an outlook on upcoming functionalities that will further improve multicore applications.

Table of contents

1. Introduction

For decades, more hardware speed translated directly into higher software performance, without the need for extensive adaptation. However, developers hit the "power wall" in the early 2010s – a point where further performance increase was limited due to rising current density and power dissipation. In 2004, INTEL provided a solution to this challenge by introducing the first chip with multiple cores for desktop computers.[1] This new approach was quickly adopted across the industry, although most software remained single-threaded. Profiting from the new multicore designs required a significant software redesign, so that the new technology could not be fully utilized in the same speed as multicore solutions where introduced.

With a short time delay, the power wall also stopped single-core vehicle ECUs from further performance growth. The switch to multicore became indispensable, especially as enabler for more centralized vehicle architectures that can accommodate a growing number of functionalities in fewer ECUs – with the result of reduced overall hardware costs for each vehicle. Consequently, the first automotive multicore microcontrollers entered the market in the 2010s, presenting developers with the challenge of having to convert historically grown ECU software to multicore. With the ETAS Basic Software (BSW) stack, ETAS introduced the very first AUTOSAR stack supporting multicore through process parallelization onto various cores. This solution made it possible to make optimum use of the new technology while retaining key legacy systems.

2. The challenges of parallelization

The distribution of application and base software across cores is the key to taking full advantage of multicore technology. Unfortunately, however, **two cores do not automatically translate into double speed**. The logical reason is that some tasks must be performed after each other, so the beneficial gain of doubling cores can never be 100%. While Amdahl's law can predict the theoretical speedup when using multiple cores, additional (real) factors limit the performance. Some examples are shown in the box on the right.

The challenge consists in gaining a maximum efficiency boost by enabling as many parallel operations as possible, i.e. cutting down on sequential operations wherever possible and enabling optimal memory use to avoid waiting times. Theoretically, adding more cores seems a good solution. However, this also makes concurrency issues more likely, and adds additional issues mainly related to memory access.

Two general types of architectures – homogeneous with multiple identical cores and heterogeneous with multiple different cores – are often mixed in one single device. A typical chip might have several identical "main" cores, plus some specialized cores to accelerate certain functions, such as a built-in hardware security module (HSM) that offers the performant advanced encryption standard (AES). When it comes to memory topology, most current microcontrollers have a hybrid design with a shared memory resource as well as private memories for each core. The different memory areas have different access times. Hence, the memory layout has a big influence on the performance of the system. **Around ten percent of runtime can typically be gained by optimizing the memory access patterns.**

Practical factors limiting speedup (without process parallelization limits):

- **Synchronization and communication overhead**: some part of the program might require data from another core, or it might have to synchronize, resulting in processor wait states.
- **Context switch overhead**: there is always an overhead when a CPU has to do a context switch – from as little as two cycles on some microcontrollers, to millions of cycles in desktop processors or embedded microprocessors, which might need to flush its caches, perform a page table walk, and reload RAM contents from disk.
- **Contention**: if cores share a hardware resource and one core is using it, the other cores must wait until it is available.
- **Interference effects**: even if two cores access two separate peripherals, these two peripherals might be attached to a common bus, which can then become a bottleneck.
- **Memory allocation**: a chip has several physical memory regions, with a wide range of access times. An efficient allocation moves data close to the running code.

3. Special requirements for deeply embedded systems

Deeply embedded systems in vehicles still mainly rely on single core systems, although the development towards multicore is inevitable. Safety, reliability, and real-time behavior are top priorities that have historically grown on single core systems and now need to be shifted or re-modelled for multicore. The main challenge: single core systems using multithreading may already give the impression of multiple parallel actions. Yet simply transferring the code to a multicore system can lead to errors.[2]

Although it is generally desirable to start a multicore project from scratch with thoughtful pre-analysis and planning instead of trying to "convert" an existing single core project, this approach is an exception. Legacy systems play a significant role in the automotive industry and cannot be easily exchanged or replaced. Integrating innovative solutions into existing architectures is therefore often the way to go.

When it comes to deeply embedded systems, AUTOSAR is the industry standard currently available for developers. ETAS experts contributed to the introduction of multicore into AUTOSAR with release 4.0.1 in 2009 and simultaneously developed the ETAS RTA-CAR BSW Stack to make the functionality usable.[3] The release originally consisted of extensions to the operating system (OS), the runtime environment (RTE), and the ECU Manager (EcuM). This opened up the theoretical possibility to migrate deeply embedded functionalities from single to multicore.

Unfortunately, the initial approach of AUTOSAR to become multicore-ready had some performance issues, because the complete basic software ran on one core and used remote procedure calls from the other cores. A concept called "enhanced BSW allocation in partitioned systems" (also known as master/satellite pattern) was introduced with release 4.1.1. to tackle the issues.

Although AUTOSAR comes with documentation of its functionalities and BSW distribution guides, practical implementation is still a very challenging task. ETAS therefore continuously develops its solutions, successively expanding the original stack to a comprehensive stack portfolio (RTA-CAR) to fully support any customer use case.

RTA-CAR at a glance

- Compact & efficient
- Proven
- Safety compliant
- Versatile

4. Practical implementations

The initial situation of developers aiming for multicore is very individual and strongly depends on the existing legacy systems and workflows. Basically, a distinction can be made between support for the development of a new multicore system, the migration of a functionality to a multicore system (from an existing single or multicore), or the optimization of existing distributions. ETAS has decades of experience supporting OEMs and suppliers in the development of ECU software for a variety of automotive use cases. The following paragraphs present three examples of configuration and distribution approaches from real customer projects.

In the first two, the customers already had a multicore distribution in place. However, they were facing problems with their set-up. After a brief theoretical section, we conclude with a third case in which we were able to support the client in a greenfield scenario from the start of the project. All case descriptions are summaries and focus on results.

4.1 Optimizing load distribution

An OEM contacted ETAS to support with a multicore ECU project that had runtime issues with the current distribution on two cores, specifically with the performance of core 0. The project started with a comprehensive analysis of the status quo: a runtime measurement showed that core 0 was almost fully loaded, while core 1 had plenty of free capacity left, as depicted in figure 1.

This overload was reduced to a suitable level by moving the complete Com stack, which was responsible for 15% runtime, from core 0 to core 1. **This new distribution introduced a 4 percent increase in the overall load of the system**. This was partially due to the introduction of additional cross core calls, for example between the diagnostic stack on core 0 and the Com stack on core 1. Previously, these were situated on the same core. Moving only the Com stack meant that calls, which used to be within one core, now had to cross the core boundary. Another reason for the increase of total runtime: the measurement was done before adapting the memory layout to the new distribution. The Com stack data was still in the memory of core 0, leading to longer access times from core 1 and an increased runtime.

Since only the runtime on core 0 was critical, the increase of total runtime was not problematic for this project. Nevertheless, optimization, such as moving the diagnostic stack and optimizing the memory allocation, were performed, which reduced the runtime increase in the further course of the project.

This example shows one of the many trade-offs involved in configuring a multicore system. **More distribution is not always better**. Instead, it is important to start with a clear understanding of what should be optimized (load on a single core, total runtime, interrupt latency, etc.)

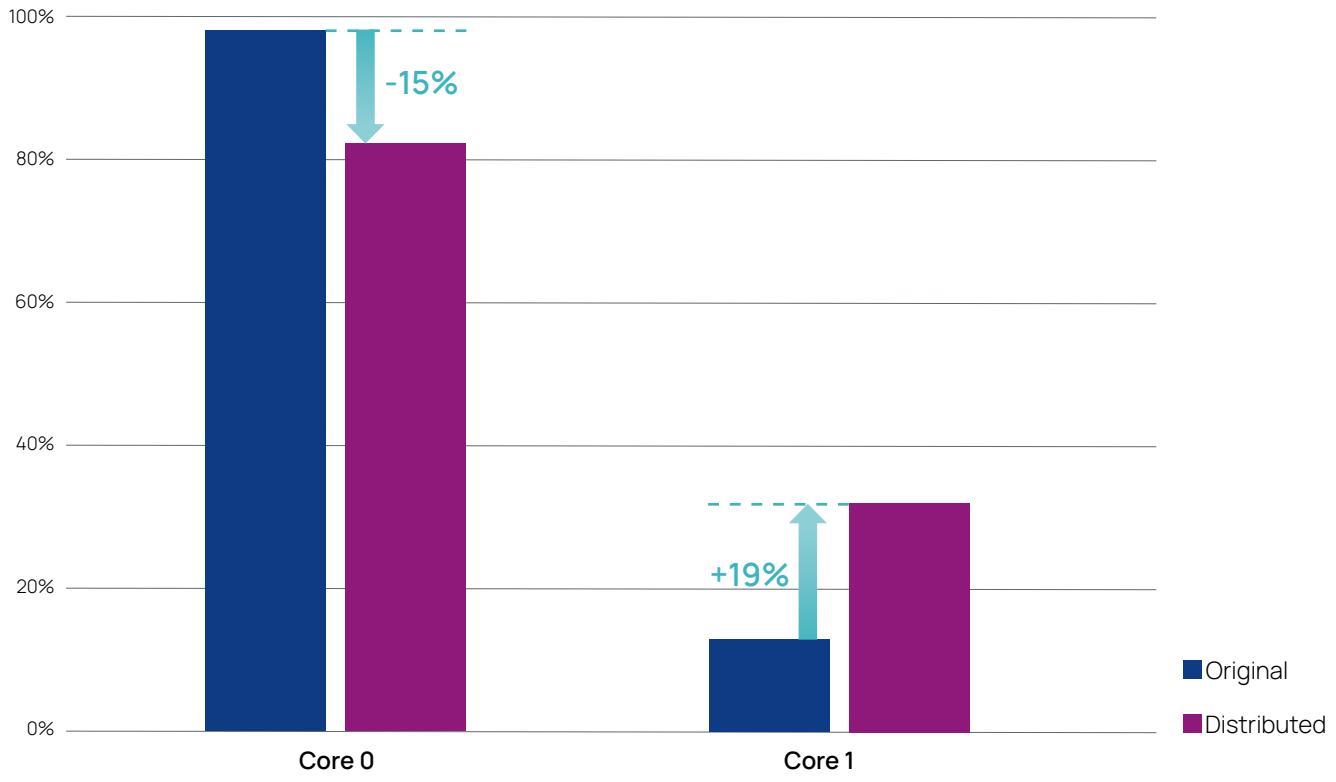

Figure 1: Moving load from core 0 to core 1 resulted in a better distribution of the runtime, at the cost of an increase in the total load.

4.2 Reducing runtime spikes

In our second use case, a Tier 1 supplier was facing a different sort of performance problem with a multicore brake ECU. While the average load was fine and each core still had runtime reserves, the system did not perform as expected during some runtime spikes. Runtime measurements showed that the maximum runtime of the Com_MainFunctionRx within the Com stack was critical: it was periodically called every five milliseconds to process newly received Pdus. However, some ASW functions only ran every 40 milliseconds. In the worst case, the Com_MainFunction was processing the data eight times more often than required.

The Com_MainFunction was therefore split into several main functions (see figure 2) with different periods, while Pdus were assigned to a function with the same period as the ASW function ultimately reading the data. As an additional optimization, the Rx indication, which is normally started by the Rx interrupt, was split and most of the processing was moved out of the interrupt context. Together, these measures solved the runtime issues of the project.

name	max[µs]	Pdus
Com_MainFunctionRx	360.65	120

name	max[µs]	Pdus
Com_MainFunctionRx_5ms	24.043333	8
Com_MainFunctionRx_10ms	36.065	24
Com_MainFunctionRx_20ms	33.810938	45
Com_MainFunctionRx_40ms	16.154115	43

Figure 2: Splitting the Com_MainFunction based on the message timing.

Diving deeper into the logic of this new distribution requires a closer look at the Com stack. It consists of several layers. The lower layers are concerned with tasks like packing and unpacking of frames, adding and removing protocol-specific headers, fragmenting and combining frames, etc. They are optimized for speed and perform little computation. Therefore, the layers below the PDU Router (PduR) require less runtime. Most of the runtime is concentrated in the higher layers (PduR level and above). Hence, ETAS recommend processing the trigger in a lower layer and letting the higher layers poll (figure 3). The distribution of the signals to the cores is executed on the level of the PduR, as explained in the next section and depicted in figure 5.

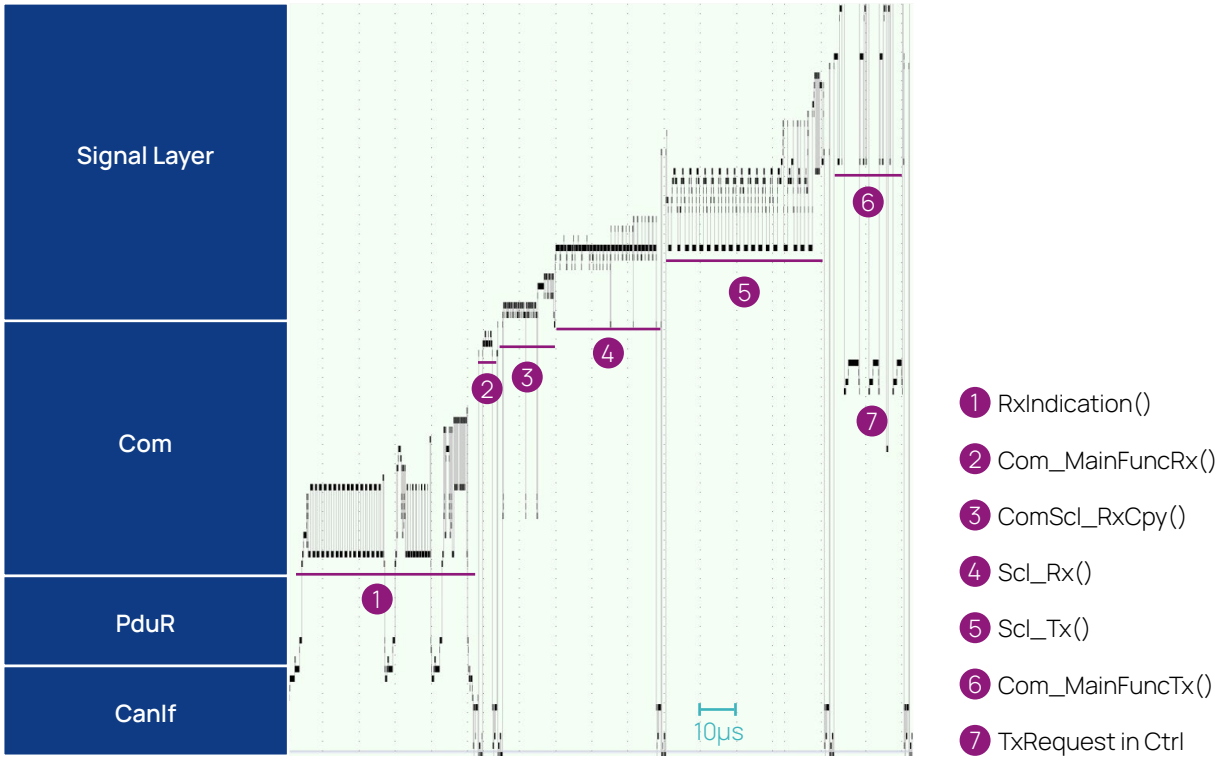

Figure 3: The main load is located on the higher layers. ETAS recommends to run lower layers (bus stack) on one core and distribute on PduR level to match ASW distribution.

4.3 Master/satellite and multi-master approach to BSW module distribution

The previous examples showed different optimization aspects, which occurred in real world multicore projects. Let's now have a look at what AUTOSAR has to say about multicore BSW and how the ETAS RTA-CAR BSW stack implements it.

The purpose of an ECU is to accomplish a task that is implemented in the ASW, and the BSW has to enable this. In a multicore system, the ASW runs on several cores. Calling functions on a different core invokes a switching cost (as do calls to a different partition on the same core, but this cost is smaller). Some BSW modules are frequently accessed from ASW. Here the switching cost can lead to runtime or latency problems. Typically, these are BSW modules that have Service SwComponents, like Dem, Fim, NvM, or WdgM. For modules like these, which need to be accessed from different partitions, AUTOSAR defines the so-called master/satellite approach. A stack is split into a master, which controls the lower layers, and satellites, which provide access for the ASW on other cores or even in other partitions. Figure 4 shows an example for NvM.

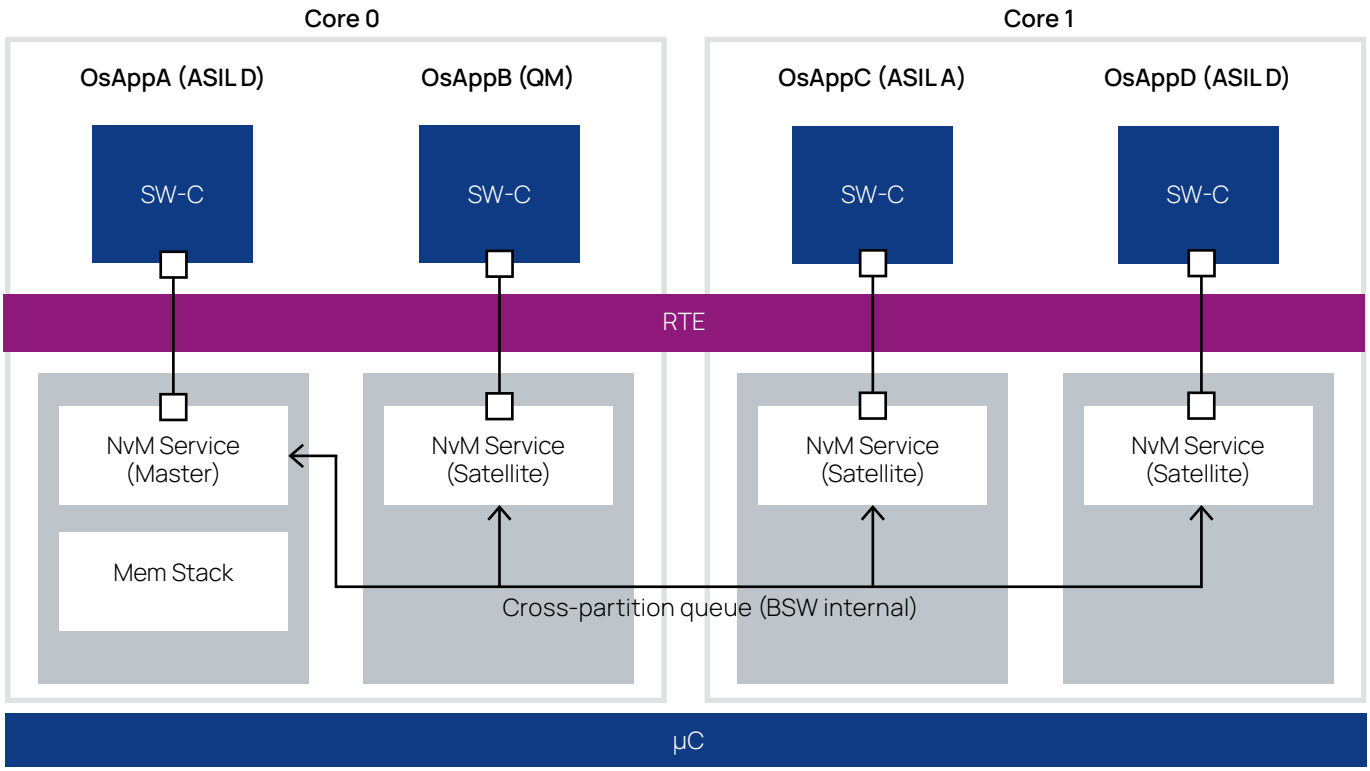

Figure 4: Modules that need to be accessed in different BSW partitions can be implemented by the master/satellite pattern.

The ETAS BSW services can be distributed between cores with the goal of local runtime optimization. There is, however, often a trade-off with latency. In case of asynchronous calls, for example when reading NvM data, the master/satellite approach offers lower call latency. In most cases, however, low latency requires synchronous calls. If the BSW is distributed across cores, the calling core is blocked until the call is completed on the called core. So, in terms of total performance as well as optimum global latency, it is advantageous to use asynchronous calls as much as possible.

In the Com stack, the highest layer is the Com module. The layer below is PduR (the PDU Router). Since the PduR already has the purpose to distribute data, this is the natural module to use for master/satellite. This is why, in the case of the Com stack, the split is not on the highest layer (Com), but one layer below (PduR), as depicted in figure 5.

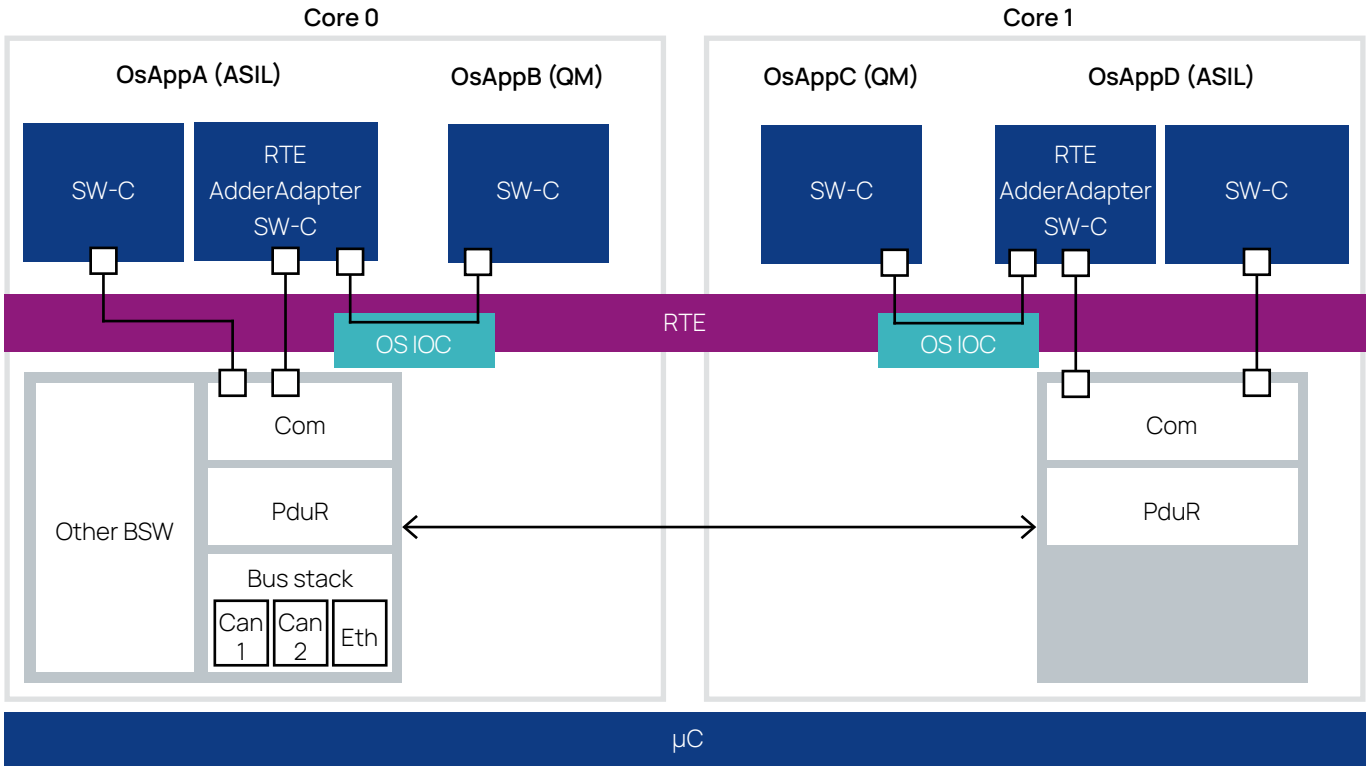

Figure 5: The distribution of the signals to the two cores takes place on the level of the PDU Router (PduR).

Multiple instantiations of the Com stack means that the Rte must call a Com instance inside the correct partition context. However, in some cases, it cannot know what the correct context would be. As an example, let's assume there is a Com stack in OsAppA and OsAppD and a SW-C in OsAppD sends a signal. Now there are two options: the Rte could call the Com stack in OsAppD or use IOC to transfer the data to OsAppA and call the Com stack in OsAppA. For this kind of signal paths, the Rte cannot make the decision. Therefore, ETAS uses a generated software component called RTE AdderAdapter. It splits the signal's path into two parts: one that uses IOC and one that calls the ComStack inside the same partition.

In some cases, projects have the requirement to map complete buses to other cores. For example, if part of the ASW is responsible for controlling a device attached to an exclusive CAN, it makes sense to move the complete CAN processing to the same core. Of course, this is also supported by the ETAS stack. Since the lower layers are now also distributed, we call this approach multi-master. In figure 6, CAN 2 is handled by core 1, while Ethernet is on core 0. In such a scenario, the PduR is also split, and its parts need to be connected. If a CAN frame received on core 1 should be forwarded to the Ethernet on core 0, then the PduR must communicate the information between these two cores.

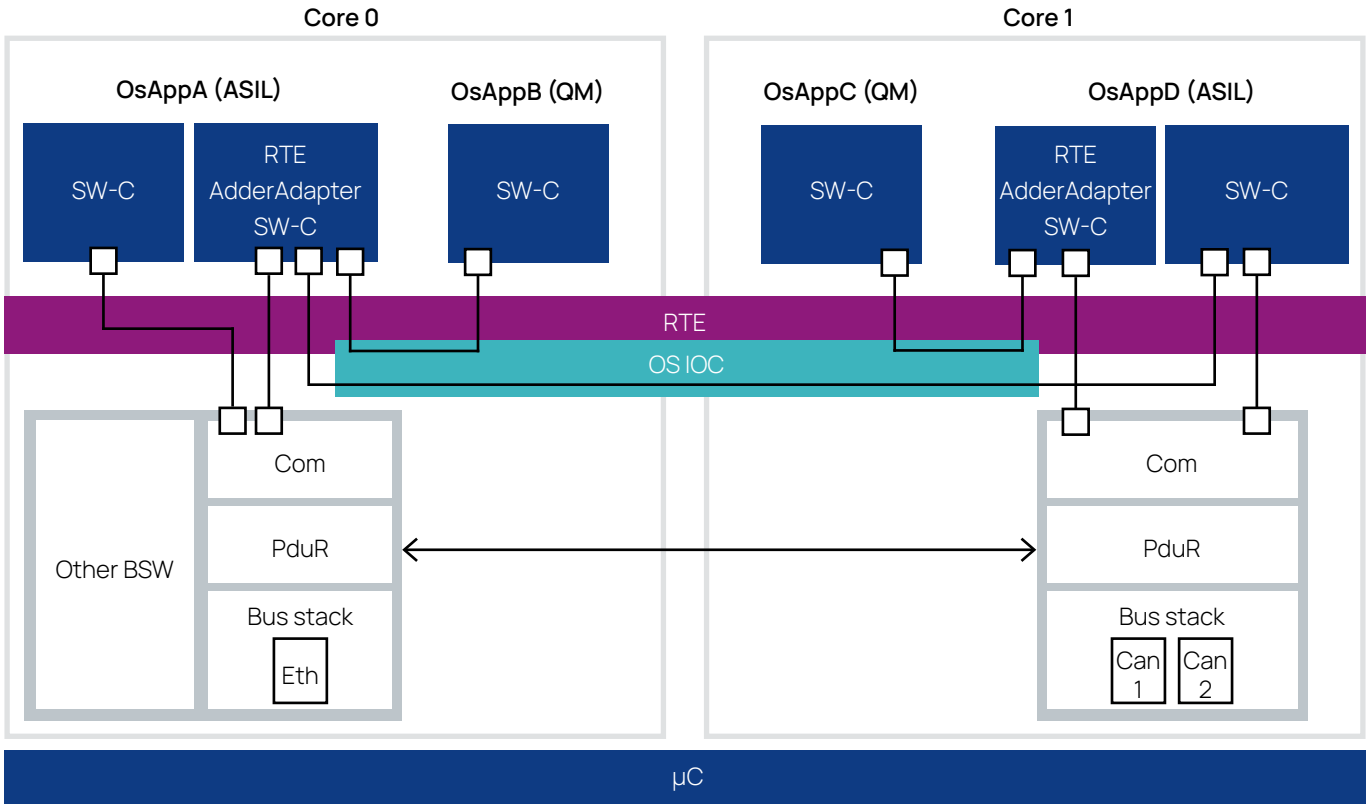

Figure 6: Depending on the project, it might be necessary to move complete buses to other cores.

4.4 Setting up an innovative multicore project from scratch

Until now, the focus was on optimizing already defined core distributions. In the third real-world example, a customer approached ETAS regarding the design of a new generation of gateway ECUs: a multicore gateway with bus mirroring and a highly distributed Com stack. In this case, the distribution between cores could be tailored in a greenfield scenario, finding the best solution for this very complex setup on five cores. To reach the customer's goals of an optimized performance from a runtime perspective, several novel requirements, for example bus mirroring, had to be implemented within a very challenging timeline.

The result, as it went into production, is shown in figure 7. The PduR level passes information between the various cores (with the mapped buses), and the task-oriented Com stack implementation allows a very fine-grained distribution of the processing load.

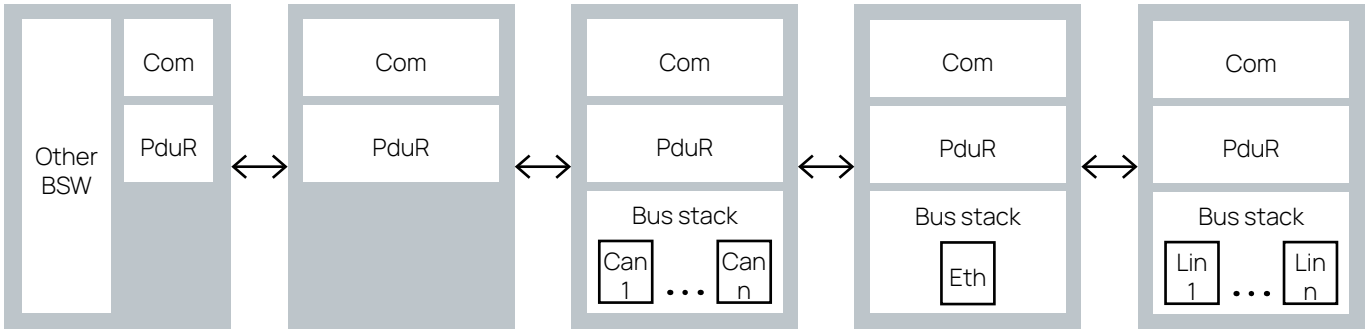

Figure 7: Final distribution across the cores for maximum performance, with optimal connection to all layers based on PduR.

To ensure data integrity between cores in this kind of setup, the typical technique is an implementation using locks. As locks might compromise performance and lead to higher waiting times, ETAS took a different approach: the implementation of the PduR interface via the proprietary ETAS RTA-CAR BSW module XCoreCDD (Cross-Core Complex Device Driver) no longer requires locks and can further optimize the process (figure 8). Even cross-bus-gateway configurations without locks are possible. XCoreCDD also supports Freedom from Interference (FFI) for safety-critical workloads.

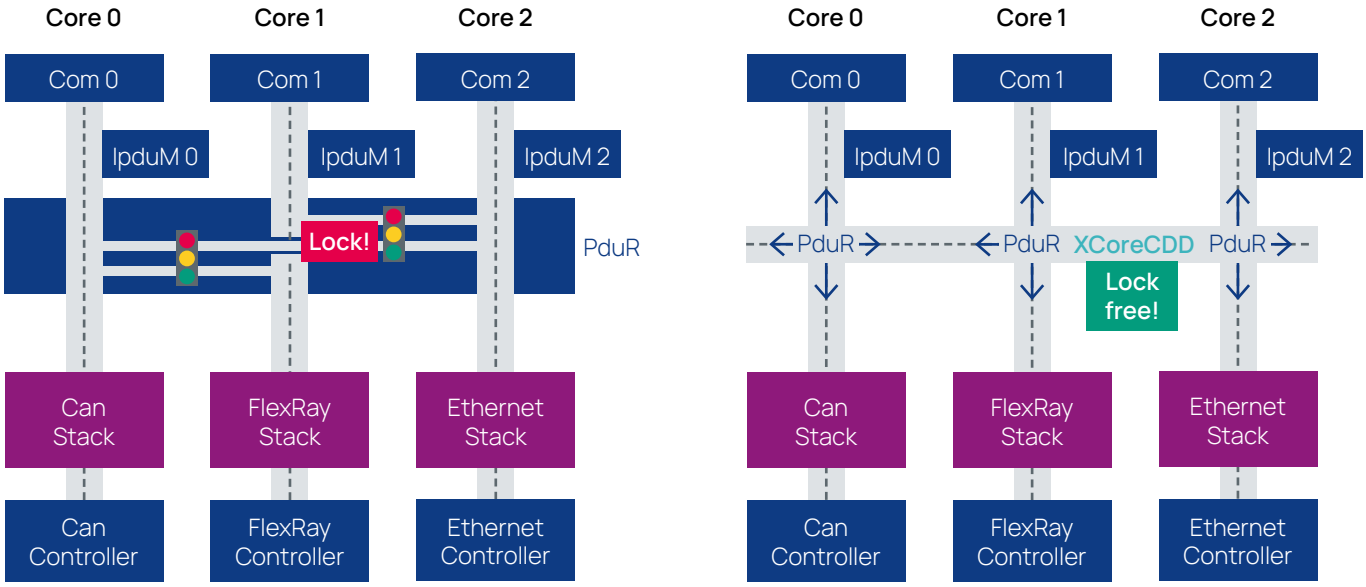

Figure 8: With the XCoreCDD, locks between cores are no longer needed, resulting in shorter waiting times and higher performance.

5. Outlook

An efficient multicore distribution is essential for next-gen vehicle architectures, impacting more and more ECUs. It's a key enabler for present projects as well as for future automotive advancements like autonomous driving and connected mobility. Building on the strong foundations demonstrated in this white paper, ETAS is continuously extending the multicore capabilities of RTA-CAR. This includes tools to understand and optimize multicore performance. Upcoming RTA-CAR releases will introduce the new exclusive area configuration editor that uses static analysis to help users to optimize the performance of their systems by configuring the right type of lock (the so-called exclusive area implementation). We are also extending the support for master/satellite to additional modules and continue to optimize our BSW stack for high performance and small size. This efficient use of resources makes sure that OEMs and Tiers can get the most out of current and future vehicle architectures.

6. Our offering for AUTOSAR multicore projects: RTA-CAR

RTA-CAR is a cutting-edge, low-footprint software solution designed for series production in automotive applications. Used by hundreds of companies worldwide, RTA-CAR powers billions of automotive ECUs in millions of vehicles. It is suitable for single and multicore projects.

Key benefits

- **Compact & efficient**: optimized for resource-constrained, low-latency real-time environments
- **Safety & security**: complies with industry standards like ISO26262 and ISO/SAE21434, along with MISRA-C & CERT-C guidelines
- **Proven success**: deployed on over 4 billion ECUs globally
- **Versatile compatibility**: suitable for all ECU types, from smart sensors to vehicle computers, and adaptable for single and multi-OEM projects
- **Wide device support**: compatible with various target devices and seamless integration with third-party MCALs and security solutions

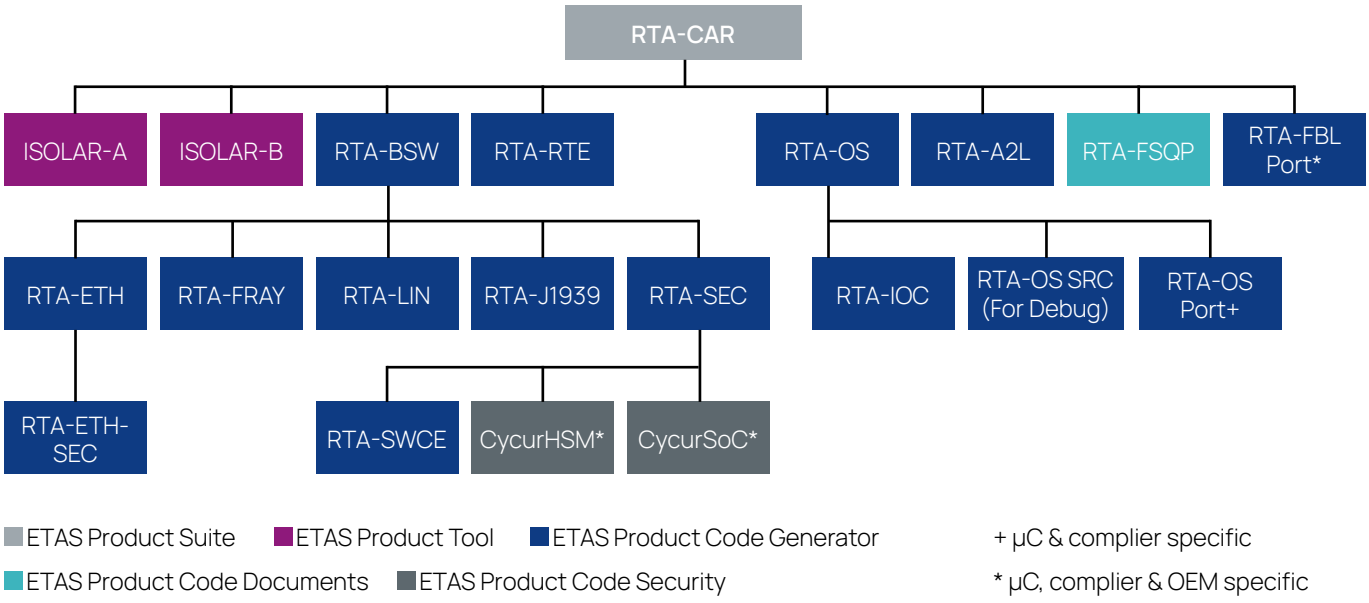

Figure 9: RTA-CAR product offering

ETAS offering beyond AUTOSAR for core distribution projects:

- **Lock-free XCoreCDD**: implementation of a lock-free pipe that connects the PduR instances. The advantages: no system block, reduced waiting times for cores, improved performance, and uninterrupted parallel processing.
- **Com adapter**: decouples ASW distribution from Com stack distribution.
- **Multi-master for WdgM**: in addition to regular master/satellite, WdgM supports multi-master, where each OsApplication runs its own master; especially useful if the software running on the different cores is independent from each other.

About ETAS

Founded in 1994, ETAS GmbH is a wholly owned subsidiary of Robert Bosch GmbH with a local presence in all major automotive markets in Europe, North and South America, and Asia.

ETAS offers comprehensive solutions for the realization of software-defined vehicles in the areas of software development solutions, vehicle operating system, vehicle cloud services, data acquisition and processing solutions, integrated customer solutions and cybersecurity.

As industry pioneers in cybersecurity, we assist our customers in managing cybersecurity-related complexities, reducing cyber risks, and maximizing their business potentials with a proven on- and offboard portfolio of software products and professional security services.

ETAS automotive security solutions are safeguarding millions of vehicle systems around the world – and are setting standards for the cybersecurity of software-defined vehicles.

References

1) https://www.electronicsweekly.com/news/products/micros/microprocessors-2004-in-review-2005-01/

2) Fridtjof Siebert. Multicore Systems – Challenges for the Real-Time Software Developer. ERTS2 2010, Embedded Real Time Software & Systems, May 2010, Toulouse, France. hal-02267727

3) https://www.autosar.org/fileadmin/standards/R24-11/CP/AUTOSAR_CP_EXP_BSWDistributionGuide.pdf

Contact information

Jakob Kristoferitsch
Senior Expert AUTOSAR Methodology

Vadim Dillmann
Release Train Architect COM

etas.com/YourAmbitionIsOurMission
etas.com/AUTOSARClassicPlatformSolution

Last updated: 04/2025

ETAS GmbH
Borsigstraße 24, 70469 Stuttgart, Germany
T +49 711 3423-0, info@etas.com

Are you interested in ETAS products or solutions? Please visit **www.etas.com**

Or follow us on social media: