TTTech

Deterministic Architecture and Middleware for Domain Control Units and Simplified Integration Process Applied to ADAS

Dr. Georg Niedrist

Learn More
About TTTech's
ADAS Solutions

www.tttech.com/ADAS

Imprint

TTTech Automotive GmbH
Schoenbrunner Strasse 7
1040 Vienna, Austria

office@tttech-automotive.com
Phone: +43 1 585 65 38-5000

FN 165 664z, Commercial Court Vienna

Printed by Paco Medienwerkstatt, Vienna

www.tttech-automotive.com

Taking the Right Turn with Safe and Modular Solutions

High Performance Domain ECUs for Advanced Driver Assistance Systems

The Austrian high-tech company TTTech Computertechnik AG is technology leader in robust networked safety controls. TTTech products are applied in various safety-relevant areas in the space and aerospace domains, energy production, railway systems and industrial process automation.

The subsidiary TTTech Automotive GmbH provides solutions for the challenges of future vehicle generations. Currently, the company is developing an innovative domain ECU for serial production in the automotive industry, intended for Advanced Driver Assistance Systems (ADAS). More than 30 different ADAS-functions can be integrated and monitored on this platform, by means of different high performance multicore processors and a corresponding middleware. Even applications with different safety levels – up to ASIL D after ISO 26262 – can be integrated easily and cost-efficiently and run simultaneously on the in-house developed software TTIntegration. Deterministic Ethernet serves as the on-board network, providing bandwidth of up to 1 Gbps. This future-proof ECU solution is being expanded to a whole model family.

Additionally, TTTech Automotive offers modular hardware and software solutions, based on safety-certified modules as well as effective system solutions, e.g. for electric drives or dynamic stability control systems. Reliable logging tools for bus systems like FlexRay, MOST and CAN complete the offering.

Introduction

Modern cars offer an ever increasing number of electronic functions in all vehicle domains, such as Advanced Driver Assistance Systems (ADAS), infotainment systems, vehicle dynamics systems and hybrid and electric drivetrains. In the past, each new customer function required yet another electronic control unit (ECU). Due to costs, packaging, wiring and thermal constraints, this is not a suitable solution anymore. Today and even more so in the future, we can observe a trend toward the modularization of automotive electronic systems and the implementation of new customer functions controlled by software only, thus effectively reducing the number of ECUs or at least not increasing it further.

In parallel to those technically mandated changes, we can also observe a change of supply chain and cooperation models. Instead of sourcing a complete closed-box system (including sensors, ECUs and possibly actuators from a single tier-1 supplier), original equipment manufacturers (OEMs) increasingly turn to a more open “cherry picking” model. This means that OEMs select the best-in-class elements (sensors and sensor processing software, ECU hardware and platform SW, application SW modules, actuators) separately from a number of tier-1 and SW suppliers and integrate the complete system on their own or in a network of partner companies. This approach by the OEMs is driven by their need for differentiation, which also results in increasing in-house development of customer functions by the OEMs.

All these trends lead to the emergence of central domain ECUs on the basis of platform architectures. For example, in the ADAS domain, the requirement for central sensor fusion quite naturally leads to a central fusion and application controller architecture. Generic platforms are needed to foster function SW reuse across different car models, which in turn is necessary to cope with the immense costs and efforts for function validation especially in the ADAS domain.

Due to a changed cooperation model and the dramatically increased technical complexity, not only the new role of the SW integrator emerges in this scenario, but also the requirements for the domain ECU architecture, the SW platform and the integration process differ from traditional ECUs very much. In the following sections, we discuss the requirements for such domain ECU architectures, with a particular focus on ADAS systems. We present a novel solution that is currently employed in full series development with a variety of partners. ■

Requirements for ADAS Domain ECUs

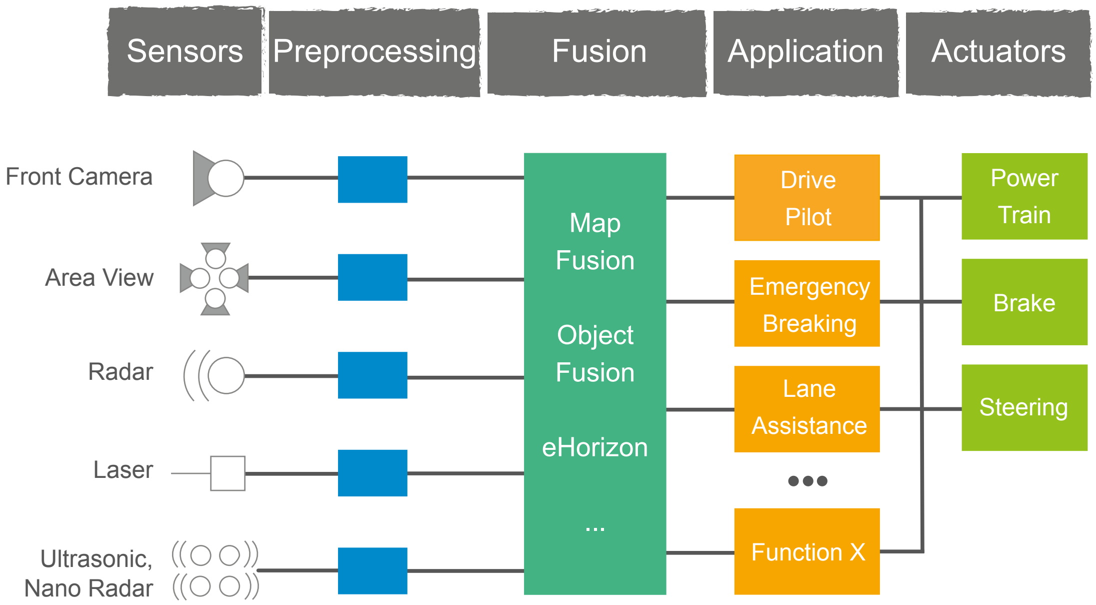


Figure 1: Abstract View of a High-End ADAS System

Figure 1 gives an abstract view of a high-end ADAS system, which, e.g., is needed in autonomous driving scenarios.

A set of diverse sensory inputs, such as radar, cameras, LIDAR and ultrasonic sensors, are processed to identify elementary pieces of information about the environment of a car (lanes, traffic signs, people, other cars, etc.). These pieces of information are fed into a central fusion layer to compute a complete and consistent representation of the surrounding of the car and the trajectories of all objects. The application layer builds upon that fused information, devises the driving strategy and implements the navigation and control algorithms to drive the vehicle, by controlling steering, braking and the drivetrain. Besides this core function of autonomous driving control, a variety of additional comfort and utility functions will usually be hosted by a central ADAS controller, ranging from customer functions (such as Automated Emergency Braking (AEB), lane assistance and surround view) to vehicle utility functions, such as event logging and black-box data recording functionalities.

The SW functions hosted by a central controller will depend on the overall vehicle architecture of the OEMs. The architecture might include raw data sensor processing or, alternatively, rely on object data received from "smart sensors" that are attached to the central controller. In any case, at least the sensor fusion layer plus all the application and utility functions must be integrated. As this task is complex and as most architecture considerations are also relevant for this "simpler" case, we will not cover the specific system architecture in further detail.

The requirements for the HW and SW architecture of an ADAS domain controller come from many sources (performance, safety, application SW development, and integration process), mastering the technical complexity, as well as modularity and scalability needs in order to support different vehicle options. Similar requirements have been present in the aerospace industry for a long time, leading to the well-established architecture of (Distributed) Integrated Modular Avionics (DIMA). The approach presented in this paper is, thus, derived from DIMA and adapted to the specific requirements set by the automotive domain in terms of costs, functionality and scalability.

Performance: The highest possible processing power is a natural requirement in the ADAS domain, given that there is an ▶

immense computational demand for sensor processing, sensor fusion and vehicle control. Depending on the overall system architecture, there might be a need for image processing devices (such as graphics processing units, GPU) alongside general-purpose processing devices with a high performance. The latter devices come as multicore System-on-Chip (SoC) designs based on ARM A53 or A57 cores, for example. Devices used in the consumer and infotainment worlds are on the forefront when it comes to a high-integration and performance/power ratio and, thus, must be included.

Safety: ADAS domain controllers will need to conform to ISO 26262 ASIL C/D in order to support autonomous driving functions. This overall safety requirement can currently not be met by high-end SoCs, and is, therefore, usually broken down by means of ASIL decomposition[1] and clever SW partitioning. For this reason, an ADAS domain controller will usually incorporate at least one ASIL D-capable automotive microcontroller in parallel to high-end SoC devices that have lesser safety capabilities. Safety capabilities on the HW side must be supplemented by platform-SW mechanisms (memory partitioning, timing supervision, protection of communication between applications, HW diagnostics and supervision) to fully support the applications.

Interfaces: ADAS domain controllers usually require a set of traditional automotive network interfaces (CAN, FlexRay™, LIN®) in addition to Ethernet and, possibly, raw-data video interfaces (LVDS, CSI) in order to connect sensors. Along with CAN or FlexRay™, usually utility functions – such as diagnostics, calibration and security features – need to be supported, which can be properly addressed by using an AUTOSAR® Basic SW. High-end SoCs might not offer those interfaces and services and need to be supplemented by a traditional automotive microcontroller.

ECU Modularity and Scalability: To support vehicle options, a domain controller will need to come in several variants, from a top-line system to basic functions. SW reuse across all the variants is mandatory in order not to end up in developing separate ECU SW packages for each variant. It is essential to abstract the underlying HW, operating systems and communication mechanisms. The application SW components (SWCs) must be completely independent of the underlying platform implementation, which must not contain any ad hoc application-specific elements. Under this condition, we can add or omit processing elements (as required

1
ASIL decomposition means that a higher automotive safety integrity level (ASIL) can be achieved by appropriately combining elements that have lower levels. For example: Two elements which each fulfill ASIL B requirements can result in an ASIL C-capable or ASIL D-capable system.

by performance and safety demands for the different ECU variants) and still freely reuse and move SWCs between processors, thus drastically decreasing SW development and validation efforts.

Development Process: In a rapid prototyping scenario, application SW in the ADAS domain is initially often developed with modelling tools, such as ADTF or MATLAB®/Simulink. It is highly desirable to support a seamless transition from such PC-based development to an industrialized implementation on the target ECU by ensuring continuous usability of test cases across all development phases and ideally identical code to run on a PC or on the target. Mixed configurations – PC-based SWCs integrated with target-based SWCs – must be possible in a Software in the Loop (SIL) setup.

SW Integration Process: HW and SW platform architectures must allow the controlled integration of a large number of SWCs that can come from many different teams. Controlled means that the SW integrator (be it an OEM or a third party) must not be overloaded with debugging, mediation and conflict resolution tasks among SWC suppliers. We want to avoid a scenario where single SWCs work just fine, but the completely integrated system does not because of timing interaction and resource conflicts that could lead to an endless procedure of iterations and incremental fixes. So we need composability: Individually tested SWCs must immediately work, once integrated together. This also means that predictability is a key requirement with regard to resource consumption, runtime, data flow latencies and sequences of SWCs. Finally, a black-box integration approach (no source code, anonymized interfaces) is desirable to support IP protection among potentially competing SWC suppliers.

Mastering Complexity: A high-end central ADAS controller might have approached the complexity of complete vehicle electronics just a few years ago. A clean SW architecture and the abstraction of HW and operating systems must provide the required modularity, portability and simplicity of interfaces for the SWCs. So, what is not just a matter of comfort, but a question of feasibility is observability of all ECU-internal communication through appropriate debugging utilities (of course without influencing system timing, as are available at vehicle level through data logging and tracing), and the above-mentioned requirements for the SW integration properties without mutual side effects.

All of these requirements apply not only to an "open" cooperation scenario where an OEM specifies, sources and integrates a complete system that comes from contributions of a variety of suppliers, but also to the more "closed" environment of a single tier-1 system supplier integrating a complex domain ECU together with a multitude of in-house teams and, possibly, OEM contributions. ■

HW and SW Architecture Overview

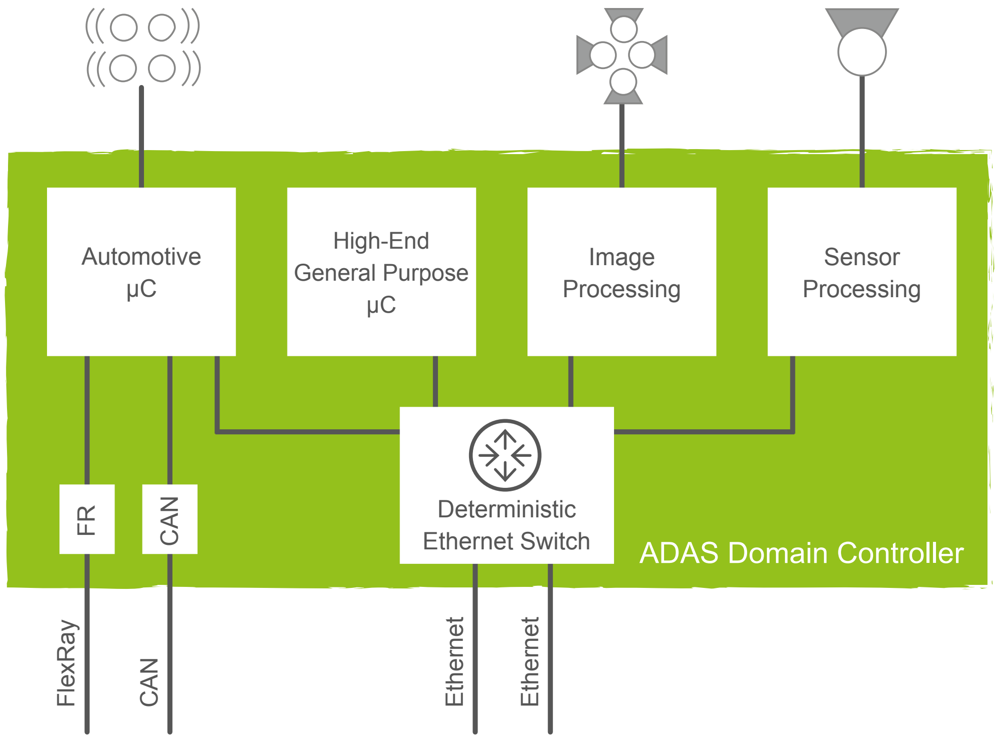


Figure 2: Abstract Block Diagram of ECU HW Architecture

As shown in Figure 2, the architecture consists of a diverse set of processing elements: an automotive microcontroller (such as Infineon Aurix or Renesas RH850), a general-purpose processing engine that is based on a field programmable gate array (FPGA, e.g., Altera Cyclone V), an image processing device (such as Nvidia or Renesas R-Car H3), and further specialized sensory processing devices. All the devices are interconnected by a Deterministic Ethernet (DE) switch, and there are no direct point-to-point connections that hinder the debugging and tracing of intra-ECU communication, which would clearly be a showstopper for the integration and debugging of the highly complex interaction between a multitude of SWCs[2]. CAN and FlexRay™ provide external interfaces to the traditional car network, with Ethernet connecting high-end car networks and serving as a debugging interface.

The Deterministic Ethernet (DE) switch can be integrated in an FPGA or included as a standard off-the-shelf device (e.g., the NXP SJA1105T chip). Besides the normal best-effort Ethernet switching capability, the DE switch provides a time-triggered, deterministic mode of operation where all communication and switching takes place according to a predefined schedule in a completely predictable way (similar to the FlexRay™ mode of communication). All processing elements are synchronized to the switch and schedule their internal task processing accordingly. Thus, the DE switch provides the central heartbeat of the ECU and ensures deterministic, collision-free and jitter-free communication among the SWCs on the processing hosts. The DE switch, thus, is the core element to ensure the above-mentioned composability and deterministic integration property.

Note: The processing elements just use their standard, integrated Ethernet MAC devices and need no special HW precautions. However, there are SW enhancements at driver and operating system (OS) level to support synchronized communication.

Safety is implemented by ASIL decomposition across the devices, with the automotive microcontroller providing the ASIL D capability, the FPGA fulfilling up to ASIL B requirements, and the GPU being used for QM-rated algorithms. All the devices provide memory protection and ensure the associated freedom from interference to support mixed-criticality operation. End-to-end communication protection is applied to all safety-relevant inter-SWC (intra-ECU) communication. ▶

2
A possible exception might be direct, specialized point-to-point connections between devices to share raw sensory data. However, this has no impact on the communication and integration of SWCs and is, therefore, not covered here in further detail.

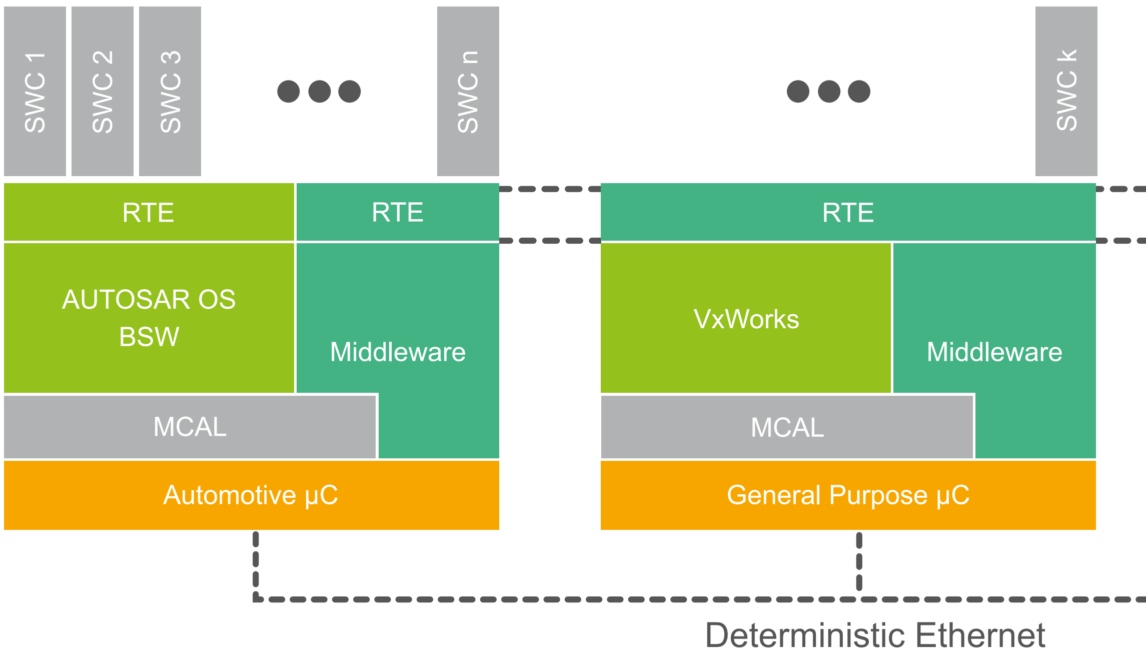
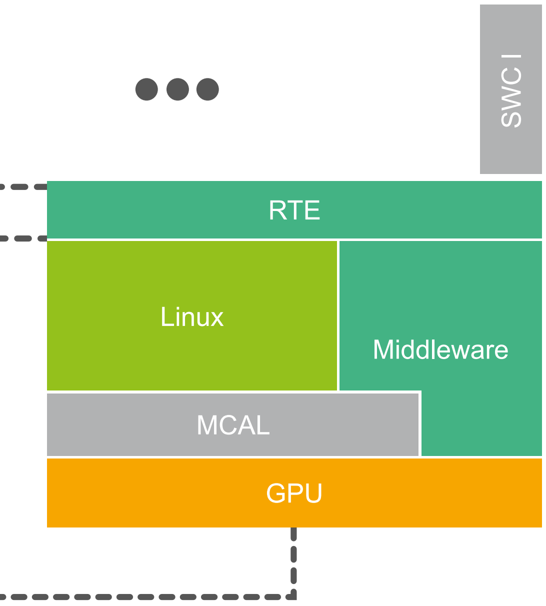


Figure 3: High-Level SW Architecture Overview

Operating systems (OS) are selected according to a best-fit approach:

The automotive microcontroller uses AUTOSAR® Basic SW and OS. This contains the proven SW stacks for vehicle communication interfaces and basic SW services, as well as commonly used automotive functions (diagnostics, calibration). This choice ensures compliance to OEM vehicle networks and operation requirements, and also allows the easy integration of OEM-specific utility and legacy functions, such as special in-house security protocols.

VxWorks by Wind River is used as a safety-enabled OS for the general-purpose SoC (FPGA), where safety-relevant applications up to ASIL B need to be run, but their performance demands exceed the automotive microcontroller's specifications. Besides the safety capability, VxWorks offers a wealth of proven functionalities and services, and POSIX® compliance.



Linux is the natural choice for non-safety-related processing elements on high-end SoCs (GPUs), in particular, as it offers easy, often out-of-the-box integration of graphics processing libraries, such as OpenCL, OpenCV, CUDA and similar services.

This selection is quite natural, taking into account the different processing elements, but leading to a very heterogeneous OS solution. Without any additional measures, the applications will find a completely different operating environment and API on each host, which strongly ties each application SW component to a predefined host. Additionally, basic mechanisms, such as communication and data management, are not compatible with different operating systems. Without further precautions, the applications would need to implement proprietary ad hoc mechanisms and adaption layers, which would be very likely to mash up platform functionalities with application code.

Our solution to providing a common execution environment and API on all the hosts and to ensuring HW and OS abstraction is to place a generic middleware layer on top of each OS and include an AUTOSAR® runtime environment (RTE) type of communication. This RTE layer plus a broad set of utility functions form the generic middleware on all the hosts. This offers one common, host-independent API to the SWCs and achieves full HW and OS abstraction, providing complete location transparency, which means that all services are available on all hosts transparently. This also ensures full portability of SWCs. They may be flexibly located to processing elements according to their requirements for performance, memory and safety. If these requirements should change, it is also possible to move SWCs freely between processing cores and hosts or optimize the utilization of processing power and memory of the ECU. For the SW integrator, this simply amounts to a configuration step at compile time, without any code change of the SWC!

To achieve the deterministic integration property with its benefits (predictability, no integration side effects), the time-triggered mode of operation is also ▶

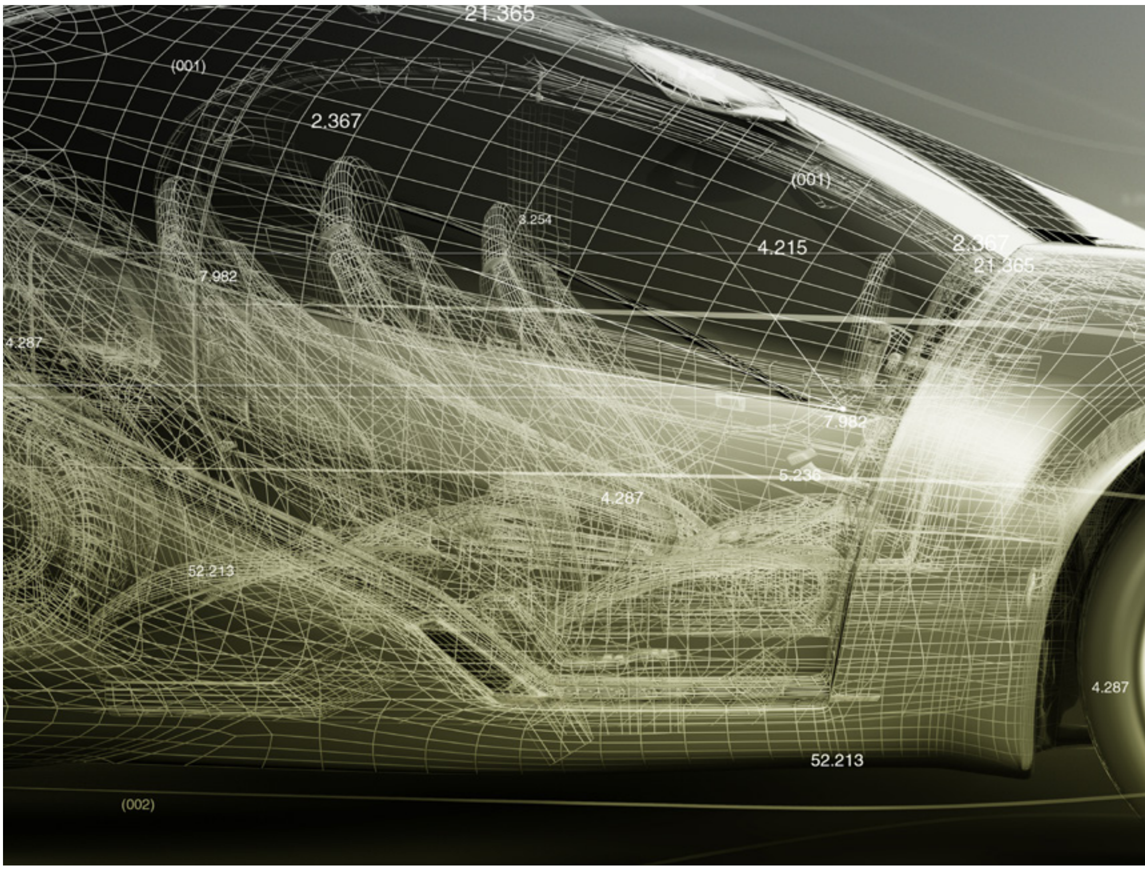
implemented in the middleware, by means of OS-specific enhancements for synchronization and task scheduling/dispatching.

Finally, the middleware layer is also extended to the PC environment, allowing seamless SIL interaction (co-simulation) among applications on the PC and the target ECU and bridging the gap between rapid-prototyping and industrialized SW components, while keeping all timing relations fully intact, and without any necessary code change on the part of the SWCs.

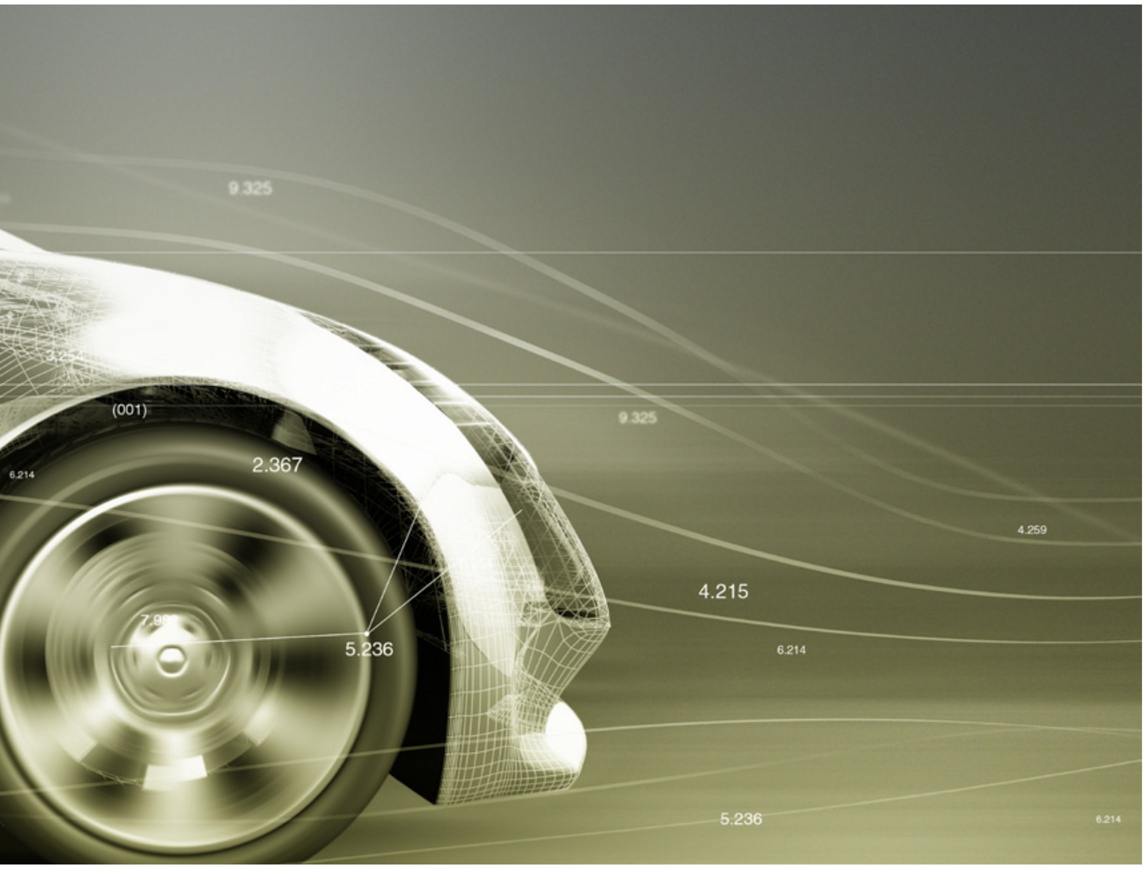
To sum up, there are two key elements used to fulfill all the requirements for performance, safety, development and integration processes, as well as modularity and scalability:

- A deterministic architecture exploiting the time-triggered paradigm for (ECU-internal) network communication and task scheduling.

- A common, generic middleware layer abstracting the diversity of the underlying microcontrollers and operating systems.



It should be pointed out that this approach is completely scalable, as processing hosts may be added for different ECU variants or car models without having an impact on the overall architecture and allowing the complete reuse of SWCs across those variants. The same architecture can even be used in a distributed way at vehicle level, e.g., by using OABR Ethernet interfaces between ECUs. It is also remarkable that all the core elements build upon well-established and standardized technologies (such as AUTOSAR®, POSIX®) and time-triggered communication, as defined in SAE AS6802 and the current TSN (Time-Sensitive Networking) activities within IEEE.



Middleware Functions and Benefits

This section will give a more detailed overview of middleware functionalities, their usage and benefits.

Communication: As mentioned above, the communication paradigm implemented by TTTech is based on an AUTOSAR® RTE, with ultimately all internal signals mapped to Ethernet communication. The communication paradigm uses off-the-shelf Ethernet communication stacks provided by the OS suppliers, even though there are some specific enhancements to incorporate the deterministic, time-triggered approach. The SW integrator generates the communication layer at compile time, on the basis of signal information provided by the OEM and the SWC suppliers. All communication is made available to the outside world via the Deterministic Ethernet switch, thus giving total transparency for debugging and function performance analysis (e.g., inputs and outputs of the fusion layer can be made accessible without having any timing impact on the system). A benefit of the time-triggered approach is that throughput and latencies are known in advance and do not vary depending on SWC behavior. This also allows the off-loading of SWCs to a PC via Ethernet (co-simulation) for prototyping or debugging purposes, while keeping all timing relations intact.

Synchronization and Time-Triggered Task Scheduling: As is the case for communication, all major task activation is done in a deterministic way on the basis of a predefined schedule. Violating SWCs that do not comply with their runtime restrictions are controlled and deferred. In some situations, this approach may seem too strict. Therefore, additional flexibility is built in. SWCs may utilize other processing slots which are not fully used by the respective SWCs assigned to these slots, and SWCs can (dynamically) adjust their operation at runtime to the time remaining in their scheduled slot. There are special API functions to enable both functionalities, which can mainly be used by functions that do not have a hard computational target. A fusion layer, for example, might adjust the number and granularity of identified objects depending on the available runtime.

Safety: The middleware along with the underlying OS and HW mechanisms is conceptually implemented as a Safety Element out of Context (SEooC) in the sense of the ISO 26262, essentially providing a safe ►

execution environment for each SWC and a safe communication channel for other SWCs. Generic safety mechanisms, such as memory protection, timing supervision and end-to-end communication protection between SWCs on the same host and to remote hosts are derived from the established AUTOSAR® mechanisms, and adapted and extended to non-AUTOSAR® environments by making use of functionalities that are provided by the respective operating systems and their host processors. Additional host-specific (HW depending) diagnostics and supervision functions, such as clock, power supply, memory and µC core monitoring are implemented as necessary.

Storage and Data Management: Non-volatile and volatile memory is made available to the SWCs on each host in a location-transparent manner. This means that data written by an SWC on one host is transparently made available to all the other SWCs on all hosts. Generic API functions as a part of the middleware ensure storage, protection, data transport and retrieval across the ECU. This is again an essential feature to provide complete portability and location transparency of application SWCs. In fact, the SWCs are not even aware of where data is actually stored – and do not need to care. This includes applications running on a PC (SIL scenario).

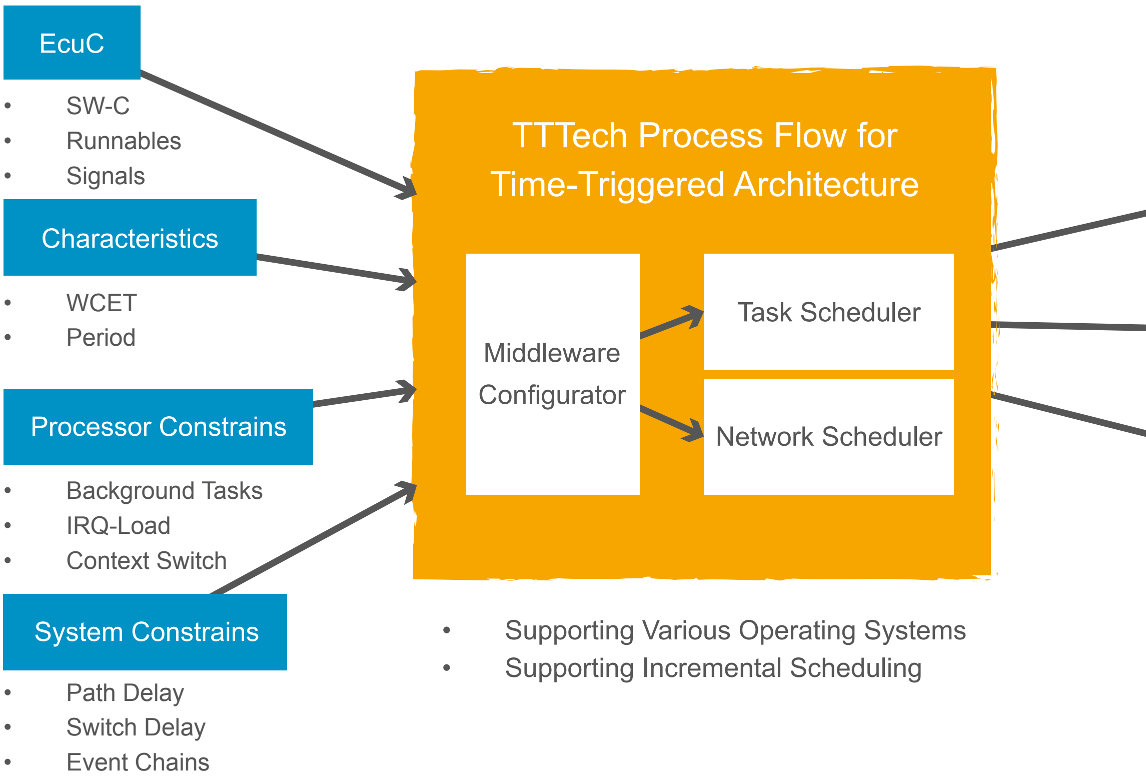
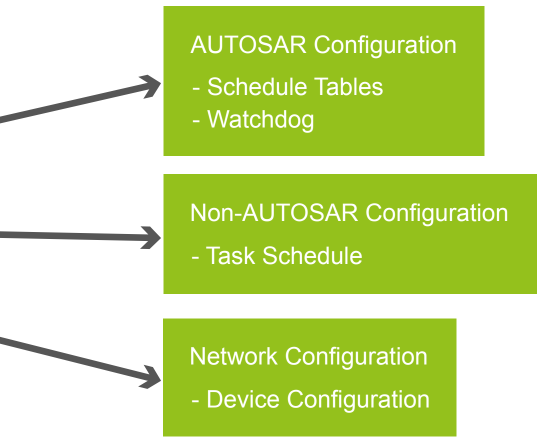


Figure 4: Platform Configuration Workflow

Diagnostics, Calibration, Flashing: These utility functions are implemented in a master/slave approach. The AUTOSAR®-based microcontroller acts as a master, controls the other processing elements (slaves) and presents the whole ECU as one homogenous entity to the outside world. There is no need for a special diagnostics or calibration tool to cope with the internal diversity and complexity of the domain ECU. OEM and SWC suppliers can just keep their existing tools and workflow.



SW Development Support: There is a full set of debugging and profiling tools that have mostly been derived from the specific OSs but were enhanced by generic mechanisms (e.g., data logging). The PC-based co-simulation environment can not only be used as a convenient way to integrate, under ADTF or MATLAB®/Simulink, prototyped SWCs with SWCs running on the target ECU. Identical C source code can be run on a PC and on the target ECU without any timing-related impact, as the time-triggered mode of operation abstracts the usually much higher performance of the PC. Therefore, rich PC-based debug facilities may be used even for fully industrialized components, which would only offer very limited access to debugging and tracing facilities in a traditional workflow (on the embedded target).

Most of the middleware, in particular communication, is generated by a set of highly integrated tools and scripts. The communication matrix and various additional information from OEM and SWC suppliers (in particular, resource budgets, latency constraints, and data flow constraints) serve as main inputs for a complete platform build. Tools from classic AUTOSAR® Basic SW (BSW) vendors and OS suppliers are fully integrated to configure their respective components. ■

SWC Development and Integration Workflow

In a traditional, ad hoc integration workflow, the respective suppliers deliver all the SWCs. The SW integrator builds a binary flash image for each host and loads it into the system. Resource constraints (processing time, memory, communication bandwidth) are hopefully aligned before, but initially rarely met by the SWC suppliers. As a result, SW integrators find themselves in a situation where total resource consumption almost for sure is out of limits, thus causing conflicts among SWCs. Overall functionality and stability will depend on the specific inputs: Even though simple environmental scenes might be treated smoothly, complex scenes will bring the system beyond its limits, leading to spurious faults and inconsistencies. A lot of mediation by the SW integrator and refinements by the SWC suppliers will be required to iteratively bring the system to a stable state. And what is worse: Once the system is working, even minor changes by one of the SWC suppliers can destabilize the complete system again. This is a nightmare not just for the SW integrator, but also for the OEM verification and validation tasks at vehicle level.

Our architecture avoids this scenario altogether and provides a stable, controlled process from individually tested SWCs to a completely integrated system. The deterministic, time-triggered scheduling of all communication and task activation provides the technical basis. This approach might seem rigid at first, but in essence just ensures the constraints that have to be eventually met anyway. So we are dealing with a conscious design step of strict resource planning and scheduling to achieve overall system composability with the added benefit that all timing behavior is known a priori. Thanks to our approach, even data flow latencies are known precisely and in advance, which in a traditional workflow would have to be measured a posteriori or investigated by simulation, but would still be prone to variability and jitter.

The actual SW integration workflow is carried out in three phases per SW release (see Figure 5), which may be triggered by OEM-defined SW integration steps and based on a functional feature release plan that is synchronized among the SWC suppliers.

1. Platform Release: The SW integrator configures and builds the platform SW, including all BSW, OSs and middleware and releases it to the SWC suppliers. Resource boundaries (timing, memory) are predefined and aligned for each SWC and implemented through an appropriate task and communication sched-

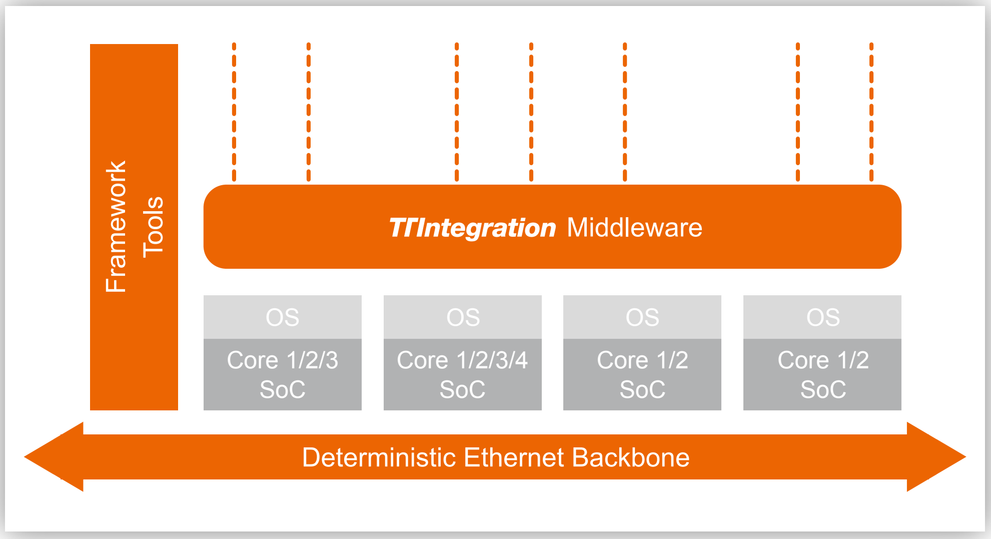
ule. All SWCs are “stubbed”, i.e., replaced by empty templates and dummy functions.

2. (Single) SWC Integration: The SWC supplier integrates the application on the platform by replacing the stubs and can test the SWC within its predefined boundaries, in exactly the same environment and with exactly the same timing as on the final integrated system. The tested SWCs along with the used test cases are delivered to the SW integrator for the system release.

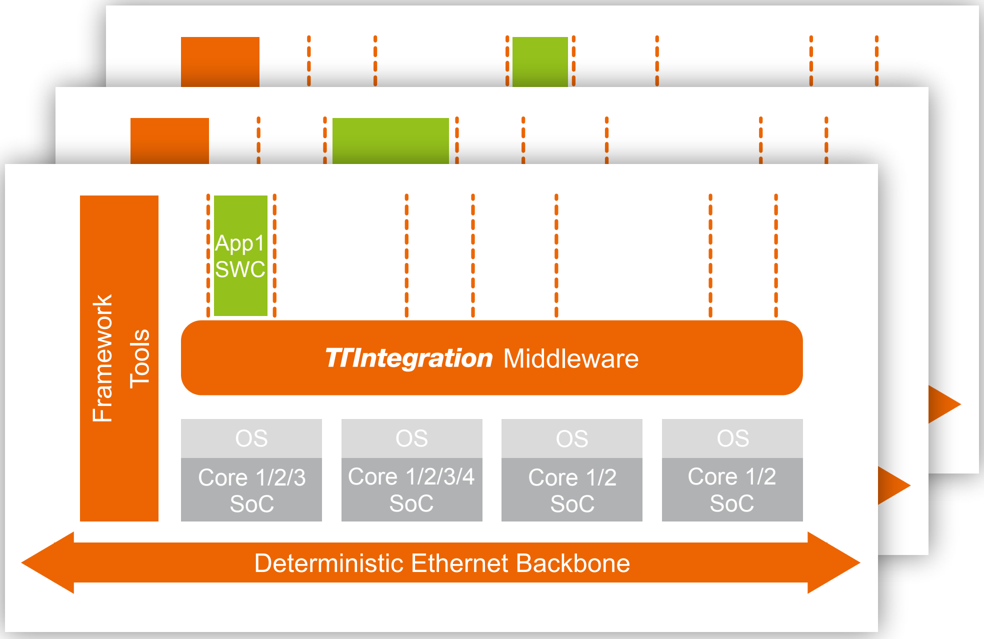
3. System Release: The SW integrator integrates all SWCs and delivers the complete build to the OEM and to the SWC suppliers for vehicle or test bench integration. All the SWCs will run exactly as in the single SWC setup (step 2), all test vectors are reused and results remain valid. The whole system is immediately stable, without any further measures.

Actually there is also a step 0 that provides a so-called SIL release – an initial PC-based environment for SWC development that is independent of the target. ▶

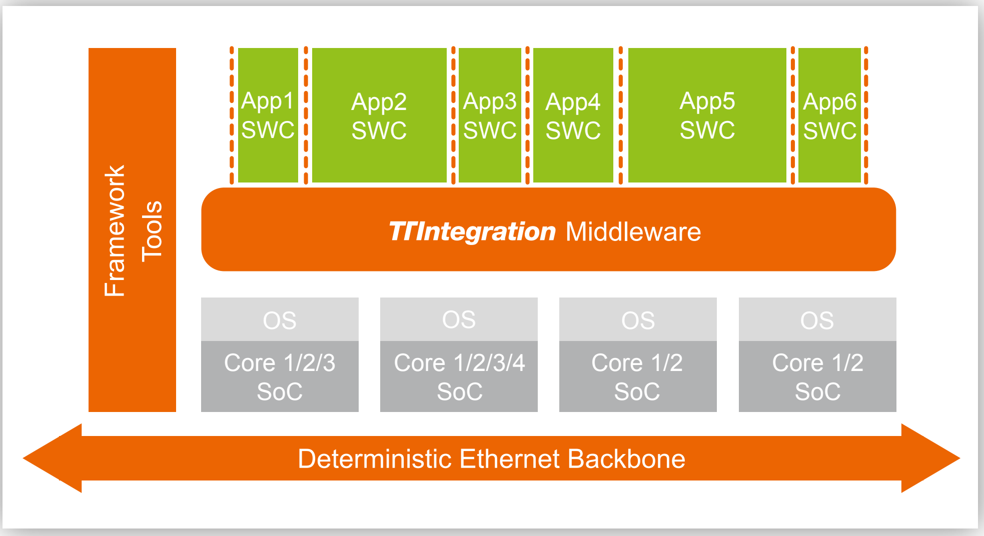
Figure 5: Software Integration Process



Phase 1: Software Release



Phase 2: (Single) SWC Integration



Phase 3: System Release

At step 3, SWCs are checked if they fulfill their resource boundaries before integration ("acceptance test") and possibly rejected if they do not[3]. The system may not be in a fully working state if other SWCs depend on the violator, but at least this is known in advance and corrective measures can be taken. In a traditional architecture and workflow, such violations might at first go unnoticed with simple scenes and low computational demand and, thus, lead to an unstable system state later, for example, when it comes to complex environmental scenes, which is a situation much worse to deal with.

Each supplier can use the system release as a basis for further feature development and improvement, by incremental builds to upgrade their own SWC, as long as it obeys the defined boundaries.

The particular benefits and results of this controlled integration process (and the HW/SW architecture and middleware approach enabling it) can be summarized as follows:

- A running system without timing or memory collisions from SWCs is generated in one predictable, well-defined integration process.
- The actual runtime requirements of all the SWCs are known and collected in an integration report.
- The actual data flow latencies and sequences of SWC activations are known and collected in an integration report (these figures are actually known already after the first step – scheduling/platform release).
- Data flow latencies and all timing relations are not only known exactly, but fixed. There is no variability or jitter, which drastically reduces the amount of application testing and application validation.
- Side-effects from SWCs, such as the blocking of other SWCs, are avoided. ■

3
In an ideal world, this should not happen. However, to expedite SWC development in early project phases, strict timing checks might still be turned off because not all SWCs are available yet and spare time budget might be implicitly released to the existing SWCs.

Experiences From Series Development

The above-mentioned HW architecture was implemented in a real series project with a very low impact on ECU costs. The only additional device needed, besides the various processing engines, was the Deterministic Ethernet switch, which was integrated in an FPGA, whose processing capabilities and interface flexibility would have been needed anyway. An alternative is NXP's SJA1105T.

The ECU was implemented in several variants to support different option packages, ranging from automatic emergency braking only to autonomous driving functions. Due to the HW and OS abstracting properties of the middleware, SWCs are not aware on which variant they are running and may even be located on different hosts for different ECU variants. In addition, SWC suppliers do not have to spend any extra effort to support several ECU variants.

The SW Integration process was smoothly carried out by a surprisingly small team. The SWC suppliers quickly adapted to the new process, because it was clearly defined and straightforward, thus giving SWC suppliers a well-defined development

and integration platform with a multitude of useful features and relieving them from most of the housekeeping functions required in a traditional setup. SWC suppliers can simply focus on implementing the required algorithms, with the required basis being delivered by the platform and middleware. The necessary precautions (estimating resource consumptions of the SWCs and aligning resource constraints system-wide in advance) are needed in any series development and thus do not cause any additional effort.

The capability to move SWCs among different processing hosts without any code change is used quite frequently, especially as continuous refining, learning and optimizing over the SW integration steps reveal too optimistic initial resource estimates for some SWCs, which are offset by too pessimistic estimates for other SWCs, thus leading to changes in the allocation of SWCs over the course of the project.

Currently, more than 30 applications from more than 10 suppliers are integrated, ranging from sensory processing, various sensor fusion components to a multitude of customer functions, such as automated parking control and automated driving. It is our firm conviction that this project, with its extreme technical and organizational complexity, would probably not have succeeded without the implementation of a deterministic architecture, generic middleware and a strict integration process. ■

Conclusion

This paper presented a new architecture and a new integration approach to highly-integrated domain ECUs, which is currently being successfully implemented in the series development of a complex central ADAS controller that contains a diverse set of processing units, ranging from traditional automotive microcontrollers to high-end graphics engines. Two key elements enhance the architecture to fulfill all requirements set on performance, safety, ECU modularity and scalability, and the development and integration process:

> A deterministic HW and SW architecture, including a Deterministic Ethernet switch, exploits the time-triggered paradigm for ECU-internal communication and task scheduling, thus enabling predictable timing behavior of application SW components and fully integrated systems.

> A common, generic AUTOSAR® RTE-based middleware layer abstracts the diversity of the underlying microcontrollers and operating systems for the application SW components.

This architectural basis is implemented with an exhaustive feature set of debugging and utility functions. It ensures full portability and location transparency for the application SW components and, therefore, full flexibility with respect to SWC allocation and system partitioning.

The integration process enabled by the deterministic, time-triggered paradigm guarantees full predictability and composability and ensures smooth SW integration without iterative, time-consuming and costly integration hassle of traditional ad hoc approaches. ■

“With TTTech’s high performance ADAS platform we are already set today for all future driver assistance systems.”

Georg Niedrist has steadily gained expertise in the development and implementation of innovative networking and computing platform technologies throughout his professional life.

After completing his studies with a doctor’s degree at Technical University Vienna, he worked for more than ten years at Siemens AG in Vienna, being responsible for the development of highly reliable telecommunication switches. Additionally, he also accompanied customer projects as a project manager.

Since joining TTTech in 2004, Georg Niedrist has been implementing customer projects in the areas of automotive, aerospace and off-highway vehicles, thus broadening his technological experience tremendously while becoming acquainted with different customer requirements.

In 2008 he started focusing on the automotive area and became TTTech Automotive’s technical head, in this position he is responsible for all products and project handling.

Dr. Georg Niedrist

Director Products & Projects at TTTech Automotive GmbH

Taking the Right Turn with Safe and Modular Solutions for the Automotive Industry

TTTech Automotive GmbH
Schoenbrunner Strasse 7
1040 Vienna, Austria
Tel. + 43 1 585 65 38-5000
Fax + 43 1 585 65 38-5090
products@tttech-automotive.com

Ensuring Reliable Networks TTTech