Software development

Do AUTOSAR and functional safety rule each other out?

While simplicity is a factor in safety-critical applications, AUTOSAR has over 6,000 configuration parameters and well over 100,000 lines of code, providing an unimagined range of options. This vast scope of configuration variants makes a safety analysis and the implementation of the verification measures that are necessary in highly safety-critical applications (ASIL-C, D) practically impossible. However, there is a solution to this dilemma: to divide the software into safety related and non-safety related components and to ensure that both are strictly separated.

By Dr. Alexander Mattausch

An increasing number of electronic control units (ECUs) incorporate software-implemented safety mechanisms based on AUTOSAR. They rely on the functions provided by the AUTOSAR basic software (BSW). However, these are generally developed according to quality management (QM) guidelines only. So how can we build a safe software architecture on this basis? The only practical solution is to divide the entire software up into small units. As a result, only a few of the software components have to meet safety requirements, vastly reducing the quantity of code to be verified according to the highest safety integrity level. This simplifies the development process, reduces complexity and improves safety.

The AUTOSAR stack itself can also be broken down into safety related and non-safety related components. The difficulty is to ensure the freedom from interference between various components as required by the ISO 26262 standard. It is necessary to ensure that there is no interference between safety related components and the rest of the software and/or that any possible interference is reliably detected. This principle of strict partitioning also applies when software components with different ASILs (Automotive Safety Integrity Levels) are integrated in one ECU. Only when interference is prevented can the safety relevant components be kept as small as possible and strictly encapsulated.

Freedom from interference is the key

ISO 26262 differentiates between three types of freedom from interference:

- spatial freedom from interference
- temporal freedom from interference
- exchange of information

The aspect of spatial freedom from interference refers to ECU's memory. Data in safety related components have to be protected against access from other components.
In order to have correct data at all times, a safety related component can, for example, store its data doubly – normally and bit-inverted – in the memory and check for consistency each time the data is accessed. However, this only permits the recognition of modified data. It doesn't prevent the data being modified. Alternatively, the data can be protected with the help of the Memory Protection Unit (MPU) which is available in modern processors. It recognises and prevents forbidden access, so the application can react to errors with the assistance of the operating system kernel. This is the preferred option for the protection of large amounts of data and particularly for higher ASILs.

Temporal freedom from interference relates to the timing domain. It is necessary to ensure that the safety related software has the necessary computing time and is executed in the expected order. In general, this is ensured with the assistance of external intelligent watchdogs. If certain checkpoints aren't reached in the prescribed sequence at the given times, this is recognised as an error. The activation of application tasks also falls within this type of freedom from interference: Priority-based scheduling is essential for the correct execution of the tasks in the predefined sequence. However, the operating system can thereby influence the safety related application, which means that it 'inherits' the ASIL of the application.

The third freedom from interference aspect that has to be ensured pursuant to ISO 26262 is exchange of information. Safety related data and signals have to be protected in such a way that corrupted or missing data is recognised.

An important prerequisite for all these functions is, however, that the software can rely on the processor. The processor has to be able to recognise errors itself with the diagnostic coverage that is necessary for the required ASIL. For lower ASILs such as A or B, self-test software which is provided by the processor manufacturer and periodically invoked often suffices. For higher ASILs, lockstep processors with ECC memory are commonly used. This involves two processor cores automatically executing all calculations in parallel and reporting an error when the results deviate from each other. Memory errors are recognised when when the CPU detects a wrong ECC checksum upon memory access. One important basic rule is that if a processor doesn't satisfy the prerequisites for the required ASIL, this cannot be compensated by the software.

The operating system kernel as central component

Using the described freedom from interference mechanisms, the AUTOSAR basic software can also be decoupled from the rest of the software on the ECU. The basic software then runs encapsulated in a separate memory partition and the various software components on the ECU are protected from each other with the help of the MPU. The central component in this setup is the operating system kernel (see Fig. 1).

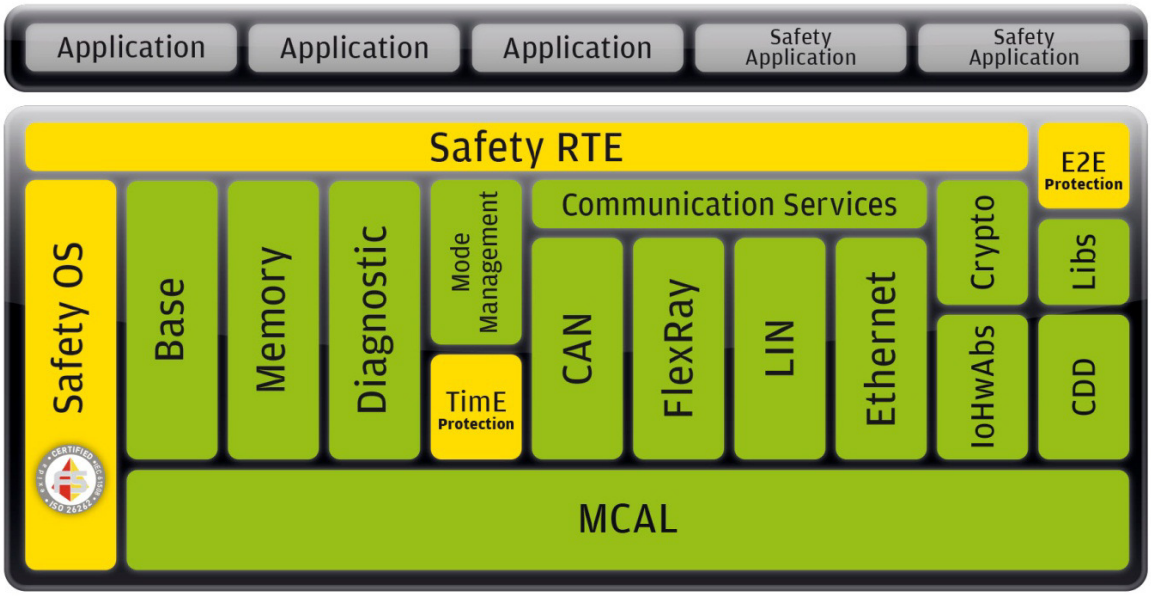

Fig. 1. *AUTOSAR architecture with EB tresos Safety products (yellow). The smart selection of modules developed pursuant to ISO 26262 makes it possible to keep the number of safety relevant codes as small as possible.*

Since the kernel knows which task has to be activated, this is the best location for the MPU management. If the task and interrupt handling is combined with MPU programming and developed according to the highest ASIL in the system, the safety mechanisms can thus be placed directly on this kernel. They are then also protected from unauthorised data access by the appropriate MPU configuration. Safe systems such as the EB tresos Safety OS provide the basic functionality of an operating system together with flexible memory partitioning, all developed entirely in accordance with ASIL-D. Since the kernel is based on a system call interface and also protects itself by memory protection, it is completely detached from the application. The kernel differentiates between privileged and non-privileged CPU mode with reduced rights, such as those offered by modern processors. This design makes it extremely robust against all types of application errors.

However, an operating system isn't safe merely as a result of a combination of task and interrupt management and memory protection. Complex software architectures use shared data areas which have to be protected against simultaneous access as well as notification mechanisms which inform other dormant software components about any incidents that occur. Particularly the AUTOSAR Runtime Environment (RTE) often uses these mechanisms when

software components communicate with each other. This is why EB tresos Safety OS also offers the OSEK resource mechanism which, with the assistance of the Priority Ceiling protocol, enables constitutes the deadlock-free implementation of a locking mechanism. The OSEK events provide functions for notifying dormant tasks according to ASIL-D. This offers application developers the advantage that all basic mechanisms required for the functionality of a complex software architecture are available in an ASIL-D variant and no further protection is necessary within the application. Especially the often implicitly assumed basic functions of an operating system - context switching and interrupt handling - are part of the ASIL-D kernel. The application developer can then focus on the essentials, i.e. the application's functionality.

The basic software also benefits from this approach. It can also be viewed as an application that is 'locked in' by a small operating system kernel. Since modern processors usually permit selective access to the peripherals in the non-privileged mode, there is no reason why the BSW shouldn't be operated with reduced rights and active memory protection. This ensures freedom from interference to the other system components and, at the same time, the safety related parts of the software are thus only the kernel and the safety mechanism in the application.

How can this be implemented in practice? Many car manufacturers deliver software components for OEM-specific functions which use the AUTOSAR stack. For example, EB offers a BMW-specific AUTOSAR package in which the basic software and the BMW components run in non-privileged mode of the CPU with active memory protection. On the Leopard processor of Freescale and ST Microelectronics on which reference implementation has taken place and now is going into series production in projects up to ASIL-D, there is also additional support from the integrated Peripheral Protection Module. This allows to selectively enabling the peripherals for access in non-privileged mode. In conjunction with memory protection, basic software, application and operating system are strictly separated in this setup.

Control flow and time monitoring

Another important aspect of protecting safety related software is temporal freedom from interference. The kernel merely handles task activation, but not compliance with timing constraints. Since these are closely linked to the application's safety requirements, monitoring them in a separate software component or directly in the application is much better than monitoring them directly in the kernel.

In safety development, time monitoring usually has to satisfy three basic requirements. First and foremost, it has to recognise whether the software is functioning or not. This is referred to as 'alive supervision'. Another key monitoring task is 'deadline monitoring', i.e. ensuring that safety related actions are performed within a defined timeframe. Finally, deviations from pre-defined execution sequences have to be identified. This is called 'control flow monitoring'. Under some circumstances, both can require finer granularity than is offered by the operating system kernel with its task activation mechanisms. This is another argument in favour of outsourcing into an own module.

In the EB tresos product family, EB tresos Safety TimE Protection provides the necessary functionality (see Fig. 2).

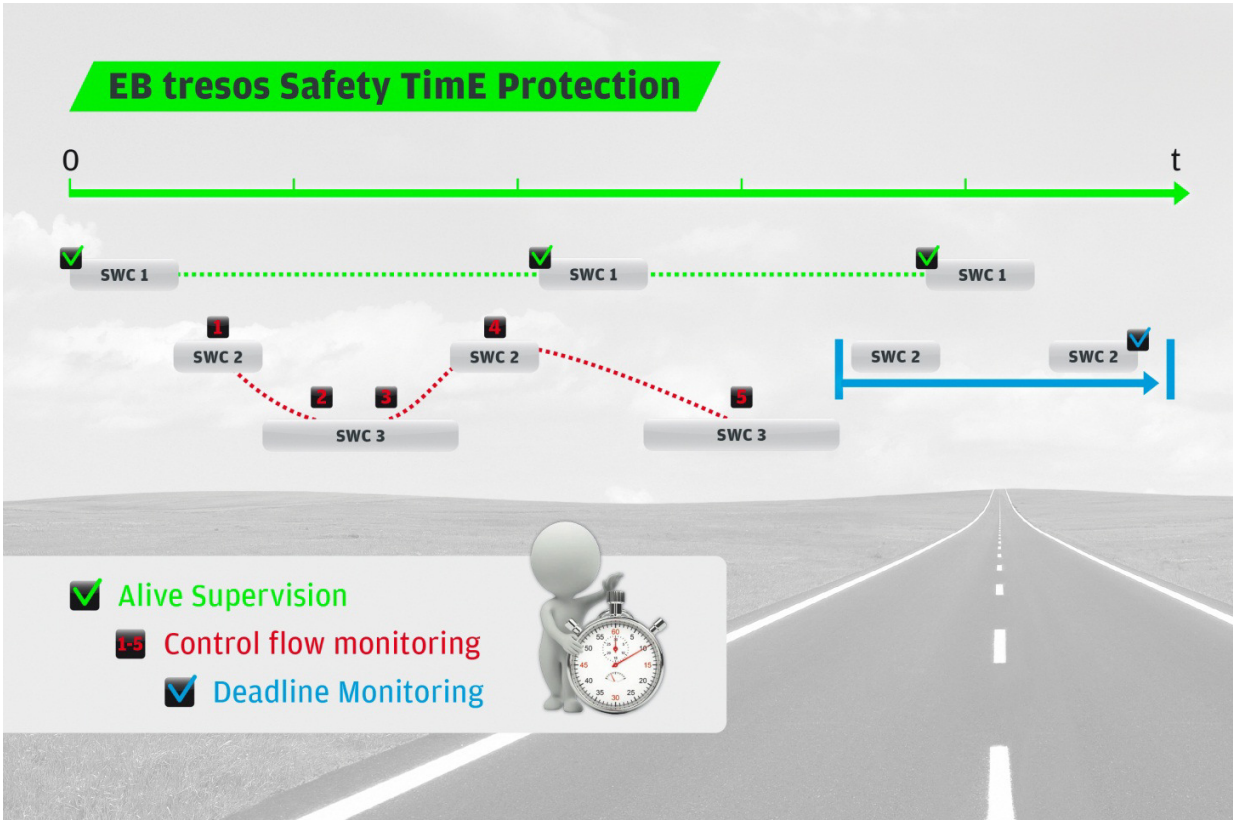

Fig. 2. *EB tresos Safety TimE Protection in conjunction with a watchdog provides alive supervision through the monitoring of defined checkpoints, control flow monitoring through sequence checks and deadline monitoring through comparisons with maximum time allowed. The diagram shows correct behaviour without errors.*

It implements the AUTOSAR watchdog stack and, with the help of an external hardware watchdog, it ensures that any deviations from the predefined call sequence or any deadline breach initiates a switch to a safe mode. To do this, the module uses a time stamp that is either provided by the application developer or the operating system kernel.

For applications with only a small amount of safety related data or for low ASILs, the freedom from interference could be argued via control flow monitoring alone. If no safety requirements are directly allocated to the operating system, checksums or redundant storage can be used to ensure data integrity. The developer can monitor the correct execution of the safety mechanism with checkpoints at suitable positions and the control flow monitoring mechanism. However, this approach reaches its limits as soon as the safety mechanism's complexity and the volume of data to be protected increase. Furthermore, high safety requirements and the associated higher ASIL values require a more robust system, so that control flow monitoring as the only means of protection is no longer sufficient. Therefore, projects with ASIL-C or D use a combination of the

Safety OS for memory partitioning and control flow monitoring to ensure temporal freedom from interference.

Runtime environment and communication

In modern ECUs, applications aren't generally run directly on the operating system kernel, but are developed as AUTOSAR Software Components (SWC) using the AUTOSAR Runtime Environment (RTE). The basic software and processor hardware are abstracted via the RTE to enable the simple portability of the application. Although the RTE is only an abstraction layer placed on the kernel and the BSW-APIs thereby providing the functionality of other AUTOSAR modules in the application, it facilitates communication between the individual software components both within the ECU and via the network. If the data to be communicated are safety related, there are two different protection options.

Within the ECU, protection can be realised directly via the RTE. In the EB tresos product family, this is performed by EB tresos Safety RTE. Here the most important safety related function is exchange of information, because the RTE performs this task itself rather than delegating it to other modules. The RTE uses the operating system for the activation of other software components or the protection of data, thus it is only necessary to verify that the OS API has been correctly used. Of course, the basis for this is an operating system developed according to the appropriate ASIL.

For the communication across ECUs, there are many error sources which can result in data transmission failure. These sources are often outside the ECU, which means that proper software development cannot prevent errors in this case. Error recognition is the only solution, which is why AUTOSAR has specified the E2E module (end-to-end protection) (see Fig. 3). With the assistance of checksums and message counters, this module recognises data packets that have been lost, transmitted twice or transmitted incorrectly.

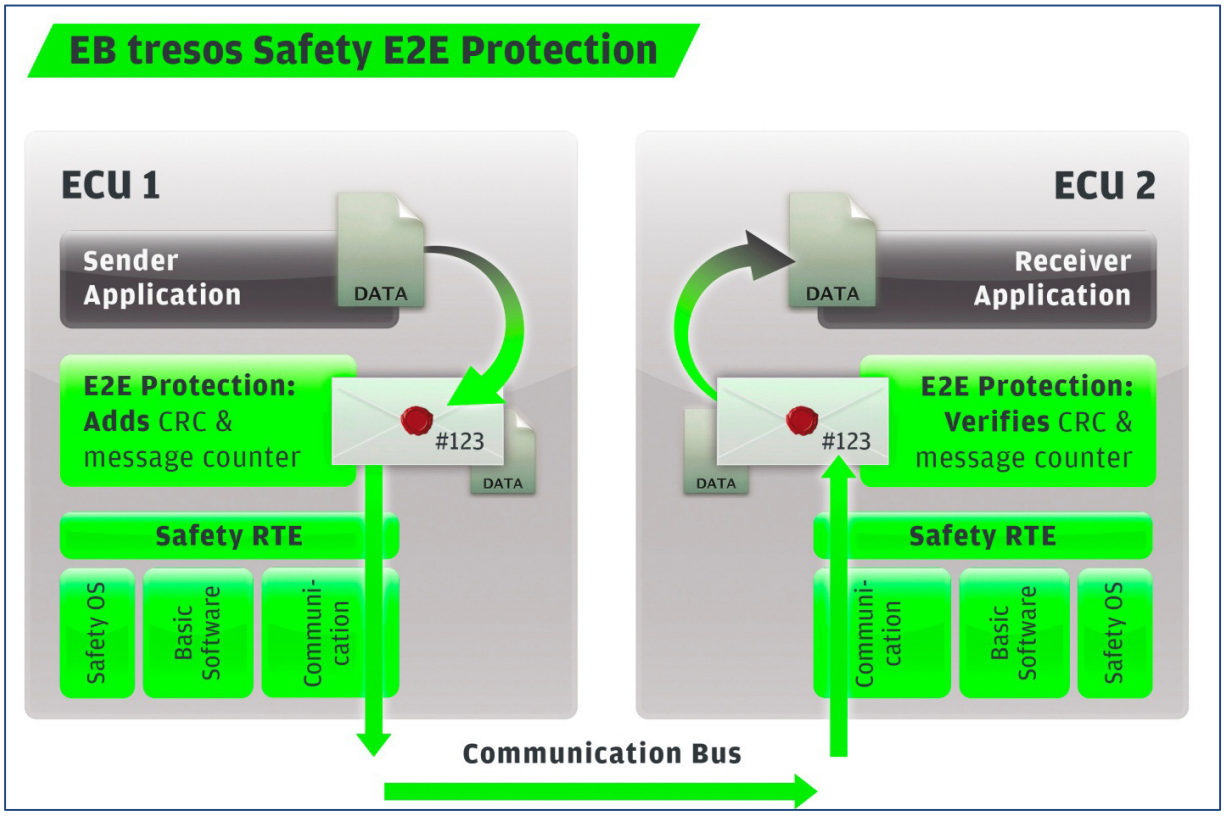

Fig. 3. *Safe communication can be provided with checksums (CRC) and message counters. Corrupted or missing messages are reliably identified by EB tresos Safety E2E Protection.*

AUTOSAR and safe ECUs

If you restrict yourself to the absolutely necessary AUTOSAR modules and 'lock in' the highly complex AUTOSAR basic software and application software, AUTOSAR allows for development up to the highest ASIL. While the application and the safety mechanism both are still responsible for verifying and controlling the data that are read in, AUTOSAR and, in particular, the EB tresos Safety product family provide mechanisms ensuring the functional integrity. The loss of data integrity can be recognised and prevented with the operating system kernel, while timing errors can be identified with alive supervision, deadline monitoring and control flow monitoring. Safety RTE and E2E protection supplement the package by protecting communication. The reduction of safety components to their basic functionality and their encapsulation from the other AUTOSAR functions ensures that these modules remain as small and simple as possible. Therefore, despite the high complexity of AUTOSAR, it is possible to find simple solutions and develop safe ECUs.

Fig. 4. *Automotive Safety Integrity Level (ASIL) rates the safety relevance of errors on a scale from A to D according to the three criteria of driving situation frequency of occurrence, controllability of a situation when a function error occurs and severity of the consequences.*

Dr. Alexander Mattausch
studied physics at the University of Erlangen-Nuremberg and obtained his PhD in Theoretical Solid State Physics there in 2005. He has been working for Elektrobit Automotive GmbH as Senior Project Manager for Operating System Development for automotive ECUs since 2007.