TASKING

Freedom From Memory Interference

How Safe and Secure Are Current Defences?

INTRODUCTION

This whitepaper is published in two parts. The first part which your are reading now addresses the current state of the art regarding memory interference detection and recovery. First the concept "freedom from interference" is introduced. Subsequently the new challenges regarding avoidance of memory interference imposed by multi-core architectures are explained. The first part ends with a detailed look at the memory interference prevention and recovery guidance from ISO 26262 and explains how the guidelines affect the safety mechanisms implemented in today's automotive microcontrollers and operating systems.

The second part of this whitepaper explains a novel technique to detect and repair memory interferences "ASIL Aware Static Analysis of Memory Interferences" and shows how it is implemented in the TASKING Safety Checker tool. This tool facilitates memory interference detection at software build time, whereas today's hardware and operating systems interference detection mechanisms operate at runtime and possibly detect failures not earlier than after a product has been shipped to the end user.

Although this paper is based on ISO 26262 and addresses specific requirements of the automotive industry the text is also relevant for other application domains where embedded software is developed in accordance to safety standards such as IEC 61508, ISO 25119, or EN 50128.

FREEDOM FROM INTERFERENCE (FFI)

ISO 26262 defines freedom from interference as the "absence of cascading failures between two or more elements that could lead to the violation of a safety requirement". An "element" is a "system or part of a system including components, hardware, software, hardware parts, and software units". "Cascading failure" is a "failure of an element of an item causing another element or elements of the same item to fail".

As shown in the illustration below from ISO 26262-1, element 1 is free from interference from element 2 if no failure of element 2 can cause element 1 to fail.

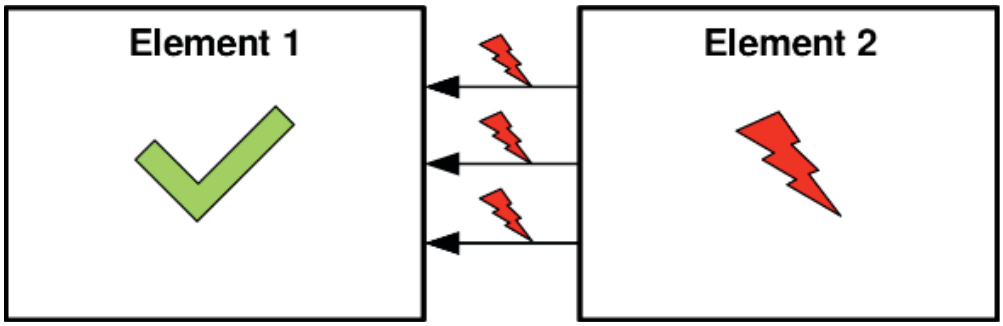

Illustration 1: Element 1 is Free from Interference. Copyright © ISO.

Element 3 interferes with element 4 if a failure in element 3 causes element 4 to fail.

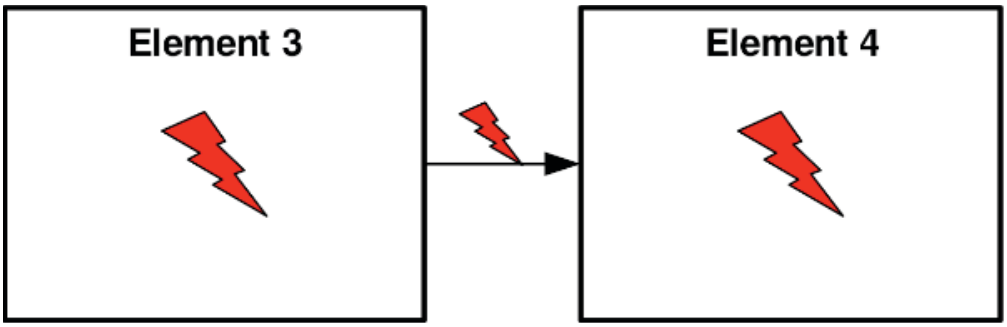

Illustration 2: Element 3 Interferes with Element 4. Copyright © ISO.

TYPES OF INTERFERENCE

ISO 26262 specifies three types of interference: timing and execution, memory, and exchange of information. Interference in the time domain occurs when execution of a safety-relevant software element is blocked due to a fault in another software element. This type of problem is detected and handled by what is called a watchdog - a combination of hardware and software, often supported through the operating system (RTOS). Once an error is detected the RTOS tries to recover the system or bring the system to a safe state.

Illustration 3: Interference in Timing and Execution.

Interferences in the memory space domain occurs when a software element accesses or modifies code or data belonging to another software element. This type of interference is related to corruption of memory content and device configuration data. Such interferences are typically detected at run-time by a hardware component called the Memory Protection Unit (MPU). Once an error is detected, the MPU triggers an interrupt and the fault handler tries to restart all software elements located in the partition that caused the interrupt to occur.

Illustration 4: Interference in (Shared) Memory Space.

Interferences due to exchange of information are sender and receiver related and are caused by errors such as: repetition of information, loss of information, delay of information, insertion of information, blocking a communication channel, etc...

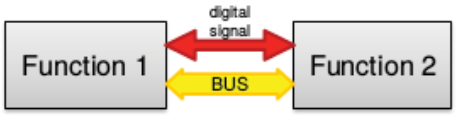

Illustration 5: Interference Due to Exchange of Information.

TRENDS RAISING THE IMPORTANCE OF FFI

In the past, new functionalities in applications were implemented using a separate electronic control unit (ECU). As a result, most modern cars contain somewhere between 50 to 150 ECUs. The large number of ECUs causes reliability and cost issues which limits the product's potential to scale. Today, there has been a shift from separate ECUs toward a more integrated architecture where multiple software elements are co-located on a single ECU, often referred to as a domain controller.

There are several advantages that such integrated architecture provides, including:

- More reliable and efficient designs are possible with multiple components easily communicating with each other on a single ECU.
- Reduced material costs with less ECUs, wires, and connectors required in the design.
- Reduced potential for hardware failures with less transference of information between ECUs.

However, there are also risks associated with such integrated mixed criticality systems, such as:

- Increased likelihood that fault in one software element could cause another safety relevant software element to fail.
- Added complexity in assuring FFI due to multi-core architectures.
- Higher chance of interference between software elements with shared global memory.

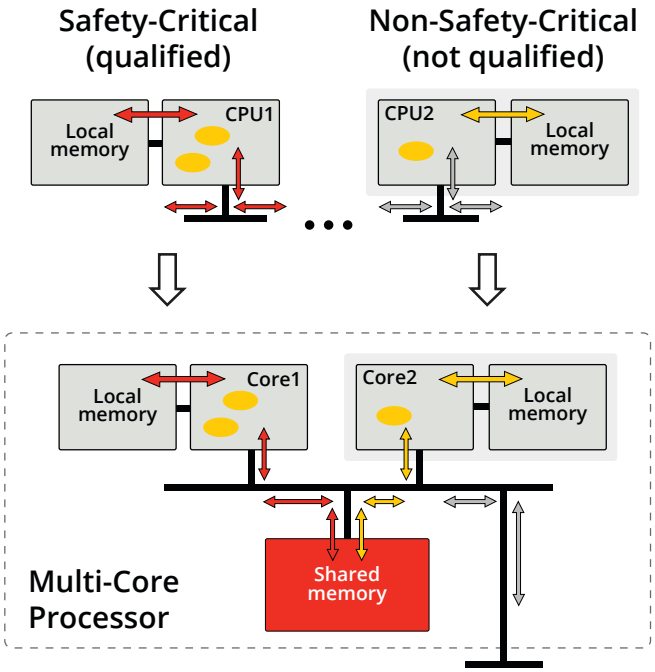

Illustration 6: Replacing a Federated Architecture by a Centralized Architecture Enables Memory Interferences.

Heterogeneous multi-core architectures further complicate the analysis of memory interferences. For example, the Infineon® Tricore AURIX device contains three Tricore™ processing cores - an ARM based HSM core, an X800 based standby controller, and a generic timer module (GTM) with several MCS cores. Interferences may occur in all cases where these cores and intelligent peripherals have access to shared memory or device control registers.

In such complex systems it is hard to verify whether the interference detection mechanisms implemented in the hardware are configured and operate correctly. It is even harder to verify whether the software running on all these components cause any memory interferences. The supply chain of automotive software adds an additional challenge to ensure freedom from interference.

For example, the development of an ECU requires interactions between the OEM (BMW®, Daimler®, GM®, PSA®) who provides the specification of an ECU to the ECU supplier (Bosch®, Continental®, Delphi®, Valeo®), the hardware supplier (Infineon®, NXP®, STMicrosystems®, Renesas®), and the supplier of the RTOS (Elektrobit®, ETAS®, Vector®). To add to this complexity, Interferences can occur between software elements that are developed by different suppliers.

The figure below demonstrates the complexity of the ECU software. It shows the directed call-graph of an ECU, with 150 software components, of which 14 are safety-relevant according to ASIL B. It includes over 1,000 assembly connectors and multiple edges between SWCs. The issue to be addressed here is “how to avoid ASIL inheritance of the 136 non-safety related software components in the system”? (As required by ISO 26262-6:7.4.10, and explained further down in this paper). The methods and tools used by developers to ensure freedom from interference have to deal with all of these challenges. TASKING Safety Checker provides tremendous help for situations like this.

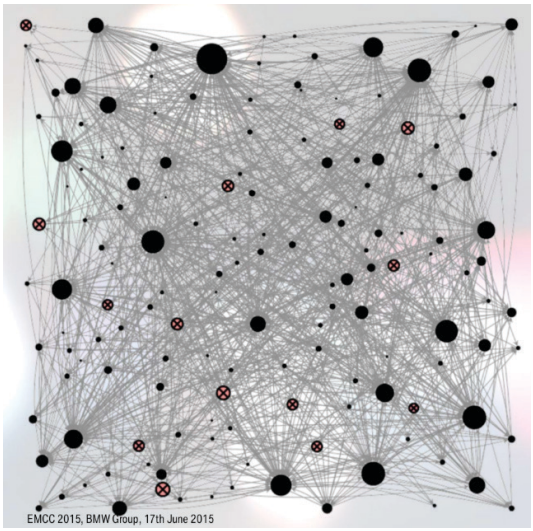

Illustration 7: Directed Call-Graph of an Integration Platform. Copyright© BMW.

(SOFTWARE) PARTITIONING

Partitioning can be used for fault containment to avoid cascading failures between software elements. It is a technique for providing isolation between software components, to contain and/or isolate faults, and potentially to reduce the effort of the software verification process. This method can be implemented in hardware, software and a combination of hardware and software.

Violations of software partitioning consist of typical error types that should be revealed during hardware/software integration testing. This presents a challenge as partitioning requirements are often negative (-), for example: "It shall not be possible for one partition to modify the data memory of another partition, unless that memory has been configured as shared." Since the partitioning mechanism is configurable, it can be difficult to verify that the test set is sufficient.

Furthermore, testing the partitioning mechanism determines whether interferences are detected and handled at run-time, but does not confirm that the software is free from interferences. This requires additional verification of the software architecture, the design, and its implementation.

ISO 26262 GUIDANCE REGARDING FFI AND PARTITIONING

The concept of "freedom from interference" is addressed in the following sections of ISO 26262:

- ISO 26262-3 Concept phase: clause 8.4.3 Allocation of functional safety requirements.
- ISO 26262-6 Product development at the software level: clause 7.4.11 Software architectural design - Software partitioning, and clause 7.4.12 Software architectural design - Analysis of dependent failures, and Annex D: Freedom from interference between software elements.
- ISO 26262-9 Automotive Safety Integrity Level (ASIL) - oriented and safety-oriented analyses: clause 6 Criteria for coexistence of elements, and clause 7 Analysis of dependent failures.

Guidance on freedom from interference is primarily goal based, while references to specific safety measures including methods, techniques or tools are limited. Here is an except from the standard:

"If the embedded software has to implement software components of different ASILs, or safety-related and non-safety-related software components, then all of the embedded software shall be treated in accordance with the highest ASIL, unless the software components meet the criteria for coexistence."

The criteria for coexistence are: if safety-related sub-elements with different ASILs, including QM, coexist in the same element, then a sub-element shall only be treated as a sub-element with a lower ASIL assigned if evidence is made available that, for each safety requirement allocated to the element, it cannot interfere with any sub-element with a higher ASIL assigned. Otherwise, this sub-element shall be assigned the highest ASIL of the coexisting safety-related sub-elements for which evidence of freedom from interference is not made available.

If software partitioning is used to implement freedom from interference between software components it shall be ensured that:

- The shared resources are used in such a way that freedom from interference of software partitions is ensured.
- For ASIL D, the software partitioning is supported by dedicated hardware features or equivalent means.
- The part of the software that implements the software partitioning is developed in compliance with the same or an ASIL higher than the highest ASIL assigned to the requirements of the software partitions.
- The verification of the software partitioning during software integration and testing is performed.

The option to develop the entire software according to the highest ASIL can be a good choice for simple SEs. However, if a sophisticated SE has to be verified in accordance to a higher safety level this has a large impact on costs, lead time and required skill sets. In such situations it may be better to ensure freedom from interference by providing evidence that for each safety requirement allocated to a SE, it cannot interfere with any other software element with a higher safety level.

HARDWARE MEASURES SUPPORTING FFI

Automotive microcontrollers support all safety requirements listed in ISO 26262. The illustration below shows the safety measures implemented in Infineon AURIX® devices that are designed for ASIL D and therefore contain a large array of safety measures.

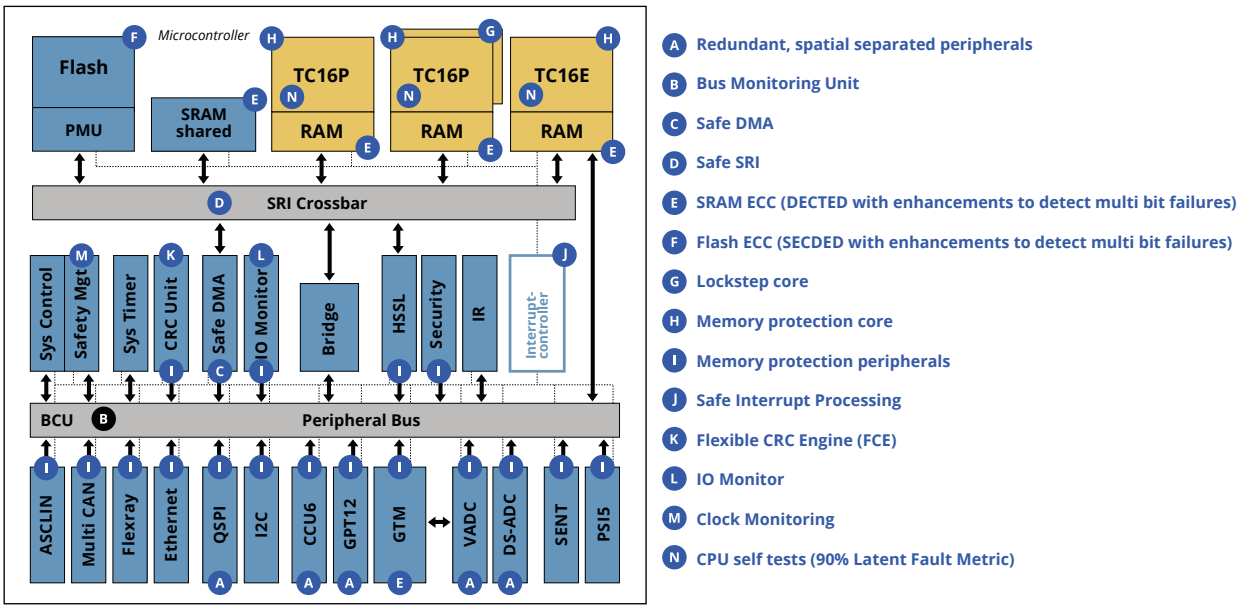

Illustration 8: AURIX® Safety Measures. Copyright © Infineon Technologies AG 2014.

The “Memory protection core” (H) and “Memory protection peripheral” (I) protect against interferences in the memory space and SFR domain. If such interference occurs the hardware generates an interrupt and the software tries to recover from the error. Notice that protection and error recovery takes place at run-time rather than build-time).

The number of partitions that can be serviced by the hardware is limited compared to the number of software elements located on an ECU (roughly 1:10). Because of this limitation, many software elements of the same ASIL have to be located in one partition. This results in a coarse-grained protection scheme where a software element is not protected against interferences from other software elements located in the same partition. RTOSes and hypervisors often use these hardware facilities to guarantee a reliable separation between partitions.

SOFTWARE MEASURES SUPPORTING FFI - RTOS AND HYPERVISOR

Run-time environments such as operating systems or a hypervisor can provide mechanisms to support freedom from interference. According to ISO 26262, such run-time environments should be developed with the same or higher ASIL than the highest ASIL assigned to the requirements of the software partitions. For ASIL D, the memory protection features of the underlying hardware should be used.

RTOS and hypervisors often use the memory protection features of the underlying hardware. A fine-grained protection model can be realized if the run-time system reprograms the memory protection facilities of the hardware, e.g. on task switches. This causes a trade-off between safety and performance, whereby the run-time performance will degrade significantly due to the large number of task switches that are typical for automotive (powertrain) software. Furthermore, an additional safety trade-off is introduced since MPU designs often contain safety measures to make the implementation more robust against transient hardware errors caused by electromagnetic interference and radiation. If the MPU is dynamically reprogrammed, the in-memory data structures that hold the memory access permissions are vulnerable to transient hardware errors that can break address mappings and permissions.

Some safety standards, like ISO 25119, state: “Partitions are statically specified, and are created at system start-up” and “programs running within a partition are always restricted to the resources allocated to the partition at system startup by the system configuration.” ISO 26262 does not provide explicit guidance about whether the memory protection system should be programmed statically (e.g. programmed once at startup time), or reprogrammed dynamically (e.g. during task switches). Be aware that the MPU’s safety measures guaranteeing the integrity of its configuration tables under more severe electromagnetic or radiation conditions are bypassed when the MPU is dynamically reprogrammed.

Safety standards for aerospace applications add explicit requirements to software-based fault isolation (SFI). Logical checks have to be added in the source code at each memory access point. The machine code in a partition is examined to determine the destinations of memory references and then checks their accuracy applications. Since indirect memory access cannot always be checked statically, instructions are added to the program to check the content of address registers at runtime immediately prior to use. TASKING compilers support runtime bounds checking and can be used to implement this requirement. When bounds checking is enabled every pointer update and dereference will be checked to detect out-of-bounds accesses, null pointers and uninitialized automatic pointer variables.

AUTOSAR FFI LIMITATIONS

The AUTOSAR partitioning concept, shown in the illustration below, is compliant with ISO 26262 and uses the safety measures of the underlying hardware to realize freedom from interference.

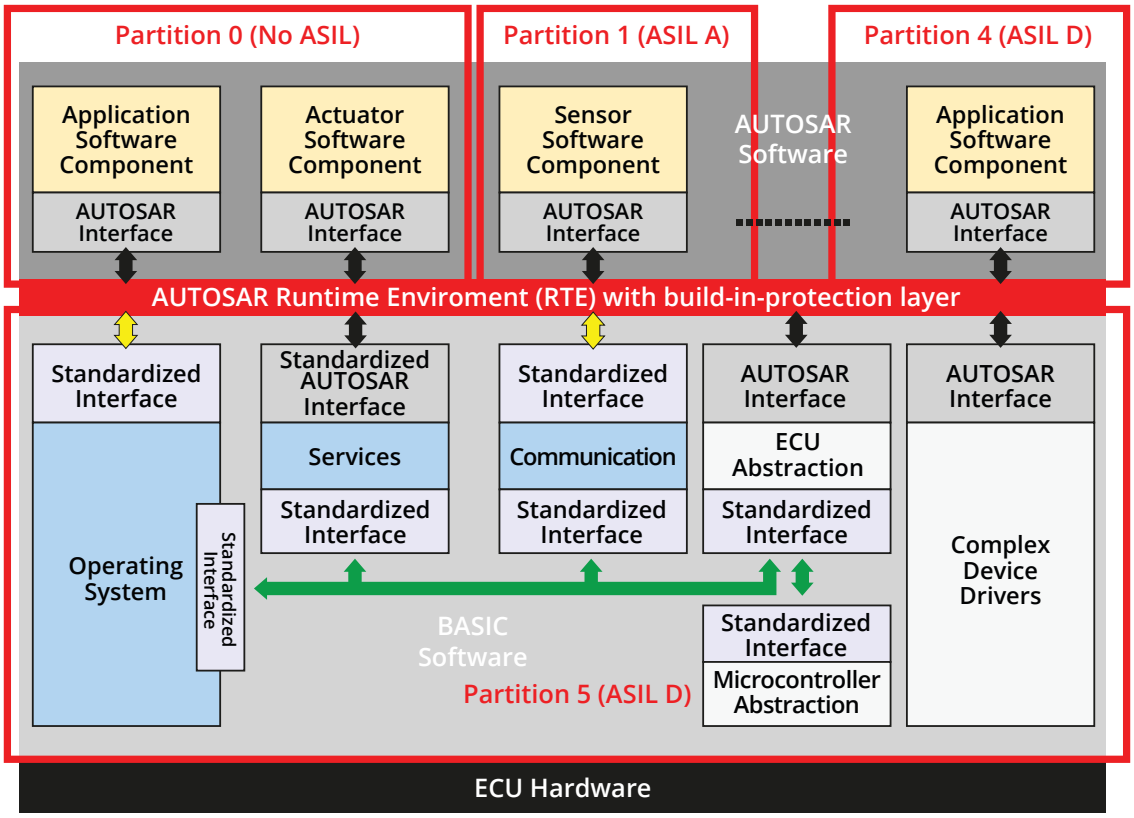

Illustration 9: AUTOSAR Partitioning Concept.

The following limitations apply to the AUTOSAR partitioning concept:

- Freedom from interference between software components of the same ASIL rating is not provided. The ISO26262 standard requires freedom from interference between software components of different ASIL ratings only.
- A violation caused by a single software component results in the shutdown or restart of the entire memory partition. As a result, all other correctly working software components located in this partition are affected as well.
- Memory Partitioning is not applicable for trusted OS-Applications. The execution of trusted/supervisor-mode memory partitions is not controlled by means of the Operating System and some MMU/MPU hardware implementations.
- Freedom from interference is not supported at the task-level. The implementation of task-level partitioning is not mandatory for AUTOSAR OS implementations.
- Depending on the architecture of the application software as well as the implementation of microcontroller hardware and the OS, there is a performance penalty associated with the use of memory partitioning. This penalty increases with the number of context switches which are performed per time unit.
- The current specification of the Basic Software does not specify memory partitioning for Basic Software Components.

CONCLUSION

It's clear that partitioning and freedom from interference is a system-level concern that is not implemented by hardware or software measures alone. Run-time environments such as AUTOSAR compliant RTOS or hypervisors provide features to mitigate the safety risks caused by memory interference. However, these solutions have limitations and error detection and recovery takes place at run-time, possibly after the system has been released for mass production.

There is a clear need for analysis tools that reveal memory interferences at software construction time. Such tools should be capable of analyzing the behavior of all software located on an ECU, performing a fine-grained analysis that also addresses interferences between different software elements with the same ASIL, and detecting intra-core and inter-core interferences.

TASKING Safety Checker is such tool. The underlying technology and the tool is described in part two of this white paper.