Document Title	Overview of Functional Safety Measures in AUTOSAR
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	664
Document Classification	Auxiliary

Document Status	Final
Part of AUTOSAR Release	4.2.2

Document Change History

Release	**Changed by**	**Change Description**
4.2.2	AUTOSAR Release Management	• New Chapter: „Hardware Diagnostics“ covers Core Test and RAM Test. • Minor corrections / clarifications / editorial changes.
4.2.1	AUTOSAR Release Management	• Initial Release

Disclaimer

This specification and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the specification.

The material contained in this specification is protected by copyright and other types of Intellectual Property Rights. The commercial exploitation of the material contained in this specification requires a license to such Intellectual Property Rights.

This specification may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the specification may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The AUTOSAR specifications have been developed for automotive applications only. They have neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Advice for users

AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

- AUTOSAR confidential -

1 Introduction

Functional safety is a system characteristic which is taken into account from the beginning, as it may influence system design decisions. Therefore AUTOSAR specifications include requirements related to functional safety.
Aspects such as complexity of the system design can be relevant for the achievement of functional safety in the automotive field.
Software is one parameter that can influence complexity on system level. New techniques and concepts for software development can be used in order to minimize complexity and therefore can ease the achievement of functional safety.

AUTOSAR supports the development of safety-related systems by offering safety measures and mechanisms. However AUTOSAR is not a complete safe solution.

The use of AUTOSAR does not imply ISO26262 compliance. It is still possible to build unsafe systems using the AUTOSAR safety measures and mechanisms.

1.1 Disclaimer

This explanatory document represents the functional safety measures and mechanisms of the latest release of AUTOSAR. Some of the described mechanisms and measures may be implemented differently or may not be available in previous releases. The user of this document shall always consult the applicable referenced documents.

1.2 Scope

The content of this document is structured into separate chapters as follows:

Functional Safety Mechanisms: This chapter contains AUTOSAR functional safety mechanisms related to freedom from interference between AUTOSAR SW-Cs.

- Memory: Partitioning mechanisms of AUTOSAR with the context of Application Software development and deployment.
- Timing: Temporal Program Flow Monitoring mechanisms using the Watchdog Manager and Timing Protection mechanisms using the Operating System.
- Execution: Logical Supervision mechanisms using the Watchdog Manager.
- Exchange of Information: Communication fault detection mechanisms using the End-2-End Library and Extensions.

Functional Safety Measures: This chapter contains topics related to the development of safety-relevant systems. The following items are covered:

- Functional Safety Measures of AUTOSAR, such as Traceability, Development Measures and the Evolution of the Standard.
- Functional Safety Measures not delivered by AUTOSAR.
- Safety Use Case: An exemplary safety related system using AUTOSAR based on the guided tour example Front Light Management.

- Safety Extensions: How safety requirements can be expressed within the AUTOSAR models and documents by means of the AUTOSAR meta-model.

Hardware Diagnostics: This chapter contains topics related to the premise, that the provided functionality of the microcontroller can be trusted. The following items are covered:
- Core Test.
- RAM Test.

1.3 Purpose

Information about AUTOSAR functional safety mechanisms and measures is currently distributed throughout the referenced documentation. Unless one knows how functional safety mechanisms are supported and where the necessary information is specifically located, it is difficult to evaluate how a safety-relevant system can be implemented using AUTOSAR efficiently.

This explanatory document summarizes the key points related to functional safety in AUTOSAR and explains how the functional safety mechanisms and measures can be used.

Note: This document supersedes the AUTOSAR document “Technical Safety Concept Status Report”, ID: 233.

1.4 Intended Audience

This document gives an overview of the functional safety measures and mechanisms of AUTOSAR and their implementation to those involved in the development of safety-relevant (ECU) systems. Therefore this document is intended for the users of AUTOSAR, including people involved in safety analysis.

2 Functional Safety Mechanisms

Modern ECUs contain highly modular embedded software, which can consist of both non-safety-related and safety-related software components, which perform functions with different ASIL ratings.
According to ISO 26262[1], if the embedded software consists of software components with different ASIL ratings, then either the entire software must be developed according to the highest ASIL, or freedom from interference shall be ensured for software components with a higher ASIL rating from elements with a lower ASIL rating.

Furthermore, the ISO 26262[2] standard provides examples for faults, which cause interference between software components. The faults are grouped as follows:

- Memory
- Timing
- Execution
- Exchange of Information

During the course of the following chapter, an overview of AUTOSAR functional safety mechanisms[3] is given. Those mechanisms assist with the prevention, detection and mitigation of hardware and software faults to ensure freedom from interference between software components.

Note: AUTOSAR functional safety mechanisms are used to support the development of safety-related systems. Therefore, functional safety mechanisms (software and hardware) are safety-related and must be developed and integrated accordingly.

[1] [ISO 26262-6 7.4.10]
[2] [ISO 26262-6, Annex D] Freedom from interference between software elements.
[3] In the context of this document, functional safety mechanisms are a concrete product part, such as memory protection. They are considered as specialization of functional safety measures, which also include process steps, like a review. This definition is in line with the definition given in ISO 26262 for these terms.

2.1 Memory Partitioning

A modular implementation of embedded systems that consists of both safety-related software components of different ASILs or of safety-related and non-safety-related software components is facilitated by AUTOSAR features that support freedom from interference between such software components.

Software Components which are developed according to a low ASIL rating may interfere by wrongfully accessing memory regions of software components with a higher ASIL rating. An execution of software components in separate memory regions or memory partitions supports the prevention of such memory access violations. Please see section 2.1.2.6 for further details.

The features described in this chapter are part of the OS and the RTE functionality, which are required to enable groups of SW-Cs to run in separate memory partitions, in order to provide freedom from interference between software components.

2.1.1 Fault Models

According to ISO 26262[4], the following memory-related effects of faults can be considered as a cause for interference between software components:

- Corruption of content.
- Read or write access to memory allocated to another software element.

The functional safety mechanism Memory Partitioning provides protection by means of restricting access to memory and memory-mapped hardware. Memory partitioning means that OS-Applications reside in different memory areas (partitions) that are protected from each other. In particular, code executing in one partition cannot modify memory of a different partition. Moreover, memory partitioning enables to protect read-only memory segments (e.g. code execution), as well as to protect memory-mapped hardware.

The memory partitioning and user/supervisor-modes related features address the following goal: Supporting freedom from interference between software components by means of memory partitioning (e.g. memory-related faults in SW-Cs do not propagate to other software modules and SW-Cs executed in user-mode have restricted access to CPU instructions like e.g. reconfiguration).

[4] [ISO 26262-6, Annex D] D.2.3 Memory

2.1.2 Description

Memory Partitioning is an extension of the RTE and the OS, which is described in the AUTOSAR Specification as „One Partition will be implemented using one OS-Application“[5] and "SW-Cs grouped in separate user-mode memory partitions"[6].
During the course of this chapter, this extension will be described as the relationship of Runnables, Tasks, Software Components and OS-Applications in the context of the AUTOSAR Methodology.

2.1.2.1 Application Software

In the AUTOSAR Architecture, Application Software is located above the RTE and consists of interconnected AUTOSAR Software Components, which atomically encapsulate parts of the Application Software functionality.

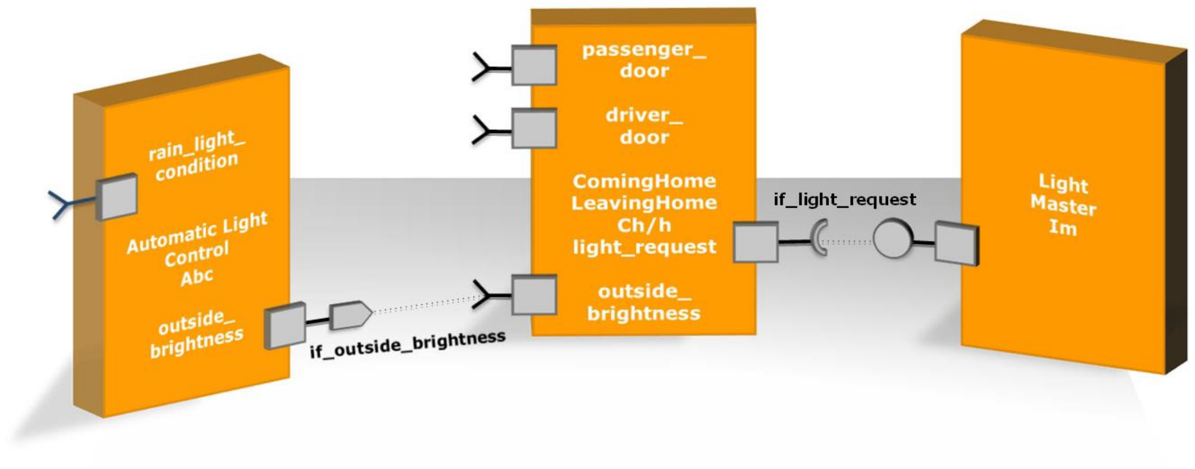

Figure 1: Application Software

AUTOSAR Software Components are hardware-independent, so they can be integrated onto any available ECU Hardware. To facilitate the inter- and intra-ECU information exchange, AUTOSAR Software Components communicate exclusively over the RTE.

AUTOSAR Software Components contain a number of Functions and Variables, which provide the internal functionality. The internal structure of an AUTOSAR Software Component, its Variables and Function Calls, is hidden from the public view via the header files. Only the external RTE calls are presented at the public interface.

[5] Specification of ECU Configuration, V3.5.0, R4.1 Rev 3, Page 154, ECUC_EcuC_00005
[6] Requirements on AUTOSAR Features, V1.2.1, R4.1 Rev 2, Chapter 4.11 Safety

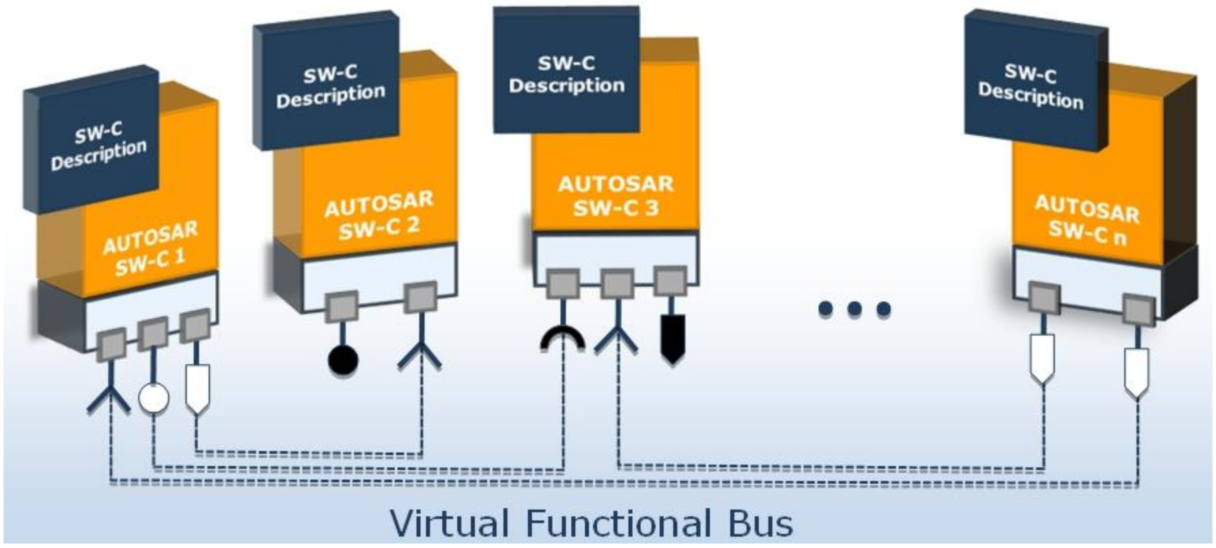

Figure 2: Software Components

AUTOSAR Software Components also contain functions, which must be invoked at runtime. Those C-functions are referred to as Runnables in AUTOSAR.
Runnables cannot be executed by themselves; they must be assigned to executable entities of the operating system. Such an assignment can be performed by inserting function calls of Runnables into OS-Task bodies.

Runnables are then executed cyclically and/or event-driven in the context of their caller OS-Task. The assignment of Runnables to Tasks is performed according to Figure 3 and Figure 4.

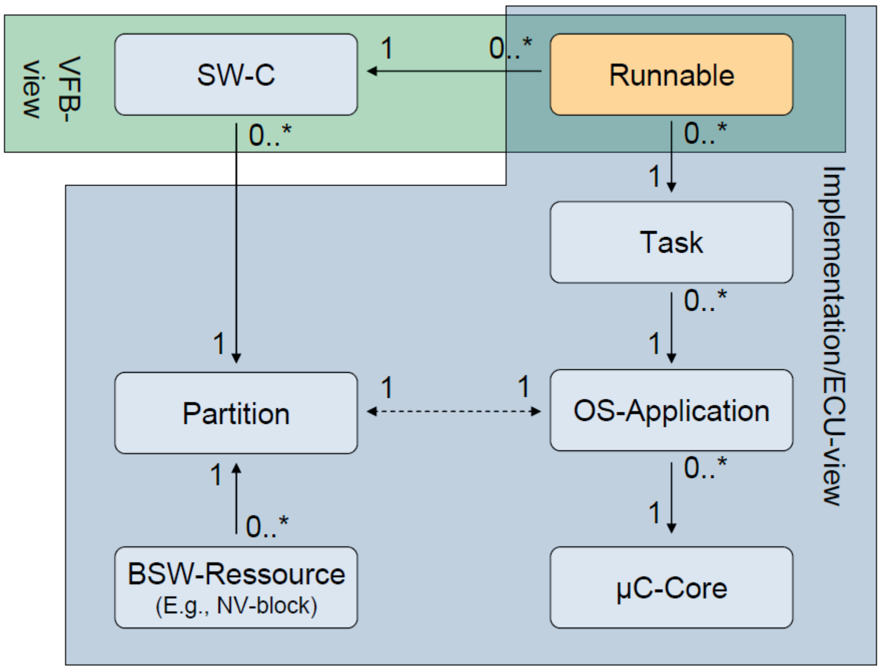

Figure 3: AUTOSAR Layered Software Architecture - Mapping of Runnables[7]

[7] Layered Software Architecture, V3.4.0, R4.1 Rev 3, Page 105

2.1.2.2 OS Applications

Figure 4 presents an interpretation of the relations from Figure 3. Runnables from AUTOSAR Software Components are assigned to OS-Tasks according to this diagram.

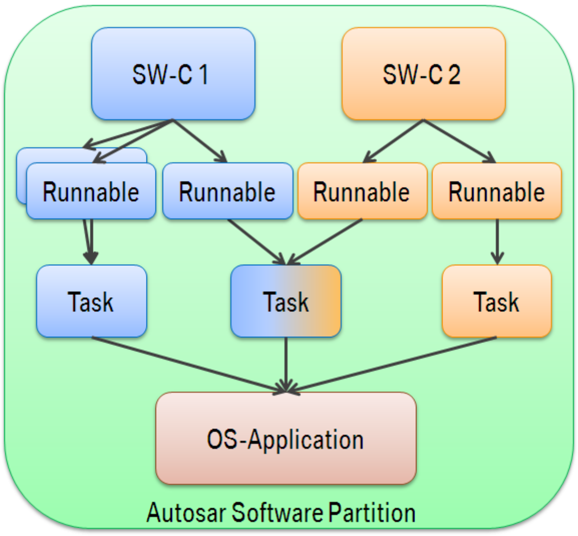

Figure 4: Mapping of Software Components to OS-Applications

AUTOSAR OS-Applications are collections of Operating System objects such as Tasks, ISRs, Schedule Tables, Counters and Alarms that form a cohesive functional unit. All objects which belong to the same OS-Application have access to each other.

The Operating System objects within an OS-Application may belong to different AUTOSAR Software Components. The RTE implements a memory area which is accessible by all members of the OS-Application without restrictions to facilitate communication between the SW-Cs efficiently.

There are two classes of OS-Applications:

1. “Trusted OS-Applications are allowed to run with monitoring or protection features disabled at runtime. They may have unrestricted access to memory and the Operating System module’s API. Trusted OS-Applications need not have their timing behavior enforced at runtime. They are allowed to run in privileged mode when supported by the processor.”
2. “Non-Trusted OS-Applications are not allowed to run with monitoring or protection features disabled at runtime. They have restricted access to memory, restricted access to the Operating System module’s API and have their timing behaviour enforced at runtime. They are not allowed to run in privileged mode when supported by the processor.”[8]

[8] Specification of Operating System, V5.3.0 R4.1 Rev 3, Chapter 7.6.1

2.1.2.3 Communication and Code Sharing

According to Figure 4 and Figure 3, an OS-Application can contain multiple AUTOSAR Software Components and associated Runnables. Runnables are only allowed to directly access variables and to perform function calls within their respective Software Component.

Internal Function Calls and Variables of a Software Component are not publically known by other Software Components, as their definitions are not presented by the header files of the external interface.
Therefore, a direct communication via variables and the execution of Code of other Software Components is not intended.

In Figure 5, this is illustrated by the example of code-sharing, which is only allowed within the Software Component and not between Software Components of one OS-Application. Communication with other Software Components shall be performed via the RTE. Runnable 4 may not execute functions belonging to SW-C 2.2.

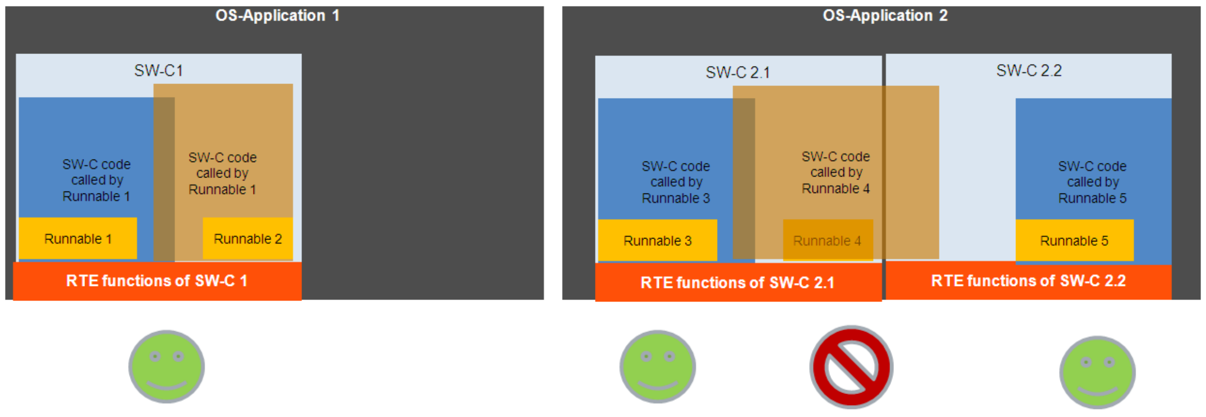
OS-Application 1
SW-C1
SW-C code called by Runnable 1
SW-C code called by Runnable 1
Runnable 1
Runnable 2
RTE functions of SW-C 1
OS-Application 2
SW-C 2.1
SW-C 2.2
SW-C code called by Runnable 3
SW-C code called by Runnable 4
SW-C code called by Runnable 5
Runnable 3
Runnable 4
Runnable 5
RTE functions of SW-C 2.1
RTE functions of SW-C 2.2

Figure 5: Code-Sharing within an OS-Application

2.1.2.4 Memory Partitioning within Application Software

Application Software in an AUTOSAR ECU can consist of safety-related and non-safety-related Software Components. Freedom from interference between Software Components with different ASIL ratings shall be ensured according to the requirements of ISO 26262[9].

The AUTOSAR Operating System provides freedom from interference for memory-related faults by placing OS-Applications into separate memory regions. This mechanism is called Memory Partitioning. OS-Applications are protected from each other, as code executing in the Memory Partition of one OS-Application cannot modify other memory regions. The corresponding requirements from the AUTOSAR OS specification are presented in Table 1.

Req. ID	***Requirement Text***
[SWS_Os_00207]	*The Operating System module shall prevent write access to the OS-Application's private data sections from other non-trusted OS-Applications.*
[SWS_Os_00355]	*The Operating System module shall prevent write access to all private stacks of Tasks/Category 2 ISRs of an OS-Application from other non-trusted OS-Applications.*
[SWS_Os_00356]	*The Operating System module shall prevent write access to all private data sections of a Task/Category 2 ISR of an OS-Application from other non-trusted OS-Applications.*

Table 1: AUTOSAR OS - Memory Partitioning for OS-Applications[10]

Application Software can consist of Software Components with different ASIL ratings. However, Software Components with different ASIL ratings should not be assigned to the same OS-Application. Memory Partitioning does not provide freedom from interference between Software Components which are assigned to the same OS-Application. The Operating System only prevents other OS-Applications from performing improper accesses. A faulty Software Component would not be prevented from modifying memory areas of other Software Components within the same OS-Application.

Note: Please consult the subsequent section for details on Task-level partitioning.

[9] [9] [ISO 26262-6 7.4.10]
[10] Specification of Operating System, V5.3.0 R4.1 Rev 3

2.1.2.5 Memory Partitioning within Software Components

A Mixed-ASIL Software Component could consist of Runnables with different ASIL ratings and therefore requires an execution environment which supports freedom from interference between those Runnables. An execution of different Runnables of one Software Component in different Memory Partitions is not possible due to the following:

Memory Partitioning is performed at the level of OS-Applications. According to Figure 3 and Figure 4 however, a Software Component can only be assigned to one OS-Application and therefore has only one Memory Partition. Also, Runnables of a Software Component can only be called by the Tasks of one OS-Application.

As shown in Figure 6, Runnables of a Software Component cannot be distributed to Tasks of multiple OS-Applications.

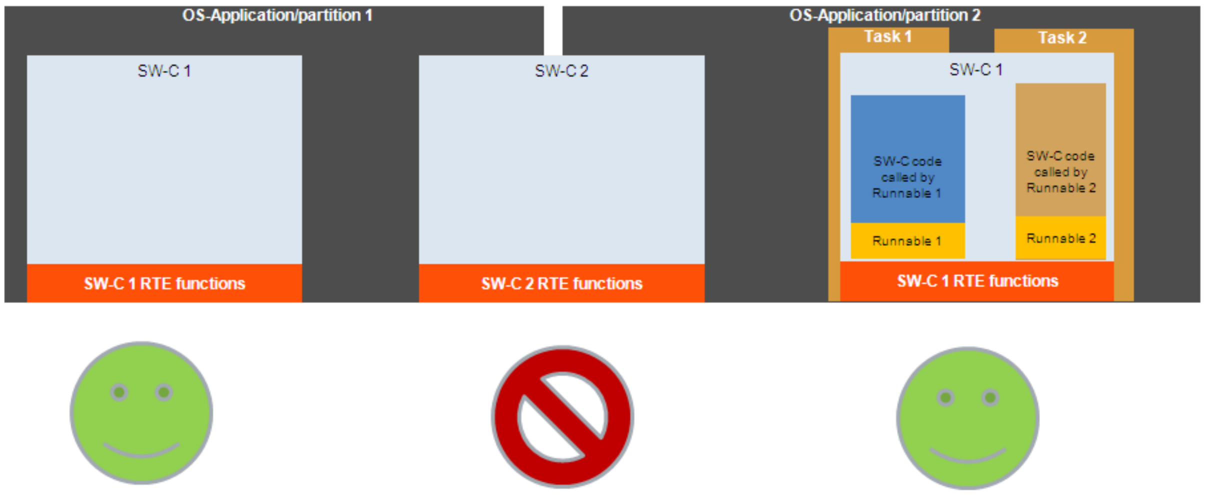
OS-Application/partition 1
OS-Application/partition 2
Task 1
Task 2
SW-C 1
SW-C 2
SW-C 1
SW-C code called by Runnable 1
SW-C code called by Runnable 2
Runnable 1
Runnable 2
SW-C 1 RTE functions
SW-C 2 RTE functions
SW-C 1 RTE functions

Figure 6: SWCs vs. Partitions

Memory Partitioning cannot be used to separate Runnables within the same SW-C. If it is necessary to have a Software Component comprise Runnables with different ASIL-ratings and an independent execution with freedom from interference is required for those Runnables, then memory partitioning at OS-Application level is not sufficient, memory partitioning has to be performed at Task-level. This approach is shown in Figure 7.

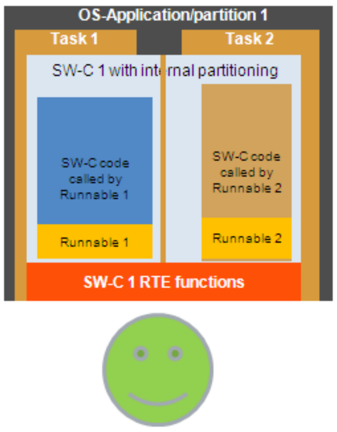
OS-Application/partition 1
Task 1
Task 2
SW-C 1 with internal partitioning
SW-C code called by Runnable 1
Runnable 1
SW-C code called by Runnable 2
Runnable 2
SW-C 1 RTE functions

Figure 7: Task Partitioning

Requirements related to Memory Partitioning at Task-level are listed in the AUTOSAR OS specification in Table 2. The use of the weak word “may” shows that an implementation of Task-level partitioning is optional for the AUTOSAR OS. Therefore, not every AUTOSAR OS implementation may support Task-level Memory Partitioning.

Req. ID	***Requirement Text***
[SWS_Os_00208]	*The Operating System module may prevent write access to the private stack of Tasks/Category 2 ISRs of a non-trusted application from all other Tasks/ISRs in the same OS-Application.*
[SWS_Os_00195]	*The Operating System module may prevent write access to the private data sections of a Task/Category 2 ISR of a non-trusted application from all other Tasks/ISRs in the same OS-Application.*

Table 2: AUTOSAR OS Requirements – Memory Partitioning for Tasks[11]

[11] Specification of Operating System, V5.3.0 R4.1 Rev 3

2.1.2.6 Implementation of Memory Partitioning

A broad variety of technical safety concepts on the system- and software level can be implemented using the mechanism Memory Partitioning.

Figure 8 shows a possible implementation whereas all Basic Software Modules are executed in one trusted/supervisor-mode[12] memory partition (highlighted in red in Figure 8). Some SW-Cs are logically grouped and put in separate non- trusted/user-mode memory partitions (highlighted in green). Selected SW-Cs belong to the same trusted/supervisor-mode memory partition as the Basic Software Modules (see fourth SW-C in Figure 8 highlighted in red). There may be several non-trusted/user-mode[13] partitions, each containing one or more SW-Cs.

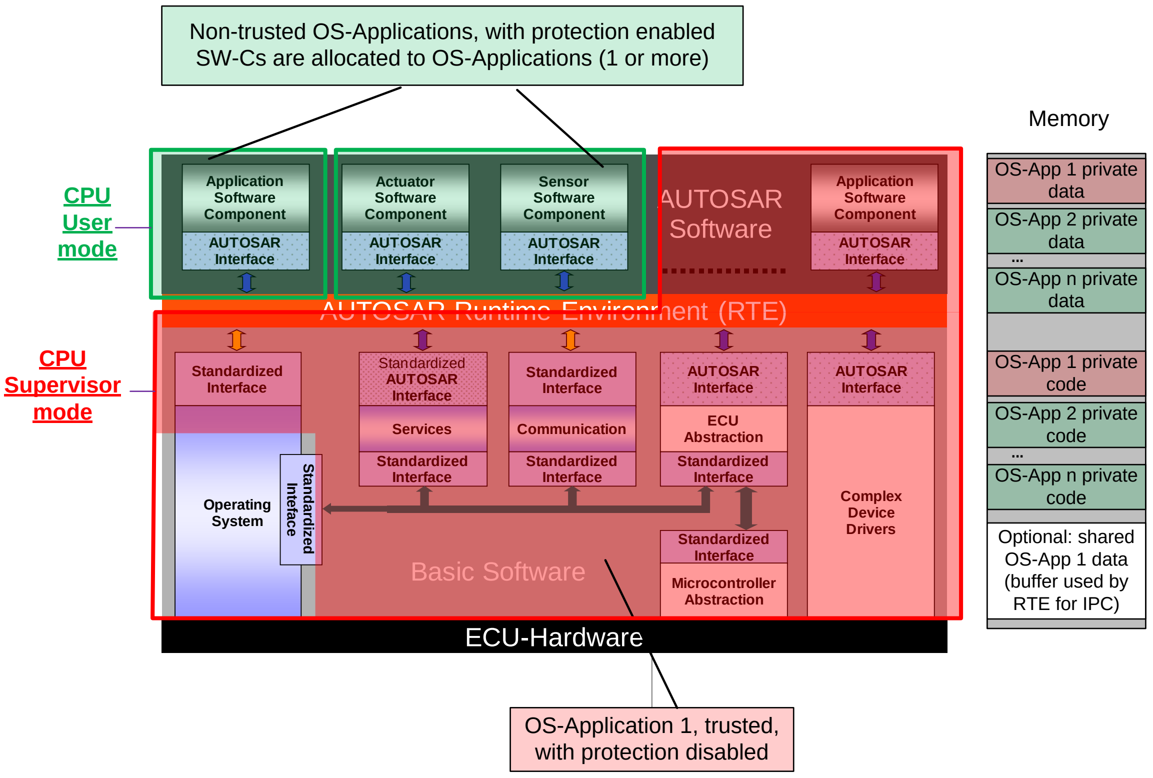

Figure 8: Memory partitioning and modes[14]

The execution of SW-Cs in non-trusted/user-mode memory partitions is restricted from modifying other memory regions, whereas the execution of SW-Cs of trusted/supervisor-mode memory partitions is not restricted.

Modern microcontrollers for safety relevant applications support memory partitioning via dedicated hardware, a Memory Protection Unit (MPU).

[12] Supervisor Mode, Privileged Mode and Elevated Mode are synonyms for the elevated CPU mode. Trusted Mode is a mode of the Software, which is executed under the elevated CPU Mode.
[13] User Mode and Non-Privileged Mode are synonyms for a non-elevated CPU mode. Non-Trusted Mode is a mode of the Software, which is executed under the non-elevated CPU Mode.
[14] Technical Safety Concept Status Report, V1.2.0, R4.1 Rev 1, Chapter 1.1.6 Memory Partitioning and User/Supervisor-Modes Related Features

Note: It is assumed that memory partitioning will be implemented on a microcontroller which has an MPU or similar hardware features[15].

With a typical MPU implementation, access to multiple sections of the microcontroller address space can be allowed for non-trusted applications. Access control is defined as a combination of Read, Write and Execute accesses. The configuration of the MPU is only permissible in supervisor mode.

Note: In some microcontroller implementations the MPU is integrated within the Processor Core. Therefore that MPU only controls accesses of the associated Core. Other Bus Masters, such as DMA controllers and additional Cores, are not controlled by this particular MPU instance.

The following table and use cases illustrate a set of possible scenarios when the configuration of the memory protection unit is derived from system requirements.
Note: This table may be incomplete with respect to the features of the specific hardware devices in use.

Address Space	***Rationale***	***Read***	***Write***	***Execute***
Flash Memory	Read, Execute and Write accesses do not modify flash memory contents. Flash memory must be erased and enabled for writing by a different mechanism first. Note: The following implications arise from the Security point of view: Reading and execution of foreign code may be used to obtain information which is otherwise not intended for the software.	O	O	O
RAM	Write access to RAM may produce memory corruptions, thereby affecting the behavior of the software.	O	X	O
Peripheral	Side effects are possible even when reading from peripheral address space. E.g.: Acknowledgement of an Interrupt is performed via a read access to the Interrupt Controller, Read access to peripherals may cause I/O errors.	X	X	X

Table 3: Configuration scenarios for Memory Protection

Legend:
X – Protection is needed
O – Protection is optional
Note: Side effects from performance point of view may arise due to Bus Contention, arbitration at interfaces, etc.

[15] [ISO26262-6 7.4.11 b)]

Use Case 1: Software Components in the same Partition.

- Software Components in the same partition have access to each other´s RAM regions, and therefore can corrupt each other´s memory contents.
- Software components do not have access to peripheral devices by definition, as they shall be not aware of the underlying microcontroller architecture. An unsafe system can be created when a software component is given direct access to peripheral devices.

Use Case 2: Software Components in different Partitions.

- Software Components in different partitions **do not** have access to each other´s RAM regions, and therefore **cannot** corrupt each other´s memory contents.
- Software components do not have access to peripheral devices by definition, as they shall be not aware of the underlying microcontroller architecture. A potentially unsafe system can be created when a software component is given direct access to peripheral devices.

Use Case 3: MCAL Drivers

- MCAL Drivers are a collection of functions, such as Read/Write/Initialize. They must be executed by another entity, such as the BSW or a CDD. Please see Figure 8 for details.
- MCAL Drivers need a Read/Write access to the peripheral space of the respective peripheral hardware module. Depending on the hardware architecture, supervisor mode of the processor may be additionally required.

2.1.3 Detection and Reaction

The functional safety mechanism Memory Partitioning provides protection by means of restricting access to memory and memory-mapped hardware. Code executing in one partition cannot modify memory of a different partition. Memory partitioning enables to protect read-only memory segments, as well as to protect memory-mapped hardware. Moreover, Software Components which are executed in user-mode have restricted access to CPU instructions like e.g. reconfiguration.

The mechanism Memory Partitioning can be implemented with the support of microcontroller hardware such as Memory Protection Unit or Memory Management Unit. The microcontroller hardware must be configured appropriately by the Operating System to facilitate detection and prevention of incorrect memory accesses. The execution of Software Components which are executed in non-trusted/user-mode memory partitions is then monitored.

In case of a memory access violation or a CPU instruction violation in a non-trusted/user-mode partition, the faulty access is blocked and an exception is raised by the microcontroller hardware. The OS and the RTE handle the erroneous software partition by performing either a partition shut down or restart of all SW-Cs of this partition.

Note: The actual reaction of the Operating System can be configured though the Protection Hook implementation. Please consult the OS SWS[16] document for further details.

Note: The AUTOSAR Document "Explanation of Error Handling on Application Level"[17] provides additional information on error handling. Within the document it is explained how error handling can be performed and where the required data (e.g. substitute values) can be obtained from. Furthermore the document provides a detailed explanation (user´s manual) on how OS-Application/Partition termination and restart in AUTOSAR is performed.

[16] Specification of Operating System, V5.3.0 R4.1 Rev 3
[17] Explanation of Error Handling on Application Level, R4.2 Rev 1, Chapter 8, Chapter 10

2.1.4 Limitations

1. Memory Partitioning of SW-Cs with the same ASIL rating.
The ISO26262[18] standard requires freedom from interference between Software Components of different ASIL ratings. However, freedom from interference between Software Components of the same ASIL rating is not required by the standard.

OS-Applications which consist of a large number of Software Components are allowed. In case a single Software Component causes a violation which results in shutdown or restart of the entire memory partition, all other correctly working SW-Cs of this memory partition are affected as well.

2. 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.

3. Memory Partitioning not supported on task-level.
The implementation of task-level partitioning is not mandatory for AUTOSAR OS implementations. Freedom from Interference within the OS-Application may be therefore not supported.

4. Performance penalty due to Memory Partitioning.
Depending on the architecture of the Application Software and 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.

5. No Basic Software Partitioning.
The current specification of the Basic Software does not specify memory partitioning for Basic Software Components with different ASIL ratings from different suppliers.

[18] [ISO 26262-9 Clause 6]

2.1.5 References to AUTOSAR Documents

Source: Requirements on AUTOSAR Features, V1.2.1, R4.1 Rev 2

[RS_BRF_01232] AUTOSAR OS shall support isolation and protection of application software

⌈

Type:	Valid
Description:	AUTOSAR OS shall support to organize all objects handled by the OS such that they can be assigned to different entities (OSApplications) and that access between OSApplications is restricted. This includes usage of hardware memory protection. Note: Assignment of Software Components to OSApplications needs to be done outside the OS
Rationale:	This is a pre-requirement to install protection mechanisms for higher level BSW and Software Components
Use Case:	Usage of memory protection properties of microcontrollers to catch erroneous write access of software components
Dependencies:	--
Supporting Material:	--

⌋ (RS_Main_00010, RS_Main_00100)

[RS_BRF_02048] AUTOSAR shall support usage of hardware memory protection features to enhance safety

⌈

Type:	Valid
Description:	If adequate memory protection mechanisms are supported by hardware, AUTOSAR shall support the usage of these hardware mechanisms in such a way that memory used by SW-Cs and BSW modules can be protected from illegal or erroneous access
Rationale:	Only if it can be shown that different groups of software components do not interfere, the groups of software components can be evaluated separately with respect to their safety requirements
Use Case:	Combine software components of different ASIL level on the same ECU
Dependencies:	--
Supporting Material:	ISO 26262-6:2011, Annex D (Freedom from interference between software elements)

⌋ (RS_Main_00010)

[RS_BRF_01248] AUTOSAR OS shall support to terminate and restart OSApplications

⌈

Type:	Valid
Description:	AUTOSAR OS shall support to terminate and – if wanted - restart OSApplications
Rationale:	If an OSApplication encounters an error, the error strategy of the ECU needs to decide if this OSApplication can be permitted to continue working, and eventually terminate or terminate and restart the OSApplication. The OS needs to offer the necessary functionality
Use Case:	Memory protection error in an OSApplication which cannot be salvaged without terminating the OSApplication
Dependencies:	--
Supporting Material:	--

⌋ (RS_Main_00010, RS_Main_00100)

2.1.6 References to ISO26262

The following references to the ISO26262 standard are related to the aspects of freedom from interference for software components with different ASIL ratings.
Additionally, concepts related to software partitioning and memory-related faults are covered.

ID	***ISO26262 Reference***
01	Part 6: [7.4.11]
02	Part 6: [7.4.12]
03	Part 6: [D.2.1]
04	Part 6: [D.2.3]
05	Part 9: [6.2]
06	Part 9: [6.4.4]
07	Part 9: [6.4.5]

Table 4: ISO26262 Memory Partitioning References

2.2 Timing Monitoring

Timing is an important property of embedded systems. Safe behavior requires that the systems actions and reactions are performed within the right time.
The right time can be described in terms of a set of timing constraints that have to be satisfied. However, an AUTOSAR software component cannot ensure proper timing by itself. It depends on proper support by the AUTOSAR runtime environment and the basic software. During integration the timing constraints of the AUTOSAR software components need to be ensured.

2.2.1 Fault Models

According to ISO 26262[19], the following Timing- and Execution-related faults can be considered as a cause for interference between software components:

- Blocking of execution
- Deadlocks
- Livelocks
- Incorrect allocation of execution time
- Incorrect synchronization between software elements

Timing protection and monitoring can be described as monitoring of the following properties: Monitoring that tasks are dispatched at the specified time, meet their execution time budgets, and do not monopolize OS resources.
To guarantee that safety-related functions will respect their timing constraints, tasks monopolizing the CPU (such as heavy CPU load, many interrupt requests) shall be detected and handled.

[19] [ISO 26262-6, Annex D] D.2.2 Timing and execution

2.2.2 Description

The following timing monitoring mechanisms are provided by AUTOSAR:
1. Timing Protection mechanisms using the Operating System.
2. Temporal Program Flow Monitoring using the Watchdog Manager.

This chapter will explain the applicability of the Watchdog Manager for implementing timing monitoring of Application Software. Temporal Program Flow Monitoring consists of the mechanisms Deadline Supervision and Alive Supervision, which will be discussed thereafter.
The Watchdog Manager also provides a mechanism called Logical Supervision, which can be combined with Deadline Supervision to provide a high diagnostic coverage. This topic is discussed in Chapter 2.3.
Also, an overview of the Timing Protection mechanisms of AUTOSAR OS will be given.

2.2.2.1 Supervised Entities

The Watchdog Manager supervises the execution of Application Software in an AUTOSAR ECU. The logical units of supervision are called Supervised Entities. There is no fixed relationship between Supervised Entities and the architectural building blocks in AUTOSAR. Typically a Supervised Entity may represent one SW-Cs or a Runnable within an SW-C, a BSW module or CDD depending on the choice of the developer.
Important places in a Supervised Entity are defined as Checkpoints. The code of Supervised Entities is interlaced with function calls of the Watchdog Manager. Those calls are used to report to the Watchdog Manager that a Checkpoint is reached.

2.2.2.2 Watchdog Manager

The Watchdog Manager is a basic software module of the AUTOSAR Architecture.
The Watchdog Manager links the triggering of the Watchdog Hardware[20] to the supervision of software execution. When a violation of the configured temporal and/or logical constraints on program execution is detected, a number of configurable actions to recover from this failure will be taken.
The Watchdog Manager provides the following mechanisms for Temporal Program Flow Monitoring:

Alive Supervision: Periodic Supervised Entities have constraints on the frequency with which they are executed. By means of Alive Supervision, Watchdog Manager checks periodically if the Checkpoints of a Supervised Entity have been reached within the given limits. This means that Watchdog Manager checks if a Supervised Entity is run not too frequently or not too rarely.

[20] See Layered Software Architecture, V3.4.0, R4.1 Rev 3, Page 42, Page 82.

Alive Supervision is performed using a single Checkpoint without transitions. The supervised Entity must cyclically call the Checkpoint to signal its timely operation. The Watchdog Manager is executed periodically by the Operating System to verify the Checkpoint parameters.
A Supervised Entity can also be monitored by multiple instances of Alive Supervision, therefore containing an independent checkpoint per Alive Supervision. Please see Figure 9.

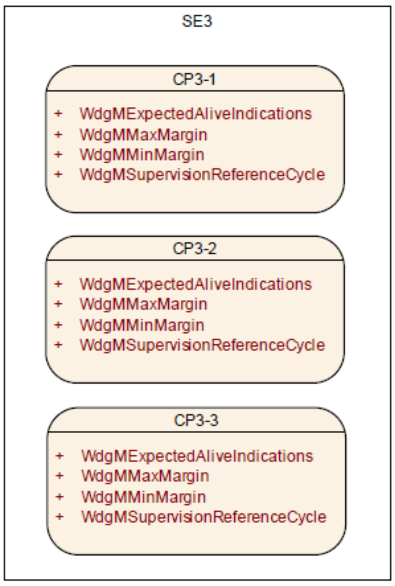
SE3
CP3-1
+ WdgMExpectedAliveIndications
+ WdgMMaxMargin
+ WdgMMinMargin
+ WdgMSupervisionReferenceCycle
CP3-2
+ WdgMExpectedAliveIndications
+ WdgMMaxMargin
+ WdgMMinMargin
+ WdgMSupervisionReferenceCycle
CP3-3
+ WdgMExpectedAliveIndications
+ WdgMMaxMargin
+ WdgMMinMargin
+ WdgMSupervisionReferenceCycle

Figure 9: Alive Supervision with independent Checkpoints[21]

Deadline Supervision: Aperiodic or episodic Supervised Entities have individual constraints on the timing between two Checkpoints. By means of Deadline Supervision, the Watchdog Manager checks the timing of transitions between two Checkpoints of a Supervised Entity. This means that the Watchdog Manager checks if some steps in a Supervised Entity take a time that is within the configured minimum and maximum. Please see Figure 10.

If the second Checkpoint is never reached, then Deadline Supervision will fail to detect this issue. This issue appears because the timing checks are performed by the Watchdog Manager after the second Checkpoint is called.

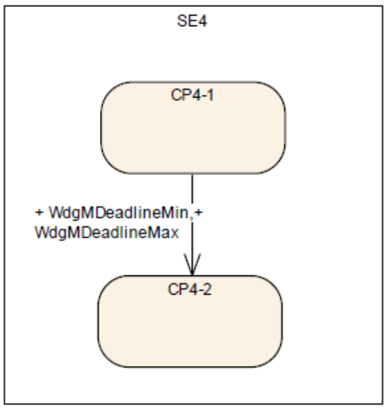

Figure 10: Deadline Supervision[22]

[21] Specification of Watchdog Manager, V2.5.0, R4.1 Rev 3, Page 43, Chapter 7.1.5 Alive Supervision Functions

2.2.2.3 Timing Protection of the Operating System

According to the AUTOSAR OS Specification, a timing fault in a real-time system occurs when a task or interrupt misses its deadline at runtime.[23]
The AUTOSAR OS does not offer deadline supervision for timing protection. Deadline supervision is insufficient to correctly identify the Task or Interrupt causing a timing fault in an AUTOSAR system. A deadline violation may be caused by unrelated Tasks or Interrupts interfering with the execution. Please consult the AUTOSAR OS Specification[23] for further details.
Whether a task or interrupt meets its deadline in a fixed priority preemptive operating system like AUTOSAR OS is determined by the following factors:

- The execution time of Task/Interrupt in the system.
- The blocking time that Task/Interrupt suffers from lower priority Tasks/Interrupts locking shared resources or disabling interrupts.
- The inter-arrival rate of Task/Interrupt in the system.

For safe and accurate timing protection it is necessary for the operating system to control these factors at runtime to ensure that Tasks/Interrupts can meet their respective deadlines. The AUTOSAR OS provides the following timing protection mechanisms:

1. Execution Time Protection. An upper bound for execution time of Tasks or Cat2[24] Interrupts, the so called Execution Budget, is monitored via the OS to prevent timing errors.
2. Locking Time Protection. An upper bound for blocking of resources, locking and suspending of interrupts, the so called Lock Budget, is monitored by the OS.
3. Inter-Arrival Time Protection. A lower bound between tasks being activated or Cat 2 Interrupts arriving, a so called Time Frame, is monitored via the OS to prevent timing errors.

Note: Execution time enforcement requires hardware support, e.g. a timing enforcement interrupt. If an interrupt is used to implement the time enforcement, the priority of this interrupt shall be high enough to “interrupt” the supervised tasks or interrupts.

[22] Specification of Watchdog Manager, V2.5.0, R4.1 Rev 3, Page 61, Chapter 7.3 Watchdog Handling
[23] Specification of Operating System, V5.3.0 R4.1 Rev 3, Chapter 7.7.2
[24] Category 2 Interrupts are managed by the Operating System. Category 1 Interrupts are executed outside of the Operating System and therefore cannot be monitored.

2.2.3 Detection and Reaction

The Watchdog Manager provides three mechanisms for Temporal and Logical Program Flow Monitoring: Deadline Supervision, Alive Supervision and Logical Supervision.
The supervision mechanisms are configured statically. For the monitoring of a Supervised Entity, more than one supervision mechanism can be employed.

Based on the results from each of enabled mechanisms, the status of the Supervised Entity (called Local Status) is computed. When the status of each Supervised Entity is determined, then based on each Local Supervision Status, the status of the whole MCU is determined (called Global Supervision Status).

Depending on the Local Supervision Status of each Supervised Entity and on the Global Supervision Status, the Watchdog Manager initiates a number of mechanisms to recover from supervision failures. These range from local error recovery within the Supervised Entity to a global reset of the ECU.

The following error recovery mechanisms can be employed by the Watchdog Manager:

1. Error Handling in the Supervised Entity

In case the Supervised Entity is an SW-C or a CDD, then the Watchdog Manager may inform the Supervised Entity about supervision failures via the RTE Mode mechanism. The Supervised Entity may then take its actions to recover from that failure.
The Watchdog Manager may register an entry with the Diagnostic Event Manager (DEM) when it detects a supervision failure. A Supervised Entity may take recovery actions based on that error entry.

2. Partition Shutdown

If the Watchdog Manager module detects a supervision failure in a *Supervised Entity* which is located in a non-trusted partition, the Watchdog Manager module may request a partition shutdown by calling the BswM.

3. Reset by Hardware Watchdog

The Watchdog Manager indicates to the Watchdog Interface when Watchdog Interface shall no longer trigger the hardware watchdog. After the timeout of the hardware watchdog, the hardware watchdog resets the ECU or the MCU. This leads to a re-initialization of the ECU and/or MCU hardware and the complete re-initialization of software.

4. Immediate MCU Reset

In case an immediate, global reaction to the supervision failure is necessary, the Watchdog Manager may directly cause an MCU reset. This will lead to a re-initialization of the MCU hardware and the complete software. Usually, a MCU reset will not re-initialize the rest of the ECU hardware.

Note: The AUTOSAR Document “Explanation of Error Handling on Application Level”[25] provides additional information on error handling. Within the document it is explained how error handling can be performed and where the required data (e.g. substitute values) can be obtained from. Furthermore the document provides a detailed explanation (user´s manual) on how OS-Application/Partition termination and restart in AUTOSAR is performed.

2.2.4 Limitations

1. The granularity of Checkpoints is not fixed by the Watchdog Manager. Few coarse-grained Checkpoints limit the detection abilities of the Watchdog Manager. For example, if an application SW-C only has one Checkpoint that indicates that a cyclic Runnable has been started, then the Watchdog Manager is only capable of detecting that this Runnable is re-started and check the timing constraints. In contrast, if that SW-C has Checkpoints at each block and branch in the Runnable the Watchdog Manager may also detect failures in the control flow of that SW-C. High granularity of Checkpoints causes a complex and large configuration of the Watchdog Manager.
2. The Deadline Supervision has a weakness: it only detects the delays (when the End Checkpoint is reported), but it does not detect the timeouts (when the End Checkpoint is not reported at all).
3. The nesting of Deadline Supervision (i.e. start 1, start 2, end 2, end 1) is not supported.
4. The Alive Supervision function with more than one checkpoint per Supervised Entity is not consistently specified within the Specification of Watchdog Manager document. For now it is recommended to support only one alive supervision checkpoint per Supervision Entity.
5. In order to shutdown or restart (as error reaction) a partition containing Supervised Entities, the integrator code (OS Application's restart task) must deactivate (or deactivate + activate) all Supervised Entities of the involved partition, by calling available functions of Watchdog Manager. This is a bit complex, in future releases of the Specification of Watchdog Manager document it is considered to add a new function of Watchdog Manager for this.
6. Libraries cannot call BSWs, so libraries cannot be supervised by Watchdog Manager. Deadline Supervision could be used however by placing checkpoints before and after a library call in the module´s code to supervise libraries.
7. It is not standardized how BSW modules are identified with Supervised Entity IDs.

[25] Explanation of Error Handling on Application Level, R4.2 Rev 1, Chapter 8, Chapter 10

2.2.5 References to AUTOSAR Documents

Source: Requirements on AUTOSAR Features, V1.2.1, R4.1 Rev 2

[RS_BRF_00131] AUTOSAR shall support program flow monitoring

⌈

Type:	Valid
Description:	AUTOSAR shall support logical and temporal program flow monitoring to detect if program flow control is violated. AUTOSAR shall offer support for ensuring that the program flow monitoring mechanisms are working properly
Rationale:	Using flow control to detect if a software components runs wild is an established safety feature Using program flow control to detect if a runnable (or a sequence of runnables) is executed out of order or not at all is a well established safety feature
Use Case:	To detect a defective program sequence. A defective program sequence exists, if the individual elements of a program (for example, software modules, subprograms or commands) are processed in the wrong sequence or period of time, or if the clock of the processor is faulty
Dependencies:	--
Supporting Material:	ISO 26262-5:2011 Annex D, ISO 26262-6:2011

⌋ (RS_Main_00010)

[RS_BRF_02056] AUTOSAR OS shall support timing protection

⌈

Type:	Valid
Description:	If configured, AUTOSAR OS shall support to supervise runtime of tasks and interrupts, together with frequency of task and interrupt activation, to detect and react if a task or an interrupt consume more runtime than configured
Rationale:	Systems are usually evaluated based on assumptions concerning runtime and frequency of tasks and interrupts. The violation of these assumptions may lead to the violation of the safety goals
Use Case:	Stop application parts which violate runtime constraints
Dependencies:	--
Supporting Material:	--

⌋ (RS_Main_00010)

[RS_BRF_01224] AUTOSAR OS shall support timing protection
⌈

Type:	Valid
Description:	AUTOSAR OS shall offer functionality to limit runtime and activation frequency of tasks and interrupts
Rationale:	This is a pre-requirement to catch problems with interrupt lines (babbling idiot) and certain programming bugs
Use Case:	Disable an interrupt line if this interrupt line fires too often
Dependencies:	--
Supporting Material:	--

⌋ (RS_Main_00100)

2.2.6 References to ISO26262

The following references to the ISO26262 standard are related to the aspects of freedom from interference for software components with different ASIL ratings. Concepts related to timing supervision are covered.

ID	***ISO26262 Reference***
03	Part 6: [D.2.1]
08	Part 6: [D.2.2]
09	Part 6: [7.4.14] Table 4: 1d

Table 5: ISO26262 Timing Monitoring References

2.3 Logical Supervision

Logical Supervision is a technique for checking the correct execution of software and focuses on control flow errors.
Control flow errors cause a divergence from the valid (i.e. coded/compiled) program sequence during the error-free execution of the application. An incorrect control flow occurs if one or more program instructions are processed either in the incorrect sequence or are not even processed at all. Control flow errors can for example lead to data inconsistencies, data corruption, or other software failures.

2.3.1 Fault Models

According to ISO 26262[26], the following Timing- and Execution-related faults can be considered as a cause for interference between software components:

- Blocking of execution
- Deadlocks
- Livelocks
- Incorrect allocation of execution time
- Incorrect synchronization between software elements

Logical and temporal monitoring of program sequences is used in the automotive industry and mentioned e.g. in ISO 26262 as a measure to detect failures of the processing units (i.e. CPU, microcontroller) and as measure for the detection of failures of the HW clock.

Faults in execution of program sequences (i.e. invalid execution of program sequences) can lead to data corruption, process crashes, or fail-silence violations.
Logical monitoring of program sequences is required/recommended/proposed by ISO 26262, IEC 61508, MISRA.

2.3.2 Description

Logical Supervision of the execution sequence of a program enables the detection of errors that cause a divergence from the valid program sequence during the error-free execution of the application. An incorrect program flow occurs if one or more program instructions are processed either in an incorrect sequence or not even processed at all.

The Watchdog Manager supervises the execution of Application Software in an AUTOSAR ECU. The logical units of supervision are called Supervised Entities. There is no fixed relationship between Supervised Entities and the architectural building blocks in AUTOSAR. Typically a Supervised Entity may represent one SW-Cs or a Runnable within an SW-C, a BSW module or CDD depending on the choice of the developer.

[26] [ISO 26262-6, Annex D] D.2.2 Timing and execution

Places relevant for logical supervision in a Supervised Entity are defined as Checkpoints. The code of Supervised Entities is interlaced with function calls of the Watchdog Manager. Those calls are used to report to the Watchdog Manager that a Checkpoint is reached.

Each Supervised Entity has one or more Checkpoints. The Checkpoints and Transitions between the Checkpoints of a Supervised Entity form a Graph.

A Graph may have one or more[27] initial Checkpoints and one or more final Checkpoints. Any sequence of starting with any initial checkpoint and finishing with any final checkpoint is correct, assuming that the checkpoints belong to the same Graph.

A graph within a Supervised Entity is called an Internal Graph. Checkpoints from different Supervised Entities can be connected by External Transitions, forming an External Graph.

Figure 11 shows a Graph representation of a While-Loop, which consists of Checkpoints and Transitions.

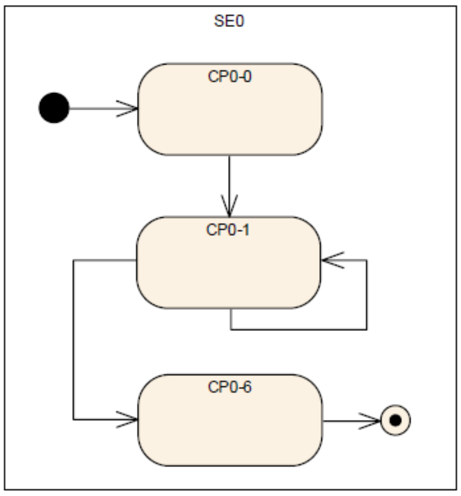

Figure 11: Abstract Control Flow Graph of a While-Loop[28]

At runtime, the Watchdog Manager verifies if the supervised Entities are executed according to the configured Graphs. This is called Logical Supervision.
Also, the Watchdog Manager can verify the timing of Checkpoints and Transitions within a Graph.
The timing of Transitions between Checkpoints can be verified via Deadline Supervision, whereas Logical Monitoring verifies the correct order of the Checkpoints. The details of Timing Monitoring mechanisms are described in Chapter 2.2.

[27] Internal graphs can have only one initial Checkpoint. External graphs can have multiple initial Checkpoints.
[28] Specification of Watchdog Manager, V2.5.0, R4.1 Rev 3, Chapter 7.1.7 Logical Supervision

2.3.3 Detection and Reaction

During design phase the valid program sequences are identified and modeled. During runtime the Watchdog Manager uses this model to supervise or monitor the proper execution of program sequences.

The Watchdog Manager provides three mechanisms for Temporal and Logical Program Flow Monitoring: Deadline Supervision, Alive Supervision and Logical Supervision.
The supervision mechanisms are configured statically. For the monitoring of a Supervised Entity, more than one supervision mechanism can be employed.

Based on the results from each of enabled mechanisms, the status of the Supervised Entity (called Local Status) is computed. When the status of each Supervised Entity is determined, then based on each Local Supervision Status, the status of the whole MCU is determined (called Global Supervision Status).

Depending on the Local Supervision Status of each Supervised Entity and on the Global Supervision Status, the Watchdog Manager initiates a number of mechanisms to recover from supervision failures. These range from local error recovery within the Supervised Entity to a global reset of the ECU.

The following error recovery mechanisms can be employed:

1. Error Handling in the Supervised Entity:

In case the Supervised Entity is an SW-C or a CDD, then the Watchdog Manager may inform the Supervised Entity about supervision failures via the RTE Mode mechanism. The Supervised Entity may then take its actions to recover from that failure.
The Watchdog Manager may register an entry with the Diagnostic Event Manager (DEM) when it detects a supervision failure. A Supervised Entity may take recovery actions based on that error entry.

2. Partition Shutdown

If the Watchdog Manager module detects a supervision failure in a *Supervised Entity* which is located in a non-trusted partition, the Watchdog Manager module may request a partition shutdown by calling the BswM.

3. Reset by Hardware Watchdog

The Watchdog Manager indicates to the Watchdog Interface when Watchdog Interface shall no longer trigger the hardware watchdog. After the timeout of the hardware watchdog, the hardware watchdog resets the ECU or the MCU. This leads to a re-initialization of the ECU and/or MCU hardware and the complete re-initialization of software.

4. Immediate MCU Reset

In case an immediate, global reaction to the supervision failure is necessary, the Watchdog Manager may directly cause an MCU reset. This will lead to a re-initialization of the MCU hardware and the complete software.

Note: The AUTOSAR Document “Explanation of Error Handling on Application Level”[29] provides additional information on error handling. Within the document it is explained how error handling can be performed and where the required data (e.g. substitute values) can be obtained from. Furthermore the document provides a detailed explanation (user´s manual) on how OS-Application/Partition termination and restart in AUTOSAR is performed.

2.3.4 Limitations

1. For Logical Supervision, Watchdog manager does not support any overlapping graphs - a checkpoint shall belong to maximum one Graph. This is required to be able to allocate a received Checkpoint notification to a Graph.
2. Watchdog Manager does not support Logical Supervision of concurrently executed Supervised Entities, because it follows only one instance of a Graph at a time.
3. In order to shutdown or restart (as error reaction) a partition containing Supervised Entities, the integrator code (OS Application's restart task) must deactivate (or deactivate + activate) all Supervised Entities of the involved partition, by calling available functions of Watchdog Manager.

[29] Explanation of Error Handling on Application Level, R4.2 Rev 1, Chapter 8, Chapter 10

2.3.5 References to AUTOSAR Documents

Source: Requirements on AUTOSAR Features, V1.2.1, R4.1 Rev 2

[RS_BRF_00131] AUTOSAR shall support program flow monitoring
⌈

Type:	Valid
Description:	AUTOSAR shall support logical and temporal program flow monitoring to detect if program flow control is violated. AUTOSAR shall offer support for ensuring that the program flow monitoring mechanisms are working properly
Rationale:	Using flow control to detect if a software components runs wild is an established safety feature Using program flow control to detect if a runnable (or a sequence of runnables) is executed out of order or not at all is a well established safety feature
Use Case:	To detect a defective program sequence. A defective program sequence exists, if the individual elements of a program (for example, software modules, subprograms or commands) are processed in the wrong sequence or period of time, or if the clock of the processor is faulty
Dependencies:	--
Supporting Material:	ISO 26262-5:2011 Annex D, ISO 26262-6:2011

⌋ (RS_Main_00010)

2.3.6 References to ISO26262

The following references to the ISO26262 standard are related to the aspects of freedom from interference for software components with different ASIL ratings. Concepts related to logical supervision are covered.

ID	***ISO26262 Reference***
03	Part 6: [D.2.1]
08	Part 6: [D.2.2]
09	Part 6: [7.4.14] Table 4: 1d, 1e

Table 6: ISO26262 Logical Supervision References

2.4 End-2-End Protection

In a distributed system, the exchange of data between a sender and the receiver(s) can affect functional safety, if its safe behavior safety depends on the integrity of such data (see "Exchange of Information" fault example in the beginning of this chapter). Therefore, such data shall be transmitted using mechanisms to protect it against the effects of faults within the communication link.

2.4.1 Fault Models

According to ISO 26262[30], the following Exchange of Information-related faults can be considered for each sender or each receiver software component executed in different software partitions or ECUs:

- Repetition of information;
- Loss of information;
- Delay of information;
- Insertion of information;
- Masquerade or incorrect addressing of information;
- Incorrect sequence of information;
- Corruption of information;
- Asymmetric information sent from a sender to multiple receivers;
- Information from a sender received by only a subset of the receivers;
- Blocking access to a communication channel.

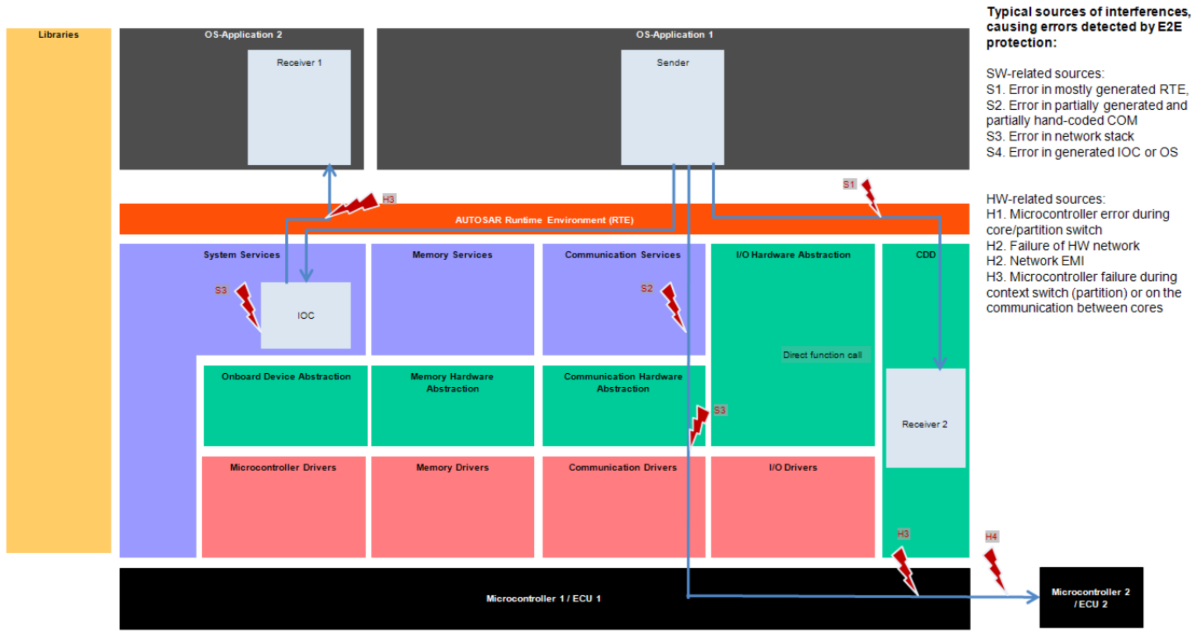
Libraries
OS-Application 2
Receiver 1
OS-Application 1
Sender
S1
H3
AUTOSAR Runtime Environment (RTE)
System Services
S3
IOC
Memory Services
Communication Services
S2
I/O Hardware Abstraction
CDD
Direct function call
Onboard Device Abstraction
Memory Hardware Abstraction
Communication Hardware Abstraction
S3
Receiver 2
Microcontroller Drivers
Memory Drivers
Communication Drivers
I/O Drivers
H3
H4
Microcontroller 1 / ECU 1
Microcontroller 2 / ECU 2
Typical sources of interferences, causing errors detected by E2E protection:
SW-related sources:
S1. Error in mostly generated RTE,
S2. Error in partially generated and partially hand-coded COM
S3. Error in network stack
S4. Error in generated IOC or OS
HW-related sources:
H1. Microcontroller error during core/partition switch
H2. Failure of HW network
H2. Network EMI
H3. Microcontroller failure during context switch (partition) or on the communication between cores

Figure 12: End-2-End Protection[31]

[30] [ISO 26262-6, Annex D] D.2.4 Exchange of Information
[31] Specification of SW-C End-to-End Communication Protection Library, V3.2.1, R4.1 Rev 3

The concept of End-2-End protection assumes that safety-related data exchange shall be protected at runtime against the effects of faults within the communication link (see Figure 12). Examples for such faults are random HW faults (e.g. corrupt registers of a CAN transceiver), interference (e.g. due to EMC), systematic faults within the software implementing the VFB communication (e.g. RTE, IOC, COM and network stacks) inside the ECU and outside, such as on Gateways.

The following faults related to message exchange via communication network have been considered in the End-2-End Library.

Fault Model	***Description***
Repetition of information	A type of communication fault, were information is received more than once.
Loss of information	A type of communication fault, were information or parts of information are removed from a stream of transmitted information.
Delay of information	A type of communication fault, were information is received later than expected.
Insertion of information	A type of communication fault, were additional information is inserted into a stream of transmitted information.
Masquerading	A type of communication fault, were non-authentic information is accepted as authentic information by a receiver.
Incorrect addressing	A type of communication fault, were information is accepted from an incorrect sender or by an incorrect receiver.
Incorrect sequence of information	A type of communication fault, which modifies the sequence of the information in a stream of transmitted information.
Corruption of information	A type of communication fault, which changes information.
Asymmetric information sent from a sender to multiple receivers	A type of communication fault, were receivers do receive different information from the same sender.
Information from a sender received by only a subset of the receivers	A type of communication fault, were some receivers do not receive the information
Blocking access to a communication channel	A type of communication fault, were the access to a communication channel is blocked.

Table 7: Fault Models of a Communication Network[32]

[32] Specification of SW-C End-to-End Communication Protection Library, V3.2.1, R4.1 Rev 3, Chapter 4.3.3

2.4.2 Description

From the perspective of Software Components, data transmission via the RTE behaves like a simple point-to-point connection. However, the implementation of this abstraction requires a highly complex infrastructure made up of software layers, communication stacks, drivers and the underlying hardware. Along with the complexity, the number of potential sources for failures also increases.

The use of the End-2-End protection mechanism assumes that the integrity of safety-relevant data has to be maintained during communication, protecting the data against the effects of faults within the communication link.

The most important aspects of the End-2-End protection are the standardization of the protection capabilities and the flexible applicability of the mechanism. Mechanisms for safe data communication within and between ECUs though the concept of End-2-End protection will be described in this chapter.

The architecture of the End-2-End protection is implemented as follows: Data Elements consisting of Application Data are extended on the sender side with additional control information, the End-2-End header. The control information usually contains a Checksum, a Counter and other options. The extended data element is provided to the RTE for transmission, as shown in Figure 13. It shows the principle of E2E, but not all details required for implementation. Especially the usage of the RTE Data Transformer to encode/decode complex data elements is omitted for simplicity.

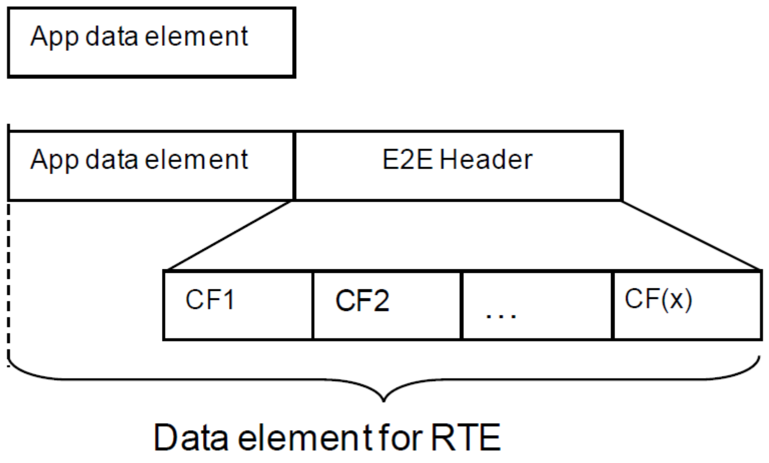

Figure 13: Data Element for RTE[33]

Data Elements are verified at the receiver side by processing the contents of the End-2-End header against the Application Data. After the received data element is processed and accepted as correct, the control information is removed and Application Data is provided to the target Software Component.
The error handling is performed at the receiver.

[33] Specification of SW-C End-to-End Communication Protection Library, V3.2.1, R4.1 Rev 3, Chapter 8.1

2.4.2.1 End-2-End Profiles

AUTOSAR specifies a set of standardized and configurable End-2-End profiles, which implement a set of protection mechanisms and specify the data format for the attached End-2-End header.

An End-2-End Profile uses a subset of the following data protection mechanisms:[34]

1. CRC Checksum, provided by the CRC library;
2. Sequence Counter incremented at every transmission request, the value is checked at receiver side for correct incrementation;
3. Alive Counter incremented at every transmission request, the value checked at the receiver side if it changes at all, but correct incrementation is not checked
4. A specific ID for every port data element sent over a port (global to system, where the system may contain potentially several ECUs).
5. Timeout detection: Receiver communication timeout and Sender acknowledgement timeout

Three End-2-End Profiles are specified in the AUTOSAR Standard, Profile 1 with two variants, End-2-End Profile 2 and End-2-End Profile 4. Upcoming releases will also specify Profiles 5 and 6.

Only the standardized End-2-End profiles shall be used, non-standard End-2-End Profile configurations may only be used in special cases, such as for legacy software.

The protection mechanisms of the End-2-End Profile 1 are described in Table 8 as follows:

Mechanism	**Description**	**Fault Model**
Counter	A 4Bit Counter is incremented with every Send-Request. This Value is explicitly sent.	Repetition, deletion, insertion, incorrect sequence
Timeout	Using a non-blocking read, the receiver can determine if the value of the counter has been increased.	Deletion, delay
Data ID	Each sender-receiver port has a unique 16-Bit ID, which is used in the CRC calculation. The CRC calculation is illustrated in Figure 14. The Data ID value is not explicitly sent. As the ID is only known at the sender and the receiver, the CRC calculation can only be correctly performed by the corresponding partners.	Insertion, addressing faults
CRC	A CRC Checksum (8-Bit) calculation is performed over all data elements, the Counter and the Data ID. This value is explicitly sent.	Corruption

Table 8: Mechanisms in End-2-End Profile 1

[34] Specification of SW-C End-to-End Communication Protection Library, V3.2.1, R4.1 Rev 3, SWS_E2E_00221

Figure 14 illustrates how the CRC calculation is performed in the End-2-End Profile 1. The value of Data ID is calculated into the CRC value, so both communication partners must use the same Data ID to correctly verify the CRC Checksum of a message.

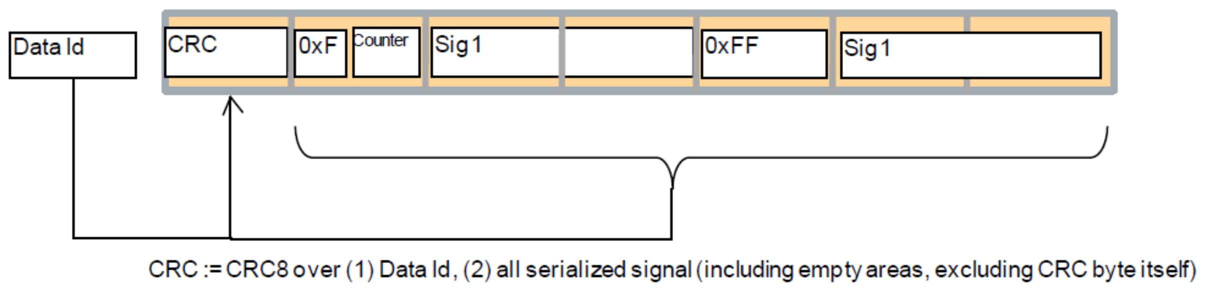
Data Id
CRC
0xF
Counter
Sig 1
0xFF
Sig 1
CRC := CRC8 over (1) Data Id, (2) all serialized signal (including empty areas, excluding CRC byte itself)

Figure 14: CRC Calculation in End-2-End Profile 1[35]

Although the length of the Data ID is 16 bits, leading to a large number of individual Data IDs, the length of the CRC checksum is only 8 bits. This means that different Data IDs will produce the same CRC checksum, thus limiting the number of independent Data IDs.

If a message is routed to the wrong destination, e.g. due to Bit-flips in a gateway, and the Data IDs produce the same CRC checksum, then the receiver would accept the misdirected message, assuming that the current counter value and the length of the message are both correct. The extent of the underlying protection against Addressing Faults is diminished. This fault model is called Masquerading.
It is possible to restrict the Data ID values so there is no overlap in the CRC Checksums. This however limits the number of independent Data IDs to 255.

[35] Specification of SW-C End-to-End Communication Protection Library, V3.2.1, R4.1 Rev 3, Chapter 8.3.4

The End-2-End Profile 2 takes a different approach in the use of the Data ID protection mechanisms. Each sender-receiver port pair has a list of Data IDs. The current value of the sequence counter determines which Data ID is used.

An appropriate selection of Data IDs is required to increase the number of messages for which detection of masquerading is possible. However, there will be overlaps of the 8-Bit Data ID and Counter values, limiting the number of independent Data IDs and Counter values to 256.
If a single erroneously received message does not violate the safety goal of the system, then the End-2-End Profile 2 allows for protection against masquerading for a greater number of messages.

Mechanism	Description	Fault Model
Sequence Number (Counter)	A 4Bit Counter is incremented with every Send-Request. This Value is explicitly sent.	Unintended message repetition, message loss, insertion of messages, re-sequencing
Message Key used for CRC calculation (Data ID)	8 bit (not explicitly sent) The Data ID used for CRC calculation is an element of a pre-defined list and depends on the current value of the Counter. The list of Data IDs is unique for each Data Element and only known to the sender and the receiver.	Insertion of messages, masquerading
Safety Code (CRC)	A CRC Checksum (8-Bit) calculation is performed over all data elements, the Counter and the Data ID. This value is explicitly sent.	Message corruption, insertion of messages (masquerading)
Timeout (detection and handling implemented by SW-C)	Timeout detection must be implemented by the SW-C.	Message loss, message delay

Table 9: Mechanisms in End-2-End Profile 2

AUTOSAR supports PDUs up to 4kB in size, either through the TCP/IP stack or through TP services of FlexRay TP, CAN TP, etc. The End-2-End Profiles 1 and 2 support an ASIL-D compliant transmission of up to 30 or 42 byte PDUs, due to the short 8-Bit CRC checksum.
The AUTOSAR Release 4.2.1 introduces a new End-2-End Profile. The End-2-End Profile 4 is specifically designed for ASIL-D compliant transmission of long data. This is supported by the use of a special 32-Bit CRC polynomial. This polynomial is superior to the widely used IEEE 802.3 CRC, as it provides a higher Hamming Distance for long data.

Mechanism	Description	Fault Model
Counter	A 16 Bit Counter is incremented with every Send-Request. This Value is explicitly sent.	Unintended message repetition, message loss, insertion of messages, re-sequencing
CRC	The 32 Bit CRC is calculated over the entire E2E header (excluding the CRC bytes) and over the user data. This Value is explicitly sent. Note: This CRC polynomial is different from the CRC-polynomials used by FlexRay, CAN, LIN and TCP/IP.	Message corruption, insertion of messages (masquerading)
Data ID	The 32 Bit Data ID shall be unique for a specific data element within the network of ECUs. This Value is explicitly sent.	Insertion of messages, masquerading
Timeout (detection and handling implemented by SW-C)	The receiver reads the currently available data, i.e. checks if new data is available. Then, by means of the counter, the receiver can detect loss of communication and timeouts.	Message loss, message delay

Table 10: Mechanisms in End-2-End Profile 4

The End-2-End Profile 4 header provides the following control fields, which are transmitted together with the protected data.

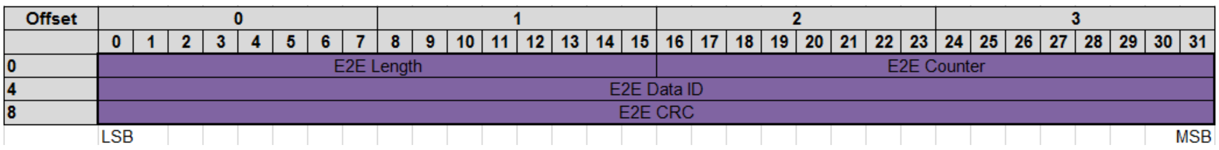

Figure 15: End-2-End Profile 4 header

Contrary to E2E Profiles 1 and 2, there is an explicit transmission of the data length, as data packets do not have a standard size. The 16 bits Length field is introduced to support variable-size data, which can have a different length in each transmission cycle. Also there is an explicit transmission of the Data ID.

2.4.2.2 End-2-End State Machine

Data Elements are verified at the receiver side by processing the contents of the End-2-End header against the Application Data using the End-2-End Profile´s check-function. It determines whether the received data of this cycle is correct and provides additional information in case of detected faults.

The AUTOSAR Release 4.2.1 introduces a State Machine, which helps to determine whether the received Application Data is acceptable with a greater level of detail. A new level of abstraction is introduced, so applications receive an overall status of the communication, instead of dealing with the status of every single message.
The new state machine supports configurable settings for the number of lost or repeated packets, recoverable and non-recoverable communication faults, as well as initialization of communication. Figure 16 illustrates the design and features of the state machine.

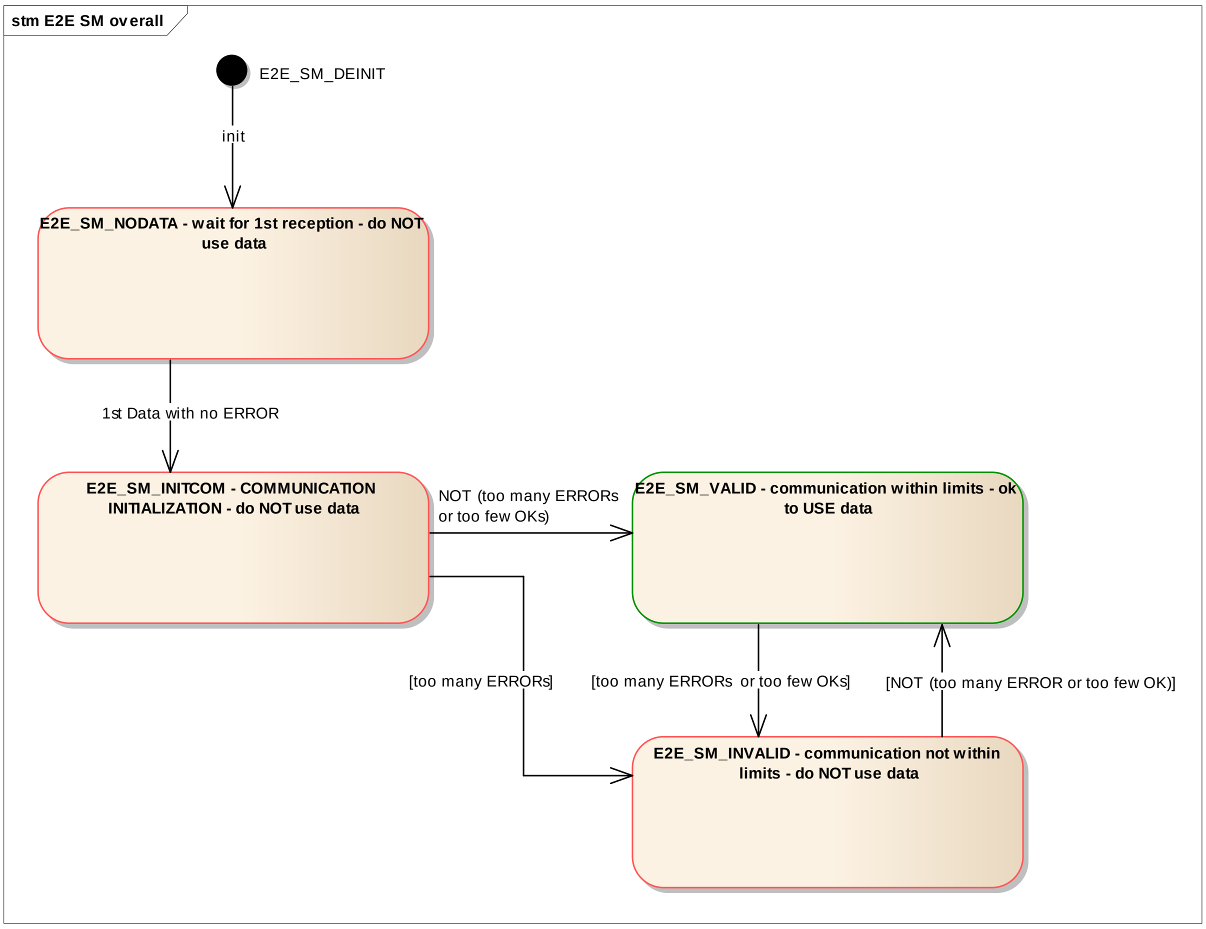
stm E2E SM overall
E2E_SM_DEINIT
init
E2E_SM_NODATA - wait for 1st reception - do NOT use data
1st Data with no ERROR
E2E_SM_INITCOM - COMMUNICATION INITIALIZATION - do NOT use data
NOT (too many ERRORs or too few OKs)
E2E_SM_VALID - communication within limits - ok to USE data
[too many ERRORs]
[too many ERRORs or too few OKs]
[NOT (too many ERROR or too few OK)]
E2E_SM_INVALID - communication not within limits - do NOT use data

Figure 16: End-2-End State Machine[36]

[36] Specification of SW-C End-to-End Communication Protection Library, V3.0.0-0.10.4, R4.1 Rev 3, Chapter 7.8.1

2.4.2.3 Integration of the End-2-End Protection Library

To enable the proper usage of the End-2-End Library, different solutions are possible. They may depend on the integrity of RTE, COM or other basic software modules as well as the usage of other SW/HW mechanisms (e.g. memory partitioning).

The End-2-End Library can be used to protect safety-related data elements exchanged between SW-Cs by means of End-2-End Protection Wrapper.
Furthermore, the End-2-End Library can be used to protect safety-related I-PDUs by means of COM Callouts.

It is also possible to have mixed scenarios, where some data elements are protected at the SW-C level (e.g. with End-2-End protection Wrapper) and some with COM End-2-End callouts.

Introduced in AUTOSAR Release 4.2.1, the RTE Data Transformer can also be used to protect data exchange of complex data elements between ECUs at the RTE level.

2.4.2.4 End-2-End Protection Wrapper

The End-2-End Protection Library can be used to protect the data communication between SW-Cs at the RTE level. To accomplish this, the End-2-End Protection Wrapper functions as a wrapper over the Rte_Write and Rte_Read functions, which are offered to SW-Cs. The End-2-End Protection Wrapper encapsulates the Rte_Read/Write invocations of the Software Component and protects the data exchange using the End-2-End Library.

In this approach, every safety-related SW-C has its own additional sub-layer (a .h/.c file pair) called the End-2-End Protection Wrapper, which is responsible for marshalling of complex data elements into the layout identical to the corresponding I-PDUs (for inter-ECU communication), and for correct invocation of End-2-End Library and of RTE. Please see Figure 17.

The usage of the End-2-End Protection Wrapper allows a use of VFB communication between SW-Cs, without the need of further measures to ensure VFB's integrity. The communication between such SW-Cs can be within an ECU (which means on the same or different cores or within the same or different memory partitions of a microcontroller) or across ECUs (SW-Cs connected by a VFB also using a network).

The end-to-end protection is a systematic solution for protecting SW-C communication, regardless of the communication resources used (e.g. COM and network, OS/IOC or internal communication within the RTE). Relocation of SW-Cs may only require selection of other protection parameters, but no changes on SW-C application code.

Also, the use of the End-2-End protection wrapper supports safe communication between software components despite a potentially unsafe communication software stack.

Note: The End-2-End Protection Wrapper does not support multiple instantiation of the SW-Cs. This means, if an SW-C is supposed to use End-2-End Protection Wrapper, then this SW-C must be single-instantiated. This limitation is based on the fact that multiple instances of a Software Component would have the same DataID, thus limiting the capabilities of the underlying protection mechanisms.

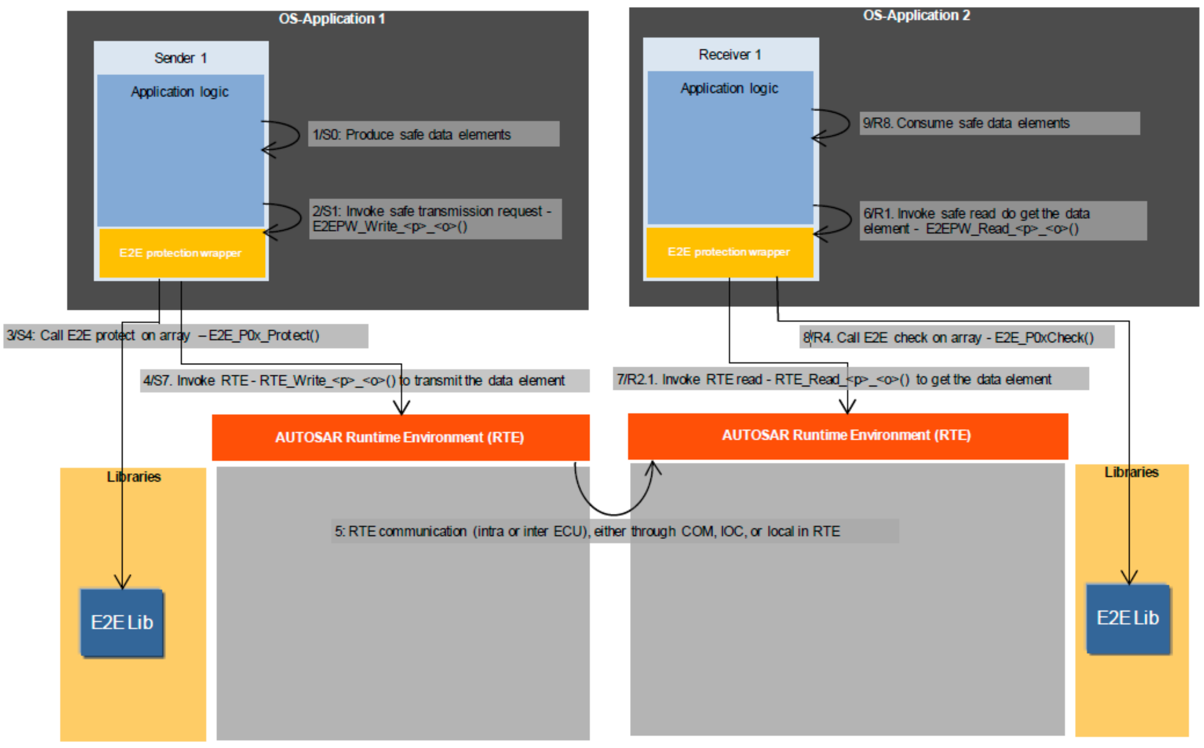
OS-Application 1
Sender 1
Application logic
E2E protection wrapper
1/S0: Produce safe data elements
2/S1: Invoke safe transmission request - E2EPW_Write_<p>_<o>()
OS-Application 2
Receiver 1
Application logic
E2E protection wrapper
9/R8. Consume safe data elements
6/R1. Invoke safe read do get the data element - E2EPW_Read_<p>_<o>()
3/S4: Call E2E protect on array – E2E_P0x_Protect()
4/S7. Invoke RTE - RTE_Write_<p>_<o>() to transmit the data element
8/R4. Call E2E check on array - E2E_P0xCheck()
7/R2.1. Invoke RTE read - RTE_Read_<p>_<o>() to get the data element
AUTOSAR Runtime Environment (RTE)
AUTOSAR Runtime Environment (RTE)
Libraries
Libraries
5: RTE communication (intra or inter ECU), either through COM, IOC, or local in RTE
E2E Lib
E2E Lib

Figure 17: End-2-End Protection Wrapper – Communication Overview[37]

[37] Specification of SW-C End-to-End Communication Protection Library, V3.2.1, R4.1 Rev 3, Chapter 13.1.1

2.4.2.5 Transmission Manager

In an ECU system where integrity of operation is not provided for COM and RTE, it is possible to transmit safety-related data through the network.

On the sender ECU, there is a dedicated SW-C called the Transmission Manager, containing End-2-End Protection Wrapper. The Transmission Manager collects safety-related data from related SW-Cs, combines them and protects them using the End-2-End Protection Wrapper. Finally, it provides the combined and protected Data as a Data Element to the RTE. Please see Figure 18.
On the receiver ECU a Transmission Manager does the reverse steps for the reception of such data.

The Transmission Manager replaces the duties of the RTE and COM, such as merging of Data Elements into PDUs and ensuring the integrity of data.

Note: The Transmission Manager SW-C module is neither part of End-2-End Library nor part of AUTOSAR. Also, the integrity of RTE communication between the SW-Cs and the Transmission Manager shall be protected by other measures.

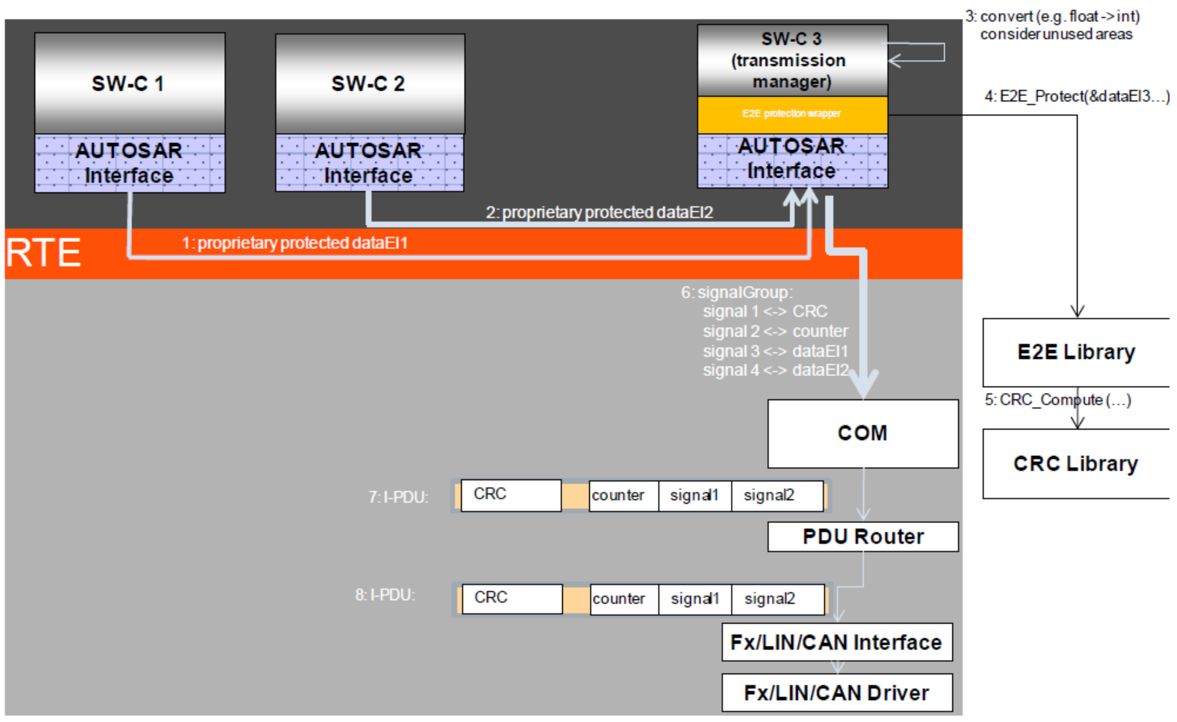

Figure 18: Transmission Manager – Sender ECU[38]

[38] Specification of SW-C End-to-End Communication Protection Library, V3.2.1, R4.1 Rev 3, Chapter 13.1.2

2.4.2.6 COM End-2-End Callout

In this approach, the End-2-End Library is used to protect the data exchange between COM modules. The End-2-End Library is invoked by COM, through COM End-2-End callouts, to protect and check the I-PDUs. The callout invokes the End-2-End Library with parameters appropriate for a given I-PDU. Please see Figure 19.

For each I-PDU to be protected and checked there is a separate callout function. Each callout function “knows” how each I-PDU needs to be protected and checked. This means that the callout invokes the End-2-End Library functions with settings and state parameters that are appropriate for the given I-PDU.

This solution works with all communication models, multiplicities offered by RTE for inter-ECU communication. In contrast to the Transmission Manager, this solution can only be used in systems where the integrity of operation of COM and RTE is provided.

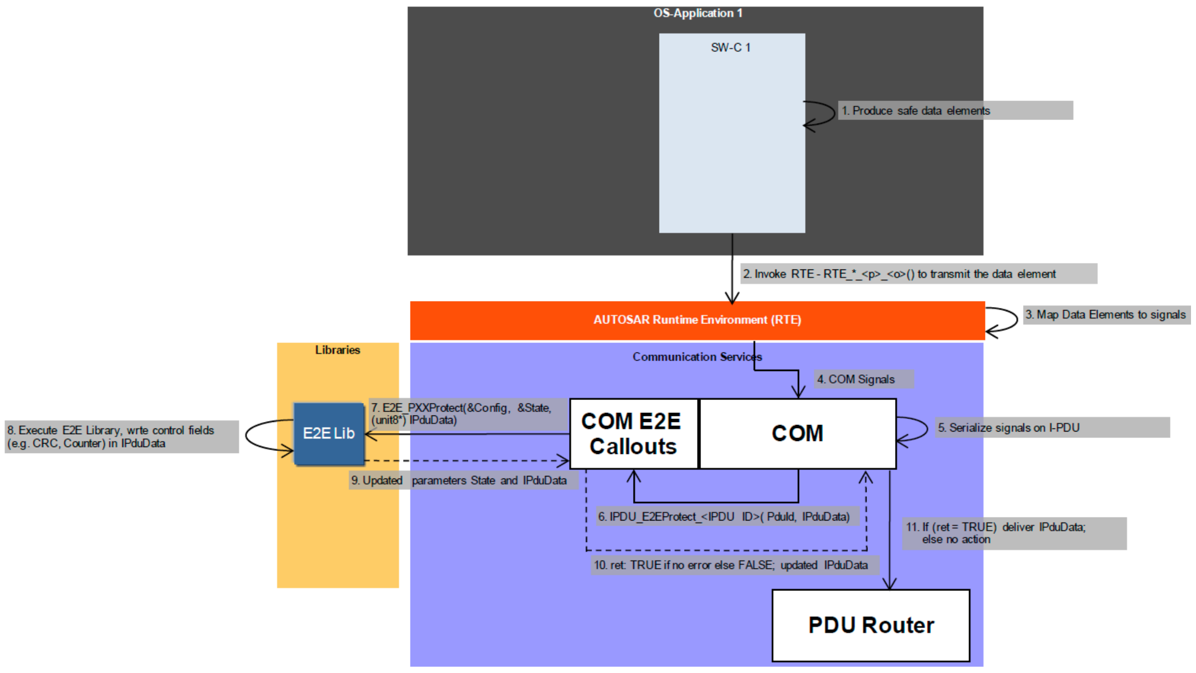
OS-Application 1
SW-C 1
1. Produce safe data elements
2. Invoke RTE - RTE_*_<p>_<o>() to transmit the data element
AUTOSAR Runtime Environment (RTE)
3. Map Data Elements to signals
Libraries
Communication Services
4. COM Signals
E2E Lib
7. E2E_PXXProtect(&Config, &State, (unit8*) IPduData)
COM E2E Callouts
COM
5. Serialize signals on I-PDU
8. Execute E2E Library, wrte control fields (e.g. CRC, Counter) in IPduData
9. Updated parameters State and IPduData
6. IPDU_E2EProtect_<IPDU ID>(PduId, IPduData)
11. If (ret = TRUE) deliver IPduData; else no action
10. ret: TRUE if no error else FALSE; updated IPduData
PDU Router

Figure 19: COM Callout - Overview[39]

[39] Specification of SW-C End-to-End Communication Protection Library, V3.2.1, R4.1 Rev 3, Chapter 13.2.1

2.4.2.7 RTE Data Transformer

Introduced in AUTOSAR Release 4.2.1, the RTE Data Transformer can be used to protect the exchange of complex data elements between ECUs.
The main difference between the previously described mechanisms for End-2-End Library invocation can be illustrated as follows:

The End-2-End protection wrapper extends the complex data element[40] under protection by adding data elements of the End-2-End header. The additional data elements can be seen by the SW-C but are ignored. The RTE Protection Wrapper, therefore, does not support the protection of individual signals, unless they are embedded within a complex data element.

COM maps the individual signals of a complex data element into PDUs. Using COM Callouts, the contents of the entire PDU are protected. The maximum PDU size is however limited by the physical properties of the interconnection bus.

Complex data elements can be prepared for transmission by being specifically arranged in a Byte-Array by a process called serialization. The serialized data array can be then protected using the End-2-End Library as a single piece. Furthermore, the serialized data array size can be dynamic on a transmission cycle basis.

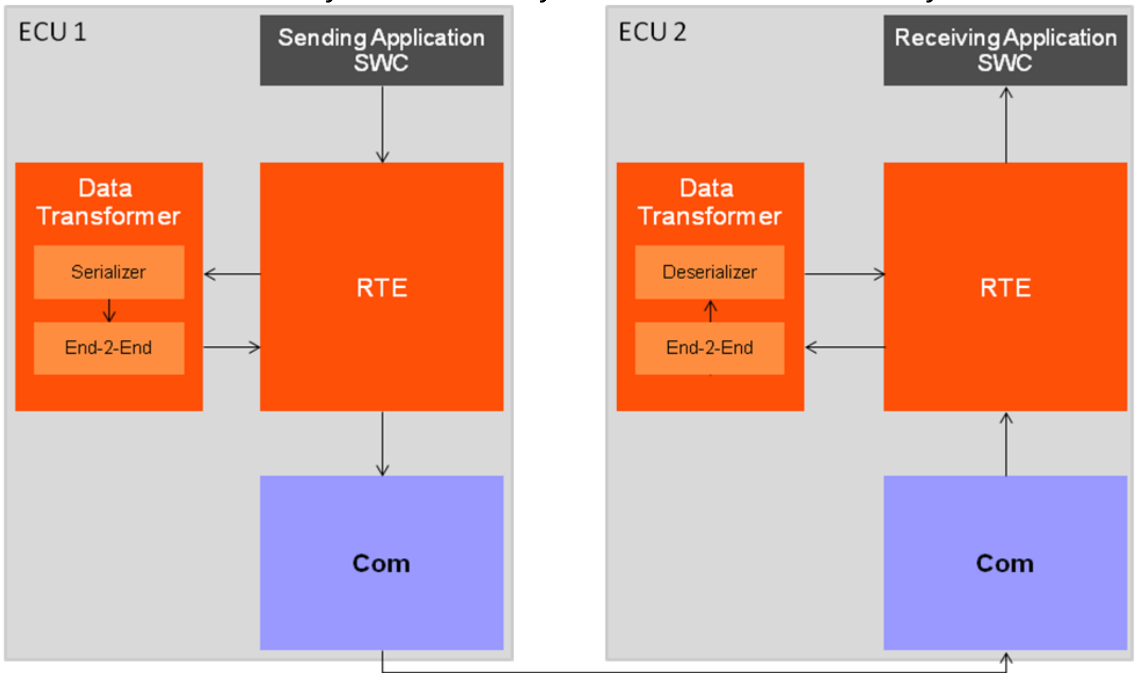

Figure 20: RTE Data Transformer - Overview[41]

As illustrated in Figure 20, the RTE Data Transformer accepts complex data (either a Sender/Receiver data element or a Client/Server operation with its arguments) from

[40] A complex data element is an instance of a complex data type. Inside a complex data type, there are one or more data types (primitive data types), like in a C struct.
[41] Based on Concept “Sender/Receiver Serialization”, V0.51, R4.2 Rev 1, Page 31, Figure 8 “Use Case 1: Transmission of large composite data types over networks with large PDUs (e.g Ethernet)”

the RTE, performs a configurable chain of data transformations (such as Serialization, End-2-End Protection, Cryptographic functions, Compression) and provides the resulting byte array, which is finally transmitted to the receiver by COM (or RTE during intra-ECU communication). Data transformation for End-2-End Protection is implemented by the End-2-End Transformer[42], which internally uses the End-2-End Library.

The complete configuration of the RTE Data Transformer is performed via AUTOSAR System Template for Inter-ECU communication and Software-Component Template for Intra-ECU communication. The resulting code is fully generated and executed via the RTE. The Software Components do not have to be aware of the specific protection mechanism used, unless detailed knowledge of the detected faults is required. The RTE Data Transformer can only be used in systems, where the integrity of operation of RTE is provided.

Note: The serialized data array size is not restricted by the PDU size of the interconnection network, as large data arrays can be transmitted using existing transport protocols.

Note: The individual data transformations are performed on data arrays and not complex data elements, therefore serialization is the first and respectively last data transformation performed by the RTE Data Transformer.

[42] Specification of Module E2E Transformer, V0.9.1, R4.2 Rev 1

2.4.3 Detection and Reaction

The End-to-End Communication Protection related features are implemented in AUTOSAR 4.0 as a standard library. This library provides End-2-End communication protection mechanisms that enable the sender to protect data prior to transmission and the receiver to detect and handle errors in the communication link at runtime.

When the End-2-End Library is used, the detection of communication faults is signaled to the receiver.

Note: The AUTOSAR Document "Explanation of Error Handling on Application Level"[43] provides additional information on error handling. Within the document it is explained how error handling can be performed and where the required data (e.g. substitute values) can be obtained from. Furthermore the document provides a detailed explanation (user´s manual) on how OS-Application/Partition termination and restart in AUTOSAR is performed.

2.4.4 Limitations

1. The appropriate usage of the End-2-End Library alone is not sufficient to achieve a safe End-2-End communication. Solely the user is responsible to demonstrate that the selected profile provides sufficient error detection capabilities for the considered network (e.g. by evaluation hardware failure rates, bit error rates, number of nodes in the network, repetition rate of messages and the usage of a gateway).

2. A communication between Software Components over the RTE is more than a simple Point-to-Point connection. Further fault models have to be considered, such as RTE errors in Data Conversion, Filtering, missing notifications, wrong order of parameters in client-server communication and delays in transmission. Those failure modes also have to be considered during the development of a safety-relevant system.

   Local RTE communication can be protected against some of the faults mentioned above by other mechanisms, such as an RTE which employs internal partitioning and other safety mechanisms and measures.

3. The use of the End-2-End protection for all Software Component communications of an ECU may be prohibitive due to runtime-overheads. Also, the limitations associated with the uniqueness of DataIDs may prevent this approach on Profile 1 and 2 due to masquerading.

4. The End-2-End Protection does not guarantee data actuality, because the End-2-End Profiles do not incorporate time stamps in the control data.

[43] Explanation of Error Handling on Application Level, R4.2 Rev 1, Chapter 8, Chapter 10

2.4.5 References to AUTOSAR Documents

Source: Requirements on AUTOSAR Features, V1.2.1, R4.1 Rev 2

[RS_BRF_00110] AUTOSAR shall offer methods to protect safety related data communication against corruption

⌈

Type:	Valid
Description:	All currently supported communication stacks (CAN, LIN, FlexRay, Ethernet) shall have a communication protection that detects corruption of communication. This includes checks whether a signal is received in sequence or not
Rationale:	To detect when data exchanged between different ECUs is corrupted or wrongly routed
Use Case:	Two SW-Cs on two ECUs exchange safety-related data
Dependencies:	--
Supporting Material:	--

⌋ (RS_Main_00010)

[RS_BRF_02104] AUTOSAR shall provide end-to-end protection support as a library

⌈

Type:	Valid
Description:	In order to support safe communication between application software components a library shall be provided that supports implementation of safe communication. This includes checking of signal integrity e. g. by checksums and sequence counters
Rationale:	Support integrity of communication data
Use Case:	Safety-related communication between too ECUs
Dependencies:	--
Supporting Material:	--

⌋ (RS_Main_00010, RS_Main_00410)

[RS_BRF_02064] AUTOSAR shall use hardware communication data integrity mechanisms

⌈

Type:	Valid
Description:	AUTOSAR shall use data integrity mechanisms which are offered by communication hardware such that major fault models described in ISO 26262 are covered
Rationale:	Cover the ISO26262 cases like: - Failure of communication peer - Message corruption - Message delay - Message loss - Unintended message repetition - Resequencing - Insertion of message and - Masquerading
Use Case:	Exchanging of information between elements executed on different ECUs including signals, data, messages, etc. Information can be exchanged using I/O-devices, data busses, etc.
Dependencies:	--
Supporting Material:	ISO 26262-5:2011 Annex D, ISO 26262-6:2011 Annex D

⌋ (RS_Main_00010)

2.4.6 References to ISO26262

The following references to the ISO26262 standard are related to the aspects of freedom from interference for software components with different ASIL ratings. Concepts related to exchange of information are covered.

ID	***ISO26262 Reference***
03	Part 6: [D.2.1]
09	Part 6: [D.2.4]
10	Part 6: [7.4.14] Table 4: 1c

Table 11: ISO26262 Exchange of Information References

3 Functional Safety Measures

In addition to Functional Safety mechanisms provided by AUTOSAR, the development of safety-relevant software is supported by Functional Safety measures which originate from AUTOSAR.

3.1 Functional Safety Measures of AUTOSAR

The following table provides a list of examples of ISO26262 Requirements mapped to the definition of AUTOSAR Basic Software.

ID	Functional Safety Measures	ISO Reference	AUTOSAR Requirement/Feature
001	Enforcement of strong typing	ISO26262-6 Table 1, 1c	AUTOSAR Meta-Model
002	Use of established design principles	ISO26262-6 Table 1, 1e	AUTOSAR Layered Architecture
003	Use of unambiguous graphical representation	ISO26262-6 Table 1, 1f	Standard representation of the AUTOSAR Meta-Model
004	Use of naming conventions	ISO26262-6 Table 1,1h	AUTOSAR Application Interfaces definition: AUTOSAR_MOD_AITable.xls AUTOSAR_EXP_AIUserGuide.pdf
005	Semi-formal Notation	ISO26262-6 Table 2, 1b	AUTOSAR Meta-Model
006	Restricted size of interfaces	ISO26262-6 Table 3, 1c	Per domain, application interfaces were proposed: AUTOSAR_EXP_AIBodyAndComfort.pdf AUTOSAR_EXP_AIChassis.pdf AUTOSAR_EXP_AIOccupantAndPedestrianSafety.pdf AUTOSAR_EXP_AIHMIMultimediaAndTelematics.pdf AUTOSAR_EXP_AIPowertrain.pdf
007	Restricted coupling between software components	ISO26262-6 Table 3, 1e	AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf Please see: Interfaces: General Rules Layer Interaction Matrix.
008	Restricted use of interrupts	ISO26262-6 Table 3, 1g	AUTOSAR_EXP_ InterruptHandlingExplanation.pdf
009	Detection of data errors	ISO26262-6 Table 4, 1c	AUTOSAR_SWS_E2ELibrary.pdf AUTOSAR_SWS_CRCLibrary.pdf
010	Control flow monitoring	ISO26262-6 Table 4, 1e	AUTOSAR_SWS_WatchdogManager.pdf

ID	Functional Safety Measures	ISO Reference	AUTOSAR Requirement/Feature
011	Graceful degradation	ISO26262-6 Table 5, 1b	AUTOSAR_EXP_ LayeredSoftwareArchitecture.pdf Please see: Integration and Runtime Aspects - Partitioning Example of restarting partition. AUTOSAR_SWS_ FunctionInhibitionManager.pdf
012	Interface test	ISO26262-6 Table 10, 1b Table 13, 1b	Acceptance Test for the AUTOSAR Stack
013	Document Management	ISO26262-8 10.4.3-10.4.6	Fulfilled by AUTOSAR Quality Management

Table 12: Mapping of ISO26262 Requirements to AUTOSAR Basic Software

3.2 Traceability

Traceability is a prerequisite for the implementation of safety-relevant systems. AUTOSAR provides traceability from the AUTOSAR project objectives to the software specifications of the AUTOSAR architecture.

3.3 Development Measures and the Evolution of the Standard

The AUTOSAR standard follows a defined life cycle, which is enforced by a dedicated Change Management. Therefore, the AUTOSAR version which is used during the product development can be easily referenced.

Systematic Faults during product development can be reduced when a defined version of AUTOSAR is used, as the specifications, the interfaces and the behavior can be clearly established.
During the development of AUTOSAR specifications, a tracking of findings and bug fixes is performed with well-established tools (“Bugzilla”). Therefore it is possible to follow the incorporation of findings and bug fixes for the users of an AUTOSAR version well ahead series production.

In model-based development, a hierarchically structured model of function blocks with well-defined inputs and outputs is used to control complexity, to model the functionality and to support code generation. Please see ISO26262 Part 6, Annex B for details. Model-based development is supported due to the use of standardized interfaces and exchange formats, as well as due to the flexibility of the AUTOSAR methodology to support extensions.

The development process of AUTOSAR Specifications involves a comprehensive review process by multiple parties and work packages. The development milestones and the associated review process conditions are defined by AUTOSAR Quality Management.

AUTOSAR supports the argumentation of Freedom from Interference by providing functional safety mechanisms. Please see Chapter 2 for details on AUTOSAR Functional Safety Mechanisms.

AUTOSAR provides a clear definition of people assignment to work packages, based on the high expertise in the respective fields.

AUTOSAR provides a definition of the generic Software Architecture, based on modularity, formality and model-based development.

3.4 Functional Safety Measures not delivered by AUTOSAR

Not all functional safety measures, which may be required for the development of safety-relevant applications, are delivered by AUTOSAR. Therefore the implementers of safety-relevant applications must ensure that the safety development life cycle is adequate.

As an example, the following functional safety measures are neither enforced nor delivered by AUTOSAR. This list does not imply completeness.

- The AUTOSAR specification does not define Safety Elements out of Context (SEooC) as described in ISO26262 Part 10, Chapter 9.
- The AUTOSAR Specification does not define the use of systematic and structured techniques for system examination, risk analysis and management, such as Hazard Analysis (HARA) and Hazard & Operability Analysis (HAZOP).
- No overall safety concept.
- No ASIL identification
- No dependent failure analysis is performed.
- No AUTOSAR safety case
- No confirmation measures
- No functional safety audits
- No conformance test
- Implementation techniques of Software Components such as low complexity, robustness, defensive programming, conventions, coding rules.
- Tracing of AUTOSAR features to Software Component implementation.
- Software integration testing
- Validation and Verification against the AUTOSAR specification.
- Defect reporting, tracking, resolution with regard to implementation.

3.5 Safety related Extensions of Methodology and Templates

The document “Safety Extensions” provides requirements upon extensions in AUTOSAR Methodology and Templates to realize and document functional safety of AUTOSAR systems. It specifies, how the AUTOSAR meta-model is to be used to enhance AUTOSAR models by information for functional safety. With the safety extensions, it is possible to:

- Describe and exchange the part of a (technical) safety concept of a system which is relevant for the realization of that system using the AUTOSAR architecture in a standardized form by means of the AUTOSAR templates.
- Provide traceability between safety-related elements of the AUTOSAR model and the safety requirements as part of the AUTOSAR templates.
- Declare the safety mechanisms/safety measures[44] that are applied for an AUTOSAR system as part of the AUTOSAR templates.
- Demonstrate the traceability between safety mechanisms/safety measures and safety requirements as part of the AUTOSAR templates.

All the safety measures and mechanisms described in “Overview of Functional Safety Measures in AUTOSAR” can be modeled and traced using the Safety Extensions as explained above.

3.6 Safety Use Case

The “Safety Use Case” is delivered as auxiliary document. It describes an exemplary safety related system using AUTOSAR based on the AUTOSAR guided tour example “Front Light Management”.
The document provides an overview of a Functional safety concept as well as the derived Technical safety concept on ECU level and is focused on AUTOSAR relevant parts. The example follows the ISO 26262 standard, but does not cover all aspects and include all details.

The safety use case shall:

- Provide an example to discuss and verify safety related concepts within AUTOSAR,
- Identify improvement potential with respect to functional safety aspects in the current AUTOSAR specifications and methodology,
- Provide a guideline for safety analyses on top of AUTOSAR methodology

Therefore the example can be adapted or changed in future to include new AUTOSAR concepts or extend the complexity of the analyzed system.

[44] In the context of this document, functional safety mechanisms are a concrete product part, such as memory protection. They are considered as specialization of functional safety measures, which also include process steps, like a review. This definition is in line with the definition given in ISO 26262 for these terms.

4 Hardware Diagnostics

Modern microcontrollers for safety-relevant applications are highly complex devices. To ensure that the desired level of integrity is achieved by the microcontroller as part of a safety-relevant system, integration and use of functional safety mechanisms and measures in hardware and software is required.
Microcontrollers must support the premise of the safety-relevant system, that the provided functionality can be trusted. Execution of Hardware Diagnostic mechanisms can support this premise. This chapter provides an overview of how hardware diagnostics are supported using AUTOSAR.

4.1 Core Test

The general objective of test by software is to detect failures in processing units which lead to incorrect results. Core Test performs test by software of processing units during microcontroller start-up and runtime.

4.1.1 Fault Models

According to ISO 26262[45], detection of failures in the following parts of the processing units are typically considered for the derivation of diagnostic coverage. The following table provides a preliminary mapping between ISO26262 and Core Test requirements.

ID	Processing unit parts	Core Test SRS Requirements
001	ALU Data Path	[SRS_CoreTst_14106] Core ALU Test
002	Registers (general purpose registers bank, DMA transfer registers…), internal RAM	[SRS_CoreTst_14104] Core Register Test
003	Address calculation (Load/Store Unit, DMA addressing logic, memory and bus interfaces)	[SRS_CoreTst_14107] Core Address Generator [SRS_CoreTst_14108] Core Memory Interfaces [SRS_CoreTst_14109] Memory Management/Protection Unit (MMU/MPU) [SRS_CoreTst_14110] Cache Controller
004	Interrupt Handling	[SRS_CoreTst_14105] Core Interrupt and Exception Detection
005	Control Logic (Sequencer, coding and execution logic including flag registers and stack control)	-
006	Configuration Registers	-
007	Other sub-elements not belonging to previous classes	-

Table 13: Mapping between Processing Unit parts and Core Test requirements

[45] [ISO 26262-5, Annex D] Table D.1 Processing Units

4.1.2 Description

The Core Test Driver is an AUTOSAR Basic Software Module which accesses the microcontroller core directly without intermediate software layers. It is located in the Abstraction Layer (MCAL).

The Core Test Driver provides several tests to verify dedicated core functionality like e.g. general purpose registers or Arithmetical and Logical Unit (ALU). Furthermore, the Core Test Driver provides services for configuring, starting, polling, terminating and notifying applications about Core Test results. It also provides services for returning test results in a predefined way.

The Core Test Driver can be used during ECU power-up and during application runtime. However it is assumed that each hardware functional block of the core under test can be accessed by the Core Test Driver exclusively.

4.1.3 Detection and Reaction

If the execution of the Core Test Driver is to be embedded into a system safety architecture concept, then it is up to the user of the Core Test Driver to choose a suitable test combination and scheduled execution order to fulfill the safety requirements of the system.

Core Test reports errors in dedicated memory and bus interfaces (e.g. Tightly Coupled Memories, caches, systems bus) and dedicated supporting functionality (e.g. interrupt controller) to the diagnostic event manager (DEM) as production errors.
Errors inside the CPU (e.g. ALU, Prefetch queue, registers) cannot be reliably reported to DEM, as these faults affect the correct operation of the Core itself.

4.1.4 Limitations

1. Transient faults are not covered by Core Test.
   The Core test can be used to detect static hardware errors during power-up and at runtime. Transient faults and intermittent faults are not covered and cannot be reliably detected by Core Test.
2. Core Test implementations may be limited to execution during start-up/power-up.
   Core Test requires exclusive access to local core resources to avoid unwanted behavior and interference between test and application during runtime. Currently, there is no resource managing entity in AUTOSAR upper layers to support exclusive access to shared resources.
3. Test results are only available to the core which executes Core Test.
   MCAL drivers intentionally miss the ability of accessing test results being executed on other cores. Currently, there is no test managing entity in AUTOSAR upper layers to handle test result processing.
4. Core Test cannot report detected faults reliably.
   Faults within the CPU itself (e.g. ALU, MAC, Registers) cannot be reliably reported to DEM, as they are being processed by the same faulty CPU.

4.1.5 References to AUTOSAR Documents

Source: Requirements on Core Test, V1.4.0, R4.2 Rev 1

[SRS_CoreTst_14104] Core Register Test Shall Be Available
⌈

Type:	valid
Description:	Shall support test according the automotive standard.
Rationale:	The automotive standard requires testing of all critical Core components.
Use Case:	Part of Core test strategy to detect failures of the Core.
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_02224)

[SRS_CoreTst_14105] Core Interrupt and Exception Detection Tests Shall Be Available
⌈

Type:	valid
Description:	Shall support test according to the automotive standard.
Rationale:	The automotive standard requires testing of all critical Core components
Use Case:	--
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_02224)

[SRS_CoreTst_14106] Core ALU Test Shall Be Available
⌈

Type:	valid
Description:	Shall support test of 'coding and execution including flag registers' as suggested by the automotive standard.
Rationale:	The automotive standard requires testing of all critical Core components.
Use Case:	--
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_02224)

[SRS_CoreTst_14107] Core Address Generator Test Shall Be Available
⌈

Type:	valid
Description:	Shall support test of 'address generation' as suggested by the automotive standard
Rationale:	The automotive standard requires testing of all critical Core components
Use Case:	--
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_02224)

[SRS_CoreTst_14108] Core Memory Interfaces Test Shall Be Available
⌈

Type:	valid
Description:	Shall support Bus test as suggested by the automotive standard
Rationale:	The automotive standard requires testing of all critical Core components
Use Case:	--
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_02224)

[SRS_CoreTst_14109] Memory Management/Protection Unit (MMU/MPU) Test Shall Be Available
⌈

Type:	valid
Description:	Shall support MMU/MPU test as suggested by the automotive standard.
Rationale:	the automotive standard requires testing of all critical Core components.
Use Case:	--
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_02224)

[SRS_CoreTst_14110] Cache Controller Test Shall Be Available

⌈

Type:	valid
Description:	Shall support Bus test as suggested by the automotive standard.
Rationale:	The automotive standard requires testing of all critical Core components. Cache controller, although not explicitly covered by the automotive standard is a standard component of the Core.
Use Case:	--
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_02224)

[SRS_CoreTst_14123] Shared Resources to Be Tested Shall Be Made Exclusively Available to Test

⌈

Type:	Valid
Description:	A mechanism for requesting and releasing shared resources in multi master systems shall be available. The caller has to handle the state of the shared resource. Saving/restoring the state prior to the call to API in NOT handled by the test itself, but rather a task of the caller.
Rationale:	In Cores some resources such as tightly coupled memory interfaces are shared with external masters, e.g. DMA. These shared resources need to be made exclusively available for testing purposes. The test can then freely manipulate them, e.g. change to test mode if supported, etc. without conflicting with the rest of the application.
Use Case:	--
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_01472,RS_BRF_01232)

[SRS_CoreTst_14117] Faults Shall Be Treated as Production Errors

⌈

Type:	Valid
Description:	The Core test module shall report detected faults inside the core to the DEM except faults detected inside the CPU itself (e.g. ALU, MAC, Registers etc.) which cannot be reliably reported.
Rationale:	React and reconfigure system according to resource availability.
Use Case:	--
Dependencies:	--
Supporting Material:	--

⌋(RS_BRF_02024,RS_BRF_02168)

4.1.6 References to ISO26262

The following references to the ISO26262 standard are related to the aspects of test by software for Processing Units.

ID	*ISO26262 Reference*
010	Part 5: [D.2.3.1] Self Test by software
011	Part 5: [Table D.13] Combinatorial and sequential logic
012	Part 5: [Table D.4] Processing Units
013	Part 5: [Table D.1] Specific semiconductor elements – Processing units

Table 14: ISO26262 Core Test References

4.2 RAM Test

The general objective of RAM Test is to detect permanent failures which can cause corruption in the volatile memory.

4.2.1 Fault Models

According to ISO 26262[46], detection of the following failures in the volatile memory is typically considered for the derivation of diagnostic coverage. The following table provides a preliminary mapping between ISO26262 and RAM Test requirements.

ID	**Failure Modes of Volatile Memory**	**RAM Test SRS Requirements**
001	Low Coverage (60%): Stuck-at for data, addresses and control interface, lines and logic.	[SRS_RamTst_13822] A Test algorithm with low coverage shall be available
002	Medium Coverage (90%): d.c. fault model for data, addresses (includes address lines within same block and inability to write to cell) and control interface, lines and logic	[SRS_RamTst_13823] A Test algorithm with medium coverage shall be available
003	Medium Coverage (90%): Soft error model for bit cells	-
004	High Coverage (99%): d.c. fault model for data, addresses (includes address lines within same block and inability to write to cell) and control interface, lines and logic	[SRS_RamTst_13824] A Test algorithm with high coverage shall be available
005	High Coverage (99%): Soft error model for bit cells	

Table 15: Mapping between Volatile Memory Failure Modes and RAM Test requirements

[46] [ISO 26262-5, Annex D] Table D.1 Volatile Memory

4.2.2 Description

The RAM Test Driver is an AUTOSAR Basic Software Module which accesses the microcontroller RAM directly without intermediate software layers. It is located in the Abstraction Layer (MCAL).

The RAM Test driver performs a test of the physical health of the RAM cells, it is not intended to test the contents of the RAM. Furthermore, RAM used for registers is also tested.
Different algorithms exist to test RAM. They target different sets of fault models, achieve different coverages, result in different runtimes and are either destructive or non-destructive. Coverage also depends on the underlying physical RAM architecture.
An ECU safety analysis must be performed to determine which RAM Test diagnostic coverage rate (Low, Medium or High) is required. Appropriate RAM Test algorithms and further configuration parameters are then selected at compile time. At run time, the application software may choose between the compiled algorithms (and between further parameters).
The RAM Test driver supports synchronous test methods called “foreground test” and asynchronous tests called “background test”. During the execution of a RAM test algorithm, no other software shall be allowed to modify the RAM area under test.

4.2.3 Detection and Reaction

During the execution of non-destructive tests, the RAM Test module saves the contents of the RAM area under test and restores the original contents thereafter.
RAM Test reports errors to the diagnostic event manager (DEM) as production errors.

4.2.4 Limitations

1. Transient faults are not covered by RAM Test.
   RAM Test can be used to detect static hardware errors during power-up and at runtime. Transient faults and intermittent faults are not covered and cannot be reliably detected by RAM Test.
2. During the execution of a RAM test algorithm, no other software and hardware shall be allowed to modify the RAM area under test The RAM Test module cannot ensure data consistency (e.g. during NMI, DMA transfers, multiple active cores in a Multicore system). Therefore the execution of RAM Test may be limited to the power-up/sleep/shutdown phase of a microcontroller.
3. Destructive tests cause corruption of contents in memory under test.
   During the execution of destructive tests, the contents of RAM area under test are not saved by the RAM Test module.

4.2.5 References to AUTOSAR Documents

Source: Requirements on RAM Test, V2.0.1, R4.2 Rev 1

[SRS_RamTst_13822] A Test algorithm with low coverage shall be available
⌈

Type:	New
Description:	A test algorithm, which fulfils a diagnostic coverage of 60 % shall be available.
Rationale:	Detect permanent faults in RAM.
Use Case:	Support of EOL, quick start-up tests and where low diagnostic coverage tests are required, e.g. if system has safety goals with low ISO 26262 ASIL rating only.
Dependencies:	--
Supporting Material:	ISO 26262-5:2011, Tables 4, 5, D.1 and D.6, sections D.2.5.1, D.2.5.2 and D.2.5.3

⌋(RS_BRF_00129, RS_BRF_02224,RS_BRF_01472)

[SRS_RamTst_13823] A Test algorithm with medium coverage shall be available
⌈

Type:	New
Description:	A test algorithm, which fulfils a diagnostic coverage of 90 % shall be available.
Rationale:	Detect permanent faults in RAM.
Use Case:	Support of EOL, start-up tests and where medium diagnostic coverage tests are required, e.g. if the latent fault metric of ISO 26262 for the ASIL level of the safety goals of a system can be achieved with medium coverage.
Dependencies:	--
Supporting Material:	ISO 26262-5:2011, Tables 4, 5, D.1 and D.6, sections D.2.5.1, D.2.5.2 and D.2.5.3

⌋(RS_BRF_00129, RS_BRF_02224,RS_BRF_01472)

[SRS_RamTst_13824] A Test algorithm with high coverage shall be available
⌈

Type:	New
Description:	A test algorithm, which fulfils a diagnostic coverage of 99 % shall be available.
Rationale:	Detect permanent faults in RAM.
Use Case:	Support of EOL, diligent start-up, shut-down or runtime tests and where high diagnostic coverage tests are required, e.g. if system has a safety goal with high ISO 26262 ASIL rating.
Dependencies:	--
Supporting Material:	ISO 26262-5:2011, Tables 4, 5, D.1 and D.6, sections D.2.5.1, D.2.5.2 and D.2.5.3

⌋(RS_BRF_00129, RS_BRF_02224,RS_BRF_01472)

[SRS_RamTst_13825] The RAM Test Module shall be usable to comply with requirements of the different ASIL levels of ISO 26262.
⌈

Type:	New
Description:	The RAM Test Module shall provide and document (fault models and fault coverage) diagnostic capability for permanent faults in RAMs to enable fulfillment of the latent fault metric targets of ISO 26262 for the different ASIL levels.
Rationale:	Usability of AUTOSAR for systems which need to comply with ISO 26262.
Use Case:	--
Dependencies:	--
Supporting Material:	ISO 26262-5:2011, Tables 4, 5, D.1 and D.6, sections D.2.5.1, D.2.5.2 and D.2.5.3

⌋(RS_BRF_02048,RS_BRF_02064)

4.2.6 References to ISO26262

The following references to the ISO26262 standard are related to the aspects of RAM Test.

ID	***ISO26262 Reference***
015	Part 5: [D.2.5.1] RAM Pattern test
016	Part 5: [D.2.5.3] RAM March test
012	Part 5: [Table D.6] Volatile Memory
013	Part 5: [Table D.1] General semiconductor elements – Volatile Memory

Table 16: ISO26262 RAM Test References

5 Appendix

5.1 Acronyms and abbreviations

<Used acronyms and abbreviations not contained in the AUTOSAR glossary>

Abbreviation / Acronym:	**Description**
HARA	Hazard Analysis
HAZOP	Hazard & Operability Analysis
SEooC	Safety Element out of Context
HTM	Hardware Test Manager
HTMSS	Hardware Test Manager on Startup and Shutdown
ASIL	Automotive Safety Integrity Level
DMA	Direct Memory Access
EMC	Electromagnetic Compatibility
IOC	Inter-OS-Application Communicator
CRC	Cyclic Redundancy Check
TP	Transport Protocol
BIST	Built In Self Test
FTTI	Fault Tolerant Time Interval
MSTP	Microcontroller Specific Test Package

5.2 Related Documents

[1] ISO26262 International Standard, First edition 2011-11-15

[2] Specification of Operating System

[3] Requirements on AUTOSAR Features

[4] Layered Software Architecture

[5] Specification of Watchdog Manager

[6] Specification of SW-C End-to-End Communication Protection Library

[7] Specification of Module E2E Transformer

[8] General Specification on Transformers

[9] Specification of ECU State Manager

[10] Specification of ECU State Manager with fixed state machine

[11] Functional Safety analysis of an exemplary system using AUTOSAR

[12] Specifications of Safety Extensions

[13] Specification of ECU Configuration

[14] Technical Safety Concept Status Report

[15] Explanation of Error Handling on Application Level

[16] Specification of Core Test

[17] Requirements on Core Test

[18] Specification of RAM Test

[19] Requirements on RAM Test