Contract number: ITEA2 – 10039

Safe Automotive soFtware architEcture (SAFE)

ITEA Roadmap application domains:

Major: Services, Systems & Software Creation

Minor: Society

ITEA Roadmap technology categories:

Major: Systems Engineering & Software Engineering

Minor 1: Engineering Process Support

WP3 - Deliverable D3.3.2

Requirement specification for a multi-class ECU concept

Due date of deliverable: 27/12/2014

Actual submission date: 28/03/2013

Start date of the task: 28/03/2012 **Duration:** 23 months

Project coordinator name: Stefan Voget (Conti-G)

Organization name of lead contractor for this deliverable: Continental France

Editor: Philippe Cuenot (Conti-F)

Contributors: Philippe Cuenot (Conti-F), Roland Geiger (ZF), Abdelillah Ymlahi-Ouazzani (Valeo) Andreas Eckel (TTTech)

Reviewers: Lionel Guichard (Conti-F), Roland Geiger (ZF), Abdelillah Ymlahi-Ouazzani (Valeo)

Revision chart and history log

Version	**Date**	**Reason**
0.1	2012-07-24	Initialization of chapter (yellow part are out dated text)
0.2	2013-10-24	Integration of change from ZF and new chapter 7 organization
0.3	2013-12-03	Integration of Valeo contribution and review from chapter 7.3 new requirement
0.4	2014-01-06	Conti-F update of document versus discussion status at 12/2013
0.5	2014-06-15	Incorporation of ZF inputs and selective reviews
0.6	2014-06-24	Incorporation of TTTech inputs by ZF
1.0	2014-06-30	Complete and final document review

1 Table of contents

1 Table of contents 3
2 List of figures 4
3 Executive Summary 5
4 Introduction and overview of the document 6
4.1 Detailed scope of WT 3.3.2 6
4.2 Structure of the document 7
5 Overview on ISO 26262 8
6 State of the art of AUTOSAR mixed criticality ECU systems 10
6.1 Independence and non-interference 10
6.2 Critical resources 10
6.3 AUTOSAR Document Reference 12
7 Architecture analysis for AUTOSAR mixed criticality ECU systems 14
7.1 Current status of AUTOSAR 14
7.2 New proposed Architecture 15
7.3 New Requirements 17
7.3.1 Failure Modes 17
7.3.2 Design pattern 21
7.3.3 Use case definition 21
8 Conclusions and Outlook 30
9 References 31
10 Acknowledgments 32

2 List of figures

Figure 1: Overview on ISO 26262 (Relevant parts highlighted) 8
Figure 2: AUTOSAR Safety Concept........14
Figure 3: TTTECH Safety Concept based on partionining........15
Figure 4: Extended Safety Concept........16

3 Executive Summary

Main goal of the work task 3.3.2 is to provide requirements and concepts for technical bricks which are necessary to achieve independence and non-interference in a multi-class concept for a single ECU. A concept is attributed as multi-class if the concept allows several software functions of different ASIL or QM classification to coexist on the same ECU, while fulfilling all relevant requirements of ISO 26262.

Besides this, the document gives an overview on the relevant sections of ISO 26262 and on the WT 3.3.2 dedicated requirements, which were derived from the ISO 26262 analysis of WT 2.1 and from the use cases described in WT 2.3. In an additional section, the current achievements on the requirements allocated to WT 3.3.2 are presented.

Existing AUTOSAR-based solutions rely on software partitions which are controlled by the operating system. They are defined by introducing and making use of so-called OS applications as new OS-related design elements. The AUTOSAR specifications imply that the infrastructure software (basic software plus RTE) then either fulfills ASILD (if safety relevant) or QM (if not safety relevant), but nothing in between is possible. Because the classification in detail is project specific and because a lower number of partitions can be handled easier this seems somehow natural, but it implies that any safety relevant function needs to be analysed and implemented according to methods and mechanisms related to ASIL D. The demands of ISO 26262 related to ASIL D are from technical and from methodological point of view most demanding. This raises the question: Does it make sense to treat many software parts according to higher demands than really necessary?

The EUROSTAR project Safe-e [2] has developed a safety partition for single core and multicore ECU to enable the use of AUTOSAR QM software building blocks in combination with safety-relevant software modules for applications up to ASILD level. In such a case QM software building blocks can be integrated in up to ASIL-D applications without running the danger of ASIL related functions being impacted by non-safety-relevant QM functions/software building blocks.

Partitioning of the AUTOSAR basic software, especially when considering extensive use of ECU-local IO with directly connected sensors and actuators, is still not supported as necessary. Therefore WT3.3.2 aims at defining requirements to support a multi-class concept for a single ECU by supporting the coexistence of several software architectural elements with concurrent ASIL levels (A, B, C, D) and QM, including the basic software. Since the basic software safety requirements strongly depend on the project, e. g. when defining which concrete resources need to fulfill a certain safety criticality class, AUTOSAR cannot provide “ready-to-use”- solutions.

The contribution of WT 3.3.2 to the SAFE meta-model is not considered in details as only concepts are introduced, and as AUTOSAR IP right for publication has to be considered.

The contents of the document may serve as input to AUTOSAR.

4 Introduction and overview of the document

The document at hand provides a state of the art analysis of the AUTOSAR mechanisms as being considered in Release 4.0.3 and elements defined in the AUTOSAR safety work package, to support a safety partition and to guarantee the independence and non-interference of that partition from the rest. Moreover, it will analyse standard ECU AUTOSAR architecture in order to identify entities to be controlled to guarantee the freedom from interference with the scope of a multi-class ECU. The safety mechanisms to control these entities are safety patterns that are today present in the AUTOSAR specification, but there are still gaps which have to be filled, in particular for the basic use of standard input/output signals.

The aim of this WT3.3.2 is to define design elements as additional requirements with respect to existing AUTOSAR definitions to allow the control of safe execution in the sense that the execution will be protected from failure propagation for independent software partitions, such that the requirements of the ISO 26262 for safe execution are fulfilled.

Some patterns will then be briefly presented to be able to later elaborate further details on software or hardware specification for implementation. The Safe-E demonstrator, related to the SAFE project, will implement a partial solution to prove the feasibility of the concept.

4.1 Detailed scope of WT 3.3.2

As part of work package 3, WT3.3.2 deals with the definition of requirements to support an ECU multi-class concept. The different ASIL software applications, clustered in separate software partitions, have to guarantee freedom from interference as documented in ISO26262-6 annex D. The technical concept, as low level requirement of hardware and software bricks, has to extend the actual capabilities of the AUTOSAR R4.0.3 release. Mainly the concept and the interrelations are considered. Before starting, some terms need to be defined

Software Partition

A software partition is an organization of several software elements, which can generally be controlled and protected by the functions of an operating system and respective hardware features. It guarantees the freedom from interference between the included software elements and others belonging to a different partition, as memory accesses (code and data) are protected from other partition access.

Timing supervision/protection

A timing fault supervision/protection monitors/defines the correctness of execution of a piece of software with respect to timing, to detect or to prevent deadlock or livelock of software execution, to ensure correct allocation of time execution or synchronization of software elements. The timing supervision/protection can be realized by services in the operating system or in the infrastructure by defining bounds for timing execution of software (e.g. this can be as a watchdog for aliveness, or at task level or fine grain piece of code for deadlock or execution time).

Memory and Exchange

A memory fault protection mechanism prevents from corruption of contents or read/write data and code incorrect access between software elements of different partitions. The exchange protection ensures correct communication exchange between a sender and a receiver, to prevent repetition, loss, delay or incorrect change in the information or sequence of information.

Infrastructure

An AUTOSAR infrastructure defines the part of the software organization of the AUTOSAR platform which is below the application layer. In AUTOSAR this is identical to the technical implementation of the Virtual Functional Bus (VFB) on a concrete ECU. The technical implementation of the VFB, together with eventually necessary complex device drivers (CDD) defines the software infrastructure. It includes Run Time Environment (RTE), to define and realize communication between application SW components and all modules of a Basic Software which e. g. provides access to hardware resources.

4.2 Structure of the document

Section 5 provides an overview on the parts of ISO 26262, which are relevant for the software partitioning and for achieving independence and non-interference, is given.

Section **Erreur ! Source du renvoi introuvable.** provides an explanation of the initial statement for multi-class systems, assuming an independent software partition according to ISO 26262. To do this, in a first step a precise definition and impact of software partition with independence and non-interference is given (6.1). In addition to this, the software and hardware critical resources involved in software partition are identified (6.2), and relevant AUTOSAR documents associated to resources are elicited (6.2).

Section 7 deals with changes required in AUTOSAR to support the concept of multi-class systems. The current version of AUTOSAR in particular with software partition is described and limitations are documented (7.1). The actual AUTOSAR safety architecture will be challenged to propose an improved concept, capable for multi-class and efficient safety control as depicted in (7.2). Typical design patterns as new requirements for AUTOSAR are proposed in (7.3).

Finally conclusions and a discussion are given in section 8.

5 Overview on ISO 26262

This section provides an overview on the relevant parts and safety process steps of the ISO 26262, which are related to safety partitions for guaranteeing the independence and non-interference between the partitions. The selection of the presented parts is based on the source of the SAFE requirements which were elicited in WT 2.1 and allocated to WT3.3.2.

Addressing the development process of electric / electronic components for passenger cars, the ISO 26262 “Road vehicles – Functional safety” came into effect in November 2011. This standard introduces a safety lifecycle which “encompasses the principal safety activities during the concept phase, product development, production, operation, service and decommissioning” ([1], part 2, p.3) and which can be seen as a guideline that demands a risk-based development approach with seamless traceability. In Figure 1 an overview on the different parts of ISO 26262 is given.

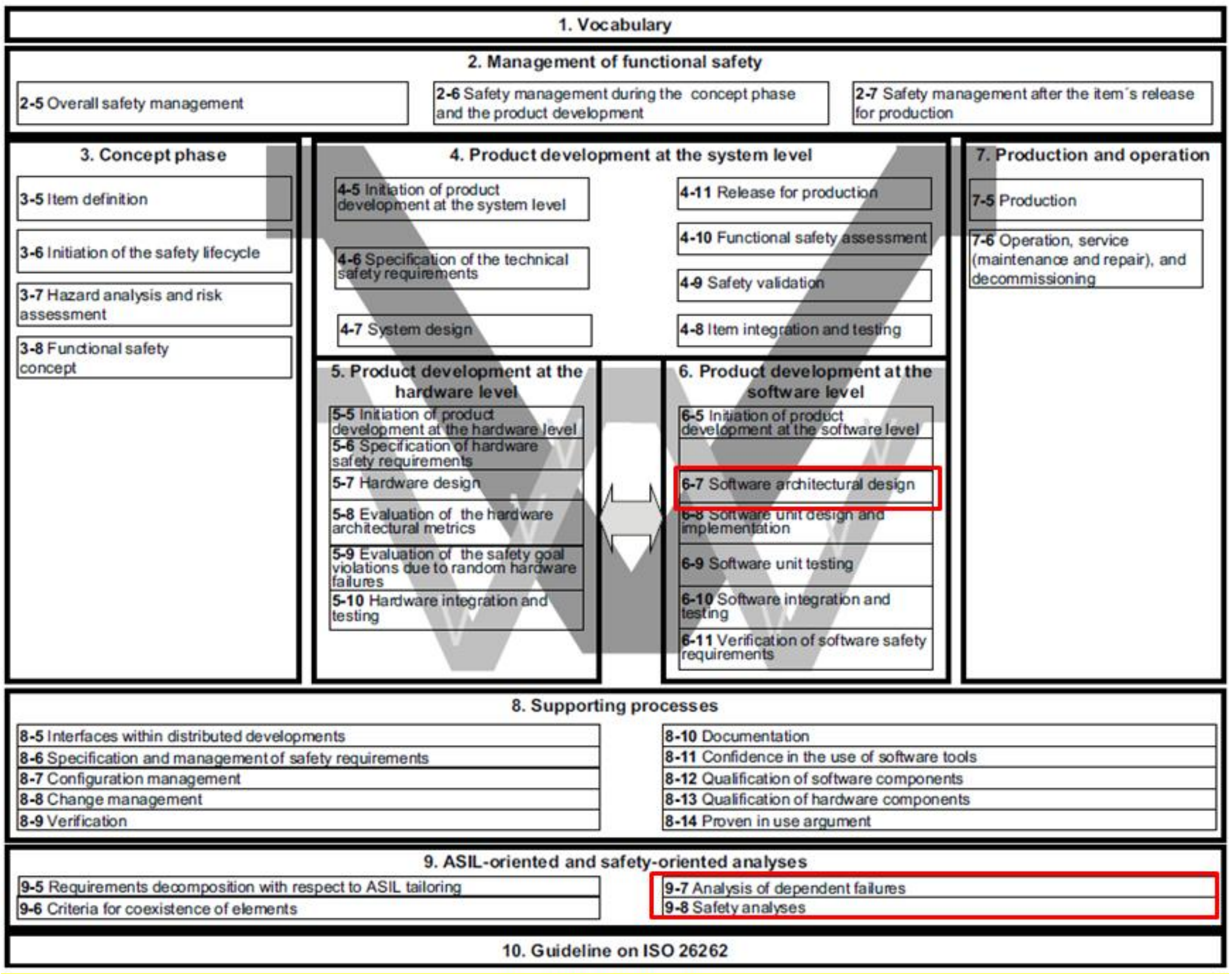
1. Vocabulary
2. Management of functional safety
2-5 Overall safety management
2-6 Safety management during the concept phase and the product development
2-7 Safety management after the item´s release for production
3. Concept phase
3-5 Item definition
3-6 Initiation of the safety lifecycle
3-7 Hazard analysis and risk assessment
3-8 Functional safety concept
4. Product development at the system level
4-5 Initiation of product development at the system level
4-6 Specification of the technical safety requirements
4-7 System design
4-11 Release for production
4-10 Functional safety assessment
4-9 Safety validation
4-8 Item integration and testing
7. Production and operation
7-5 Production
7-6 Operation, service (maintenance and repair), and decommissioning
5. Product development at the hardware level
5-5 Initiation of product development at the hardware level
5-6 Specification of hardware safety requirements
5-7 Hardware design
5-8 Evaluation of the hardware architectural metrics
5-9 Evaluation of the safety goal violations due to random hardware failures
5-10 Hardware integration and testing
6. Product development at the software level
6-5 Initiation of product development at the software level
6-6 Specification of software safety requirements
6-7 Software architectural design
6-8 Software unit design and implementation
6-9 Software unit testing
6-10 Software integration and testing
6-11 Verification of software safety requirements
8. Supporting processes
8-5 Interfaces within distributed developments
8-6 Specification and management of safety requirements
8-7 Configuration management
8-8 Change management
8-9 Verification
8-10 Documentation
8-11 Confidence in the use of software tools
8-12 Qualification of software components
8-13 Qualification of hardware components
8-14 Proven in use argument
9. ASIL-oriented and safety-oriented analyses
9-5 Requirements decomposition with respect to ASIL tailoring
9-6 Criteria for coexistence of elements
9-7 Analysis of dependent failures
9-8 Safety analyses
10. Guideline on ISO 26262

Figure 1: Overview on ISO 26262 (Relevant parts highlighted)

The requirements relevant for the safety partition to guarantee the independence and non-interference of software partitions and for safety activities and methods in software development are provided in ISO 26262:2011, Part 6 (in particular in clause 7 which focuses on “Software architectural Design”. However, also in Part 9 (Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses) requirements are given which affect directly or indirectly the safety partitioning by the demonstration of dependent failure and safety analysis. In the following, an overview on the relevant aspects from the respective parts is given.

Part 6: Product Development – Software Level

During this phase the development of the item from the software perspective is performed. Similar to system development and hardware development the software development process is assumed to be based on a V-model, starting with the specification of software safety requirements, proceeding with the software architectural design to guarantee freedom from interference between software elements and following with implementation.

Software Architectural Design

Structure of software elements and software safety mechanisms has to be defined to achieve freedom from interference between software partitions. The software architecture must be designed to effectively prevent from fault propagation between the partitions, to guarantee timing and to ensure protection of memory and information exchange. Additionally required are sufficiently safe services from the AUTOSAR infrastructure and measures to prevent from common cause failures.

Part 9: Automotive Safety Integrity Level (ASIL)-oriented and Safety-oriented Analyses

The relevant requirements for WT3.3.2 arise from two sections of [1], Part 9 “Automotive safety integrity level (ASIL) -oriented and safety-oriented analyses”, namely from chapter 7 “Analysis of dependent failures” and from chapter 8 “Safety analyses”.

Analysis of Dependent Failures

The standard provides “rules and guidance for the decomposition of safety requirements into redundant safety requirements” ([1], Part 9). This allows an “ASIL tailoring at the next level of detail” ([1], Part 9).

A common cause failure and cascading failure analysis shall be performed for the architecture, considering the operational life of the product. The analysis of dependent failures shall consider the architectural design which is supposed to consist of partitioned software elements. This induces on the architecture that specific measures are introduced and applied to architecture elements, such as redundancy, dissimilar development, safety mechanisms and physical barriers.

Safety Analyses

The safety analyses shall discover consequences of faults and failures on functions, behaviour and design of items and elements. Fault propagation in the context of software partitions and related hardware elements are targeted in this task. Moreover, the analysis provides information on causes and conditions that could lead to the violation of a safety goal or safety requirement. It also provides argumentation for the introduction of additional safety requirements. This can be achieved by an implementation of safety mechanisms to mitigate failure propagation and to ensure freedom from interference.

6 State of the art of AUTOSAR mixed criticality ECU systems

After looking at the relevant parts of ISO 26262 which are dedicated to independence and non-interference for software partitions and safety measures, it seems consequent to analyse more detailed the main factors influencing the deployment of a multi-class ECU system being based on an AUTOSAR infrastructure. A precise definition of independence and non-interference shall be given. The primary analysis shall explore the critical ECU resources and existing AUTOSAR specifications within this context. In chapter 7, state of the art solutions being based on AUTOSAR will be documented.

6.1 Independence and non-interference

The independence of elements guarantees the absence of dependent failures between two or more elements and excludes common cause and cascading failures between the elements. Without the independence failures might propagate through the elements and lead to the violation of a safety goal. For cascading failures this holds in bidirectional relation. The demonstration of freedom from interference shall exhibit the absence of cascading failures between two or more elements which might lead to the violation of a safety requirement.

Each software partition embedding the software elements of different integrity levels (from QM to A, B, C and D) shall be protected by safety mechanisms to guarantee the independence and non-interference between the associated software applications. The AUTOSAR infrastructure shall include safety mechanisms, either implemented by hardware or by software, to:

- Prevent timing violation from any lower integrity class to any higher integrity class
- Protect against timing violation for integration of object code in each integrity class
- Prevent data violation from any lower integrity class to any higher integrity class
- Protect against data violation for integration of object code in each integrity class
- Allow separate allocation of safety-objects per integrity class

6.2 Critical resources

This chapter lists software and hardware critical resources which need to be controlled in order to ensure the Freedom from Interference in a multi-class ECU. The resources are grouped according to the main issues “Power Supplies”, “Timing and Execution / Memory” and “Communication” which, except the Power Supplies, are also considered in the informative Annex D of ISO 26262-6.

Power Supplies have been added since independence is stronger affected by the supplies in current microcontroller technologies than in the past. Furthermore, execution and memory are strongly related to one another, and are therefore considered together.

Several resources realize safety measures and are inherently safety relevant, these include

- Hardware test, supervision management and protection units
  - ECC Unit
  - internal / external Watchdog
  - Memory Management / Protection Unit
  - …
- Software test measures for hardware resources
  - RAM test, ROM test, Stack test,

  - Interrupt Controller test,
  - Watchdog stack
  - …
- Software test measures for communication
- E2E-Lib / Protection Wrapper
- CRC
- Timeout Monitoring
- …

Critical Power Supply resources are the supplies of the safety-relevant HW resources.

Critical Hardware resources related to Timing and Execution / Memory are
- PLL and OS timer resources
- Blocks of the Core (ALU, FPU, Coprocessor, Registers internal bus, Crossbar, Bridge…)
- Interrupt Controller and DMA
- RAM, Flash Memory, EEPROM (emulated)
- MMU/MPU region descriptor registers
- ADC for analogue sensor signals

Critical Software resources related to Timing and Execution / Memory are
- OS (protection mechanisms)
- Memory Stack (Flash, FEE)
- State Managers (ECU, Com, Bus, Boot, BSW, APP)
- FIM (safety-relevant error reactions)
- MCAL (modules supporting safety mechanisms)
- IO HW Abstraction (safety relevant IO)
- Complex Device Drivers (if dedicated to safety-relevant functions)

Critical Hardware resources related to Communication are
- Communication Busses (Flexray, CAN, LIN, Ethernet, …)
- Serial external Communication Busses (SPI, I^2C, …)
- Communication Controllers and dedicated PLL, Transceivers

Critical Software resources related to Communication are
- RTE (calls of basic software services, checks of execution timing…)
- Inter Core Communication (only in multicore controllers)
- Com-Stacks (exchange of safety-relevant data)

6.3 AUTOSAR Document Reference

The AUTOSAR infrastructure specification provides means for the control of the above listed critical resources. The specifications will be listed first and then be analyzed in the next section to understand the actual safety mechanisms that are already present in the AUTOSAR standard. Documents related to the respective critical resources are (SWS: software specification; SRS: software requirements specification):

- AUTOSAR_SWS_OS for OS services, Memory Protection, Timing Protection, Multi-core
- AUTOSAR_SW_SynchronizedTimeBaseManager for synchronization of OS time with external timing
- AUTOSAR_SWS_RTE for RTE services and link to E2E services
- AUTOSAR_SRS_ModeManagement for link to RTE services
- AUTOSAR_SWS_WatchdogManager for watchdog services
- AUTOSAR_SWS_ECUStateManager and AUTOSAR_SWS_ECUStateManagerFixed for ECU state Manager, Initialize and reinitialize OS, SchM, BswM, configure ECU for sleep and shutdown, manage wakeup events
- AUTOSAR_SWS_BSWModeManager for Arbitrate mode requests from application layer SW-Cs or other BSW modules
- AUTOSAR_SWS_CoreTest for Core Test
- AUTOSAR_SWS_RAMTest for RAM test
- AUTOSAR_SWS_E2ELibrary as E2E library and link to communication (RAM or COM)
- AUTOSAR_SWS_FlashTest for flash test
- AUTOSAR_SWS_COM and AUTOSAR_SWS_COMManager for control of the communication stack
- AUTOSAR_EXP_ErrorDescription for CAN communication and Memory Stack error detection and recovery
- AUTOSAR_EXP_ApplicationLevelErrorHandling for application specific error detection and recovery with definition safety mechanism
- AUTOSAR_SWS_FunctionInhibitionManager for survey application level error handling by specific mechanism pattern and definition of error reporting and propagation of inhibition of function.

In addition, AUTOSAR documented the use of safety mechanisms in the following safety-related documents

- Program Flow Monitoring Related Features (from AUTOSAR_SWS_CoreTest) in AUTOSAR_TR_SafetyConceptStatusReport
- Timing Related Features (from AUTOSAR_SWS_OS and AUTOSAR_SWS_WatchdogManager) in AUTOSAR_TR_SafetyConceptStatusReport
- Communication Stack Related Features (from AUTOSAR_SWS_COM) in AUTOSAR_TR_SafetyConceptStatusReport
- End-to-End Communication Protection Related Features (from AUTOSAR_SWS_E2ELibrary) in AUTOSAR_TR_SafetyConceptStatusReport

- Memory Partitioning and User/Supervisor-Modes Related Features (from AUTOSAR_SWS_OS) in AUTOSAR_TR_SafetyConceptStatusReport
- BSW Trusted partition using mechanism OS AUTOSAR capabilities (from AUTOSAR_SWS_OS) in AUTOSAR- internal documents on BSW partitioning.

The analysis of AUTOSAR documents with respect to technical gaps regarding independence and non-interference for multi-class ECU's was performed according to the list of topics that can be taken from the left column of the following table. The table gives a summary of the performed analysis and describes possible safety mechanisms and some hints in the right column. They may be considered by the AUTOSAR standard for future extensions. Note that all state of the art techniques as e. g. defensive techniques have to be applied.

Solution for independent software partition	Validation of the software partition independence at start up (minimum of HW test).
Solution for independent resources associated to partitions	OS timer HW resource correctness.
Possibility to create partition via AUTOSAR service	Proof possibility to define service.
Possibility to protect partition from HW failure propagation	HW limitation for reliable detection and protection (block of µC connected to core, single core …). Limitation of data access by HW configuration, no add-on by SW. For fine grain split using SW mechanism.
Possibility to protect partition from software failure	Use of Mode (from Mode Manager) not defined as process related.
Possibility to protect partition from wrong configuration	No definition of configuration parameters that are safety relevant.
Possibility to protect partition from AUTOSAR service failure	Case by case integrated in RTE (resource issue). How to give a guarantee of failure coverage and error code propagation.
Possibility to manage partition w.r.t. active or passive redundancy	How to recommend usage of partition shut-down based on criteria.
Possibility to detect common root cause of failures between partitions	Supply and clock electronic design (external/internal supply or clock supervision, overheating protection).
Possibility to detect cascading failures between partitions	Measures to detect cascading failures.
Possibility to ensure correctness of resource access	Proof possibility to ensure correctness of resource access.
Timing protection w.r.t. resource access	Proof possibility of timing protection w.r.t. resource access.
ISR1 & MCAL/HW protection	Reliability of all safety peripheral in BIST or BSW self-tests (today only data and flash). HW Resource Manager in definition. Definition and coordination of safe state for ECU and BSW manager.
AUTOSAR services protection	Proof possibility of services protection.

Summary of AUTOSAR Gap Analysis

7 Architecture analysis for AUTOSAR mixed criticality ECU systems

Within this section the current status of the AUTOSAR architecture proposal for mixed criticality is shown and discussed. Then proposals for new AUTOSAR architecture extensions are given, in particular the MicroSAR TTTECH architecture considered in the Safe-e project [2, and the enhanced version proposed in the SAFE project].

7.1 Current status of AUTOSAR

Nowadays, the AUTOSAR concept supports mixed ASIL in the same ECU, as depicted in the Figure 2. It is based on the concept of trusted BSW (basic software), as defined by a trusted verification and validation (or from reuse), stating that no risk for failure propagation or introduction of error is guaranteed. The basic software and RTE (runtime environment) are completely trusted, where the plug of the ASIL-x application and independence are guaranteed by the trusted concept.

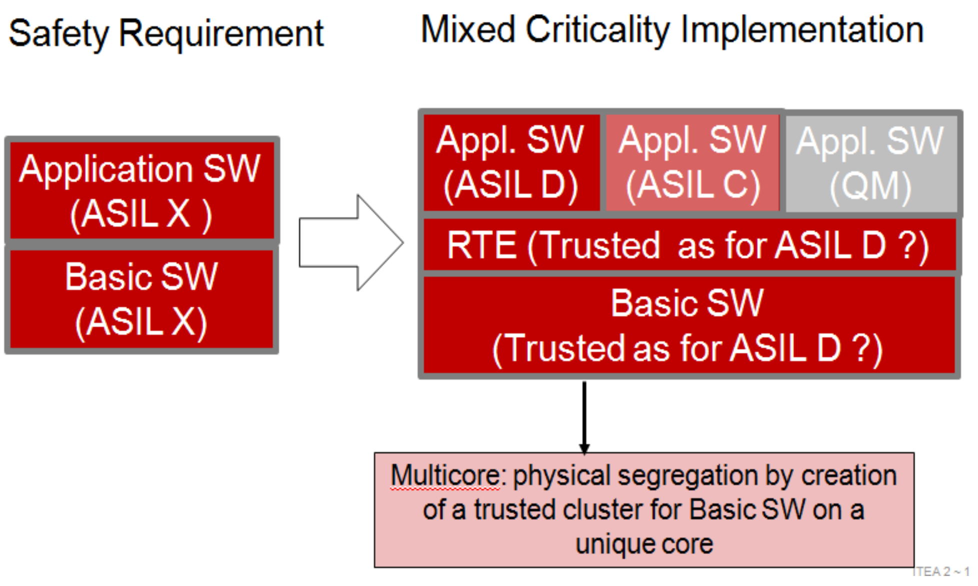
Safety Requirement
Mixed Criticality Implementation
Application SW (ASIL X)
Basic SW (ASIL X)
Appl. SW (ASIL D)
Appl. SW (ASIL C)
Appl. SW (QM)
RTE (Trusted as for ASIL D ?)
Basic SW (Trusted as for ASIL D ?)
Multicore: physical segregation by creation of a trusted cluster for Basic SW on a unique core
ITEA 2 ~ 1

Figure 2: AUTOSAR Safety Concept

The architecture raises several questions, such as:

– How to guarantee such trusted status and to ensure flexibility in AUTOSAR configuration?

– How to manage an upgrade of the platform?

– How would the cost situation look like for the complete trusted solution?

– Shall all applications pay a high price?

AUTOSAR favours the concept of software partitioning by introducing software partitions to be separated by the AUTOSAR OS. This implies to define SW-Components according to their safety classification. It opens an additional dimension in dealing with the OS since the OS in the past was widely used to perform the schedule of several tasks. The problems of the concept go along with e. g. needs for decompositions, including the IO signal paths realized within the basic software and system dependent needs to mitigate common causes.

For ASIL D e. g. diverse software design as a mechanism for error detection at the software architectural level is highly recommended. Realizing every safety relevant function accordingly implies high ECU resources consumption, but resources are always subject to limitations. There are also other technical facts which may limit certain IO signals to maximum ASIL A because e. g. a specific sensor cannot be more reliable due to its failure characteristics.

For typical IO-intensive automotive systems it is sufficient to detect critical errors and react within fault latency time in a way that preserves the safe state for the system. Such safety behaviour is classified as fail-silent, in contrary to fail-operational. In that case there are alternatives (like data redundancy) to at least mostly replace the possibilities of memory protection to achieve the safe operation of the system.

7.2 New proposed Architecture

The approach favoured by TTTECH in the MicroSAR architecture is extracted from a draft document version and sketched in Figure 3. The Basic Software is built according to standard ASIL decomposition rules and gives a safety partition as ASIL Max and a Basic SW as QM.

The safety partition is intended to separate components of different ASIL, and also QM components. The Safety partition uses barriers for protection against memory, timing, or data errors. The idea is based on “freedom from interference” as described in ISO 26262 Sec. 6, annex D. As long as freedom from interference can be guaranteed, QM components can be operated in parallel and in conjunction with safety-relevant ASIL components of a certain criticality. The solution proposed uses separated RTE’s. The safety-relevant part uses its own, protected RTE, which is reduced in some aspects.

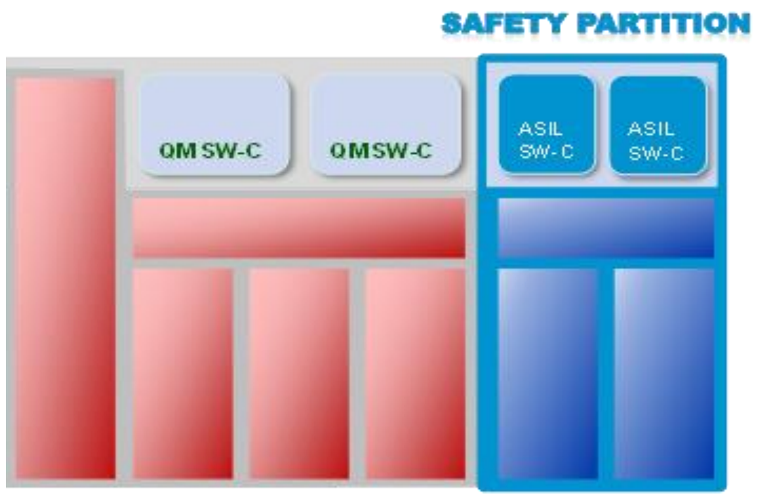

Figure 3: TTTECH Safety Concept based on partionining

The hardware is supposed to be trusted as the safety partition does not mask any failures of the hardware. Defensive coding is applied, and barriers for software error propagations are integrated. The following three resources with respect to possible faults are considered:

- Time (CPU time gap) detected by PFM (component deadline)
- Memory (overwriting) detected by MPU, OS SC3 service and Safety context interface
- I/O data com (Bus error) detected by E2E Library and wrapper protection

No level 1 interrupts are authorized, except it is guaranteed by the user that the interrupts don’t interfere with the sequence being interrupted. This seems difficult to be achieved, especially since strong timing requirements always enforce the use of this kind of interrupts.

The barriers are described as follows:

- Barrier against timing errors, based on the Safe watchdog concept with a watchdog manager supervising the collection of SW checkpoints, including timing checkpoints, and communicating with an external watchdog HW.

- Barrier against I/O errors, based on E2E protection and relying on standard E2E-Library as defined in AUTOSAR. Services are defined above the RTE to cover the complete communication path, including RTE and COM Stack.
- Barrier against memory errors, based on MPU and setting the task context accordingly. During task (or interrupt) switch the new MPU configuration is set and all relevant registers are checked (stack, PC …).

In case more than one ASIL level shall be supported in the target application, more than one safety-partition can be installed. Between two ASIL levels (ASIL-x and ASIL-y) in one system where more than one safety partition shall be used, freedom from interference between the safety partitions is guaranteed. Thus the safety measures can be scaled: QM building blocks will have low safety measures to protect their application part, lower ASIL level partitions will have more measures applicable and high ASIL partitions will be enabled to rely on the full amount of required safety measures as required by ISO 26262.

A new extended proposal is depicted in Figure 4. It provides an improvement of the proposed safety partitioning from the TTTECH safety concept and is explained further.

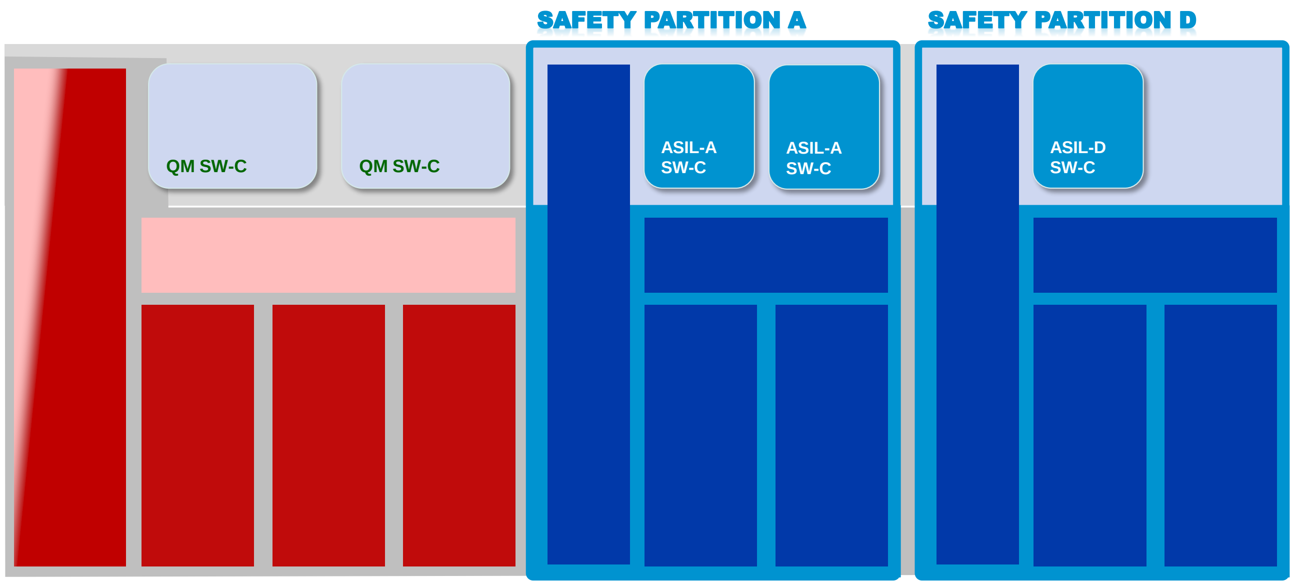

Figure 4: Extended Safety Concept

The main characteristics of the concept are:

- ASIL decomposition results in safety partition as ASIL Max and generic Basic SW as QM.
- The Safety partition is supervising the application as in the other concepts, but additionally also the BSW and RTE, all with respect to Control Flow and Data integrity. AUTOSAR today already allows several mechanisms for Control Flow and Data integrity (out of E2E) especially relevant for the data of the SW drivers of the HW abstraction layer.
- Mixed-criticality on application level is possible to be realized (the fact that this is not possible in the basic software shall be neglected here).
- Basic SW can be checked with the watchdog manager (Control flow or task context).
- The safety partition additionally detects HW Errors.

The potential limits of the architectures can be identified as:

- IO intensive systems are still not ideally supported. It is not easy possible to realise the basic software in different ASIL's, independent from the application, e. g. as a result of necessary decompositions within the basic software. This in fact raises the same questions as in 7.1 about technical feasibility and costs.

– Fail operational applications are currently not supported. The solution provided as of today will generally lead the system in a safe state even if caused by the QM part. This turns out to become a future development area.

The next chapter describes the approach to improve the AUTOSAR-based solutions. Especially the data integrity for I/O SW drivers may be improved and standardized which then may be a new requirement for AUTOSAR. The proposal is to focus on I/O standard drivers and to define basic mechanisms to check the data integrity.

7.3 New Requirements

With the safety concept as shown in Figure 4 and the AUTOSAR specification [4] including the definition of safety mechanisms, most of the needs for control flow checkers are covered. But data integrity requirements and in particular features required for standard Input//Output drivers are less covered. Therefore the focus will be on requirement collection and especially on **defining design patterns for checking data path integrity of signals passing I/O drivers, including potential failures of associated peripherals**. These design patterns, organized from AUTOSAR BSW specification [3] by selection of I/O drivers, will permit to generate new requirements and patterns for AUTOSAR applications.

The considered list of MCAL drivers is:

- ADC driver
- DIO driver
- Port Driver
- ICU driver
- OCU Driver
- PWM driver
- I/O HW abstraction

The following section will describe the related information necessary to identify these design patterns as safety mechanism for I/O signal integrity control of MCAL and HW IP elements. The analysis is performed in three steps. First the section 7.3.1 defines failure modes for the AUTOSAR MCAL driver and associated peripherals, second section 7.3.2 describes the actual state of the art of the safety mechanisms, and finally section 7.3.3 presents small uses cases of application of safety mechanism in regards to MCAL driver and Peripheral failures.

7.3.1 Failure Modes

The following list of failure modes of the SPAL drivers, which means the part of the MCAL that excludes ECU external communication (CAN-, LIN- and FR-Drivers are excluded), is based on AUTOSAR driver specification analysis. The associated failure mode of the IP hardware is non exhaustive because it is strongly related to IP peripheral and core controller architecture.

<u>ADC driver</u>

- ADC inputs signal failure modes:
    - Out of range
    - Oscillation
    - Offsets
    - Drifts

   - Short circuit to GND
   - Short circuit to VDD
   - Open load
   - Wrong Vref voltage
- ADC Internal peripheral Engine failure modes
   - ADC convertor locked
   - ADC convertor deviation
   - ADC register memory corrupted
   - ADC multiplexer failure
   - ADC Vref corrupted
   - ADC start wrong trigger
- ADC software driver failure modes
   - Wrong ADC channel input selection
   - Wrong clock configuration (frequency, pre-scaler…)
   - Wrong conversion time configuration
   - Wrong sampling time configuration
   - Wrong resolution configuration
   - Wrong ADC mode conversion selection (Single shot Vs Continuous)
   - Wrong ADC channel group selection
   - Wrong ADC initialization
   - Starting Wrong channel or group conversion
   - Collecting Wrong channel or group conversion results (e.g. Error in memory buff-ers)
   - Reporting Wrong channel or group conversion results (e.g. wrong notification)

DIO driver

- DIO input signals failure modes:
   - Short to VDD
   - Short to GND
   - Open load
- DIO Internal peripheral failure modes
   - I/O image register corrupted
   - Signal distortion caused by Power supply disturbance
- DIO software driver failure modes:
   - Wrong DIO configuration (direction, levels…)
   - writing wrong channel or port
   - reading wrong channel or port

Port Driver

- Port output/input signals failure modes:
   - Short to VDD
   - Short to GND
   - Open Load
- Port Internal peripheral failure modes
   - Input stage failure
   - Output stage failure

  - PAD pull up failure
  - Port register control memory corrupted
  - Multiplexer port failure
  - Control port failure
- Port software driver failure modes:
  - Wrong pin configuration
  - Wrong external trigger pin configuration
  - Wrong pin direction setting

ICU driver

- ICU input signals failure modes
  - Short to VDD
  - Short to GND
  - Open Load
  - Frequency drift
- ICU Internal peripheral failure modes
  - Timer block
  - Frequency error
  - Control register memory corrupted
  - Control capture cells failure
- ICU software driver failure modes:
  - Wrong Clock or pre-scaler configuration
  - Wrong Input capture interrupt configuration and handling
  - Wrong port pin configuration
  - Wrong ECU Mode Wakeup notification
  - Wrong ICU configuration
  - Wrong ICU mode management (Wakeup, sleep…)
  - Wrong ICU notification reporting
  - Wrong ICU activation conditions (Rising/Falling Edges…)
  - Wrong ICU time stamps counting
  - Wrong ICU signal management

OCU Driver

- OCU input signals failure modes
  - Short to VDD
  - Short to GND
  - Open Load
  - Frequency drift
- OCU Internal peripheral failure modes:
  - Timer block
  - Frequency error
  - Control register memory corrupted
  - Control capture cells failure
- OCU software driver failure modes:
  - Wrong Clock or pre-scaler configuration

  - Wrong Input capture interrupt configuration and handling
  - Wrong port pin configuration
  - Wrong ECU Mode Wakeup notification
  - Wrong OCU configuration
  - Wrong OCU notification reporting
  - Wrong OCU activation conditions (Rising/Falling Edges…)
  - Wrong OCU time stamps counting
  - Wrong OCU signal management

PWM driver

- PWM input signals failure modes
  - Short to VDD
  - Short to GND
  - Open Load
  - Frequency drift
- PWM Internal peripheral failure modes
  - Timer block
  - Frequency error
  - Control register memory corrupted
  - Control capture cells failure
- PWM software driver failure modes:
  - Wrong PWM duty cycle
  - Wrong PWM period
  - Wrong Timer calculation

I/O HW abstraction

- Input/output signals failure modes
  - All signal failures related to I/O pins
- Internal peripheral failure modes
  - All failure modes related to hardware IP (cannot be exhaustive)
- I/O HW software abstraction failure modes:
  - I/O set to incorrect values
  - Getting wrong peripheral notifications (e.g. Adc/Dio/Pwm)
  - Wrong I/O HwAbs valid range for /I/O
  - Wrong I/O HwAbs peripherals status on HW failures
    - Short to GND
    - Short to VDD
    - Open Load

7.3.2 Design pattern

The state of the art safety mechanisms are listed below and can be used as design capable to mitigate failure modes identified in the above section

Safety Mechanisms:

- HW built-in mechanisms:
  - Signal, Channel and Port redundancy
  - HW test patterns
  - HW diagnostics

- SW Safety mechanisms:
  - For Error detection
    - Plausibility checks
    - Majority voting (2 out of 3)
    - Range checks of inputs and outputs
    - SW diagnostic routines
  - For Error handling
    - Replacement values (NVRAM, FLASH)
    - Configuration reset
    - Graceful degradation
    - …

7.3.3 Use case definition

The uses cases for the application of safety mechanisms regarding MCAL driver and Peripheral failures are classified for selected AUTOSAR driver specification. They are detailed below.

In the below table the Safety mechanism in Bold are already present in AUTOSAR.

Analog Input

Module	Safety Mechanism	Failure Mode (FM) covered
ADC driver		
Port Driver		
…		

The I/O signal integrity check of an analog acquisition from end to end signal control, from HW pin to control of the application is defined by:

- IOSigIntegChk1 …
- IOSigIntegChkX

or to be covered by the application which uses safety mechanisms as :

- AppSM1 …
- AppSMY

Analogue Input/Output : ADC driver

Module	Safety Mechanism(s)	FM covered
ADC Driver		**ADC inputs signal failure modes**
	Range checks	Out of range
	Compare with independent signal	Oscillation
	Compare with independent signal	Offsets
	Compare with independent signal	Drifts
	Compare value with thresholds	Short circuit to GND
	Compare value with thresholds	Short circuit to VDD
	Compare with independent signal Precharge-ADC-Test (HW)	Open Load
	Monitor defined static test signal(s) Compare with independent ADC channel using independent reference Voltage	Wrong Vref voltage
		ADC Internal peripheral Engine failure modes
	Monitor defined dynamic test signal(s) Testpattern in RAM Compare with independent ADC channel	ADC convertor locked
	Monitor defined dynamic test signal(s) Compare with independent ADC channel	ADC convertor deviation
	ECC (HW) Monitor defined dynamic test signal(s) Readback register and compare Compare with independent ADC channel	ADC register memory corrupted
	Monitor defined dynamic test signal(s) Compare with independent ADC channel	ADC multiplexer failure
	Monitor defined static/dynamic test signal(s) Compare with independent ADC channel using independent reference Voltage	ADC Vref corrupted
	Independent timestamps for ADC samples and check	ADC start wrong trigger
		ADC software driver failure modes
	Monitor defined static/dynamic test signal(s) Compare with independent ADC channel	Wrong ADC channel input selection

	Independent timestamps for ADC samples and check	Wrong clock configuration (frequency, pre-scaler…)
	Readback register and compare	Wrong conversion time configuration
	Readback register and compare	Wrong sampling time configuration
	Readback register and compare	Wrong resolution configuration
	Readback register and compare Independent timestamps for ADC samples and check Compare with independent ADC channel	Wrong ADC mode conversion selection (Single shot Vs Continuous)
	Monitor defined static/dynamic test signal(s) with test signals using different PINs at different ADC groups Compare with independent ADC channel	Wrong ADC channel group selection
	Readback register and compare	Wrong ADC initialization
	Monitor defined dynamic test signal(s) Independent timestamps for ADC samples and check Compare with independent ADC channel	Starting Wrong channel or group conversion
	Monitor defined static/dynamic test signal(s) Compare with independent ADC channel	Collecting Wrong channel or group conversion results (e.g Error in memory buffers)
	Monitor defined static/dynamic test signal(s) Compare with independent ADC channel	Reporting Wrong channel or group conversion results (e.g. wrong notification)

The I/O signal integrity check of an analog signal from end to end signal control, from application to HW pin is protected by:

- Channel redundancies and plausibility checks at application level
- Complementary data redundancy / Memory protection on register level
- WdgM Program Flow Monitoring

Digital Input/Output : DIO driver (Assumption: static high/low)

Module	Safety Mechanism(s)	FM covered
DIO Driver		**DIO input signals failure modes**
	Output: Monitor output signal(s), Testpulse Input: Compare with independent signal, diagnostic voltage range	Short to VDD
	Output: Monitor output signal(s), Testpulse Input: Compare with independent signal diagnostic voltage range	Short to GND
	Output: Monitor load current, etc. Input: Compare with independent signal, diagnostic voltage range	Open Load
		DIO Internal peripheral failure modes
	Output: Monitor output signal(s) , Testpulse, Readback register and compare Input: Compare with independent signal, diagnostic voltage range	I/O image register corrupted
	Voltage monitoring (HW)	Signal distortion caused by Power supply disturbance
		DIO software driver failure modes
	Readback register and compare	Wrong DIO configuration (direction, levels…)
	Monitor output signal(s)	writing wrong channel or port
	Compare with independent signal, diagnostic voltage range	reading wrong channel or port

The I/O signal integrity check of a digital IO signal from end to end signal control, from application to HW pin is protected by:

- Channel redundancies and plausibility checks at application level
- Complementary data redundancy / Memory protection on register level
- WdgM Program Flow Monitoring

μC Port: Port Driver

Module	Safety Mechanism(s)	FM covered
Port Driver		**Port output/input signals failure modes**
	Output: Monitor output signal(s), Testpulse Input: Compare with independent signal, diagnostic voltage range	Short to VDD
	Output: Monitor output signal(s), Testpulse Input: Compare with independent signal diagnostic voltage range	Short to GND
	Output: Monitor load current, etc. Input: Compare with independent signal, diagnostic voltage range	Open Load
		Port Internal peripheral failure modes
	Compare with independent signal, diagnostic voltage range	Input stage failure
	Monitor output signal(s), Testpulse	Output stage failure
	Specific testscenario	PAD pull up failure
	Readback register and compare	Port register control memory corrupted
	Output: Monitor output signal(s), Testpulse Input: Compare with independent signal diagnostic voltage range	Multiplexer port failure
	Output: Monitor output signal(s), Testpulse Input: Compare with independent signal diagnostic voltage range	Control port failure
		Port software driver failure modes
	Readback register and compare	Wrong pin configuration
	Readback register and compare	Wrong external trigger pin configuration
	Readback register and compare	Wrong pin direction setting

The I/O signal integrity check of a port signal from end to end signal control, from application to HW pin is protected by:

- Channel redundancies and plausibility checks at application level
- Complementary data redundancy / Memory protection on register level
- WdgM Program Flow Monitoring

Input Capture: ICU driver

Module	Safety Mechanism(s)	FM covered
ICU driver		**ICU input signals failure modes**
	HW/SW Input signal diagnostic Plausibility Check	Short to VDD
	HW/SW Input signal diagnostic Plausibility Check	Short to GND
	HW/SW Input signal diagnostic Plausibility Check	Open Load
	HW/SW Frequency diagnostic Plausibility Check	Frequency drift
		ICU Internal peripheral failure modes
	HW Timer diagnostic checks	Timer block
	HW Frequency diagnostic checks Plausibility Check	Frequency error
	Peripheral Access Protection: CPU supervisor Mode MPU protection	Control register memory corrupted
	ECC (Random HW faults) + HW di-agnostic	Control capture cells failure
		ICU software driver failure modes:
	Range checks	Wrong Clock or pre-scaler configuration
	Range + Plausibility checks	Wrong Input capture interrupt configuration and handling
	CRC-Protected Configuration	Wrong port pin configuration
	Plausibility Checks	Wrong ECU Mode Wakeup notification
	CRC-Protected Configuration	Wrong ICU configuration
	Plausibility Checks	Wrong ICU mode management (Wakeup, sleep…)
	Range Checks (Configura-tion/Parameters)	Wrong ICU notification reporting
	Range Checks (Configura-tion/Parameters)	Wrong ICU activation conditions (Ris-ing/Falling Edges…)
	Plausibility Checks + CRC Protected Data	Wrong ICU time stamps counting
	Control Flow Monitoring	Wrong ICU signal management

The I/O signal integrity check of an input capture acquisition from end to end signal control, from HW pin to control of the application is protected by:

- CRC + Counter
- Redundancy (duplicated of the input signal processing)
- Plausibility checks at application level
- WdgM Program Flow Monitoring

Output Compare: OCU Driver

Module	Safety Mechanism(s)	FM covered
OCU Driver		**OCU input signals failure modes**
	HW/SW Output signal diagnostic Plausibility Check	Short to VDD
	HW/SW Output signal diagnostic Plausibility Check	Short to GND
	HW/SW Output signal diagnostic Plausibility Check	Open Load
	HW/SW Frequency diagnostic Plausibility Check	Frequency drift
		OCU Internal peripheral failure modes:
	HW Timer diagnostic checks	Timer block
	HW Frequency diagnostic checks Plausibility Check	Frequency error
	Peripheral Access Protection: CPU supervisor Mode MPU protection	Control register memory corrupted
	ECC (Random HW faults) + HW diagnostic	Control capture cells failure
		OCU software driver failure modes:
	Range checks	Wrong Clock or pre-scaler configuration
	Range + Plausibility checks	Wrong Input capture interrupt configuration and handling
	CRC-Protected Configuration	Wrong port pin configuration
	Plausibility Checks	Wrong Ecu Mode Wakeup notification
	CRC-Protected Configuration	Wrong OCU configuration
	Range Checks (Configuration/Parameters)	Wrong OCU notification reporting
	Range Checks (Configuration/Parameters)	Wrong OCU activation conditions (Rising/Falling Edges…)
	Plausibility Checks + CRC Protected Data	Wrong OCU time stamps counting
	Control Flow Monitoring	Wrong OCU signal management

The I/O signal integrity check of an output capture from end to end signal control, from application to HW pin is protected by:

- CRC + Counter
- Plausibility checks at application level
- WdgM Program Flow Monitoring

PWM Output: PWM Driver

Module	Safety Mechanism(s)	FM covered
PWM Driver		**PWM input signals failure modes**
	HW/SW Input signal diagnostic Plausibility Check	Short to VDD
	HW/SW Input signal diagnostic Plausibility Check	Short to GND
	HW/SW Input signal diagnostic Plausibility Check	Open Load
	HW/SW Frequency diagnostic Plausibility Check	Frequency drift
		PWM Internal peripheral failure modes
	HW Timer block diagnostic checks	Timer block
	HW Frequency diagnostic checks Plausibility Check	Frequency error
	Peripheral Access Protection: CPU supervisor Mode MPU protection	Control register memory corrupted
	ECC (Random HW faults) + HW diagnostic	Control capture cells failure
		PWM software driver failure modes:
	Range + Plausibiliy Checks	Wrong PWM duty cycle
	Range + Plausibiliy Checks	Wrong PWM period
	Plausibility Checks + CRC Pro-tected Data	Wrong Timer calculation

The I/O signal integrity check of a PWM signal from end to end signal control, from application to HW pin is protected by:

- CRC + Counter
- Plausibility checks at application level
- WdgM Program Flow Monitoring

IO Hardware Abstraction: I/O Hw Abstraction

Module	Safety Mechanism(s)	FM covered
I/O Hw Abstraction		**Input/output signals failure modes**
	Range + Plausibility checks	All signal failures related to I/O pins
		Internal peripheral failure modes
	Range + Plausibility + external status checks	All failure modes related to hardware IP (cannot be exhaustive)
		I/O HW SW abstraction failure modes
	Channel redundancies and plausi-bility checks at application level, Complementary data redundancy, Memory protection, Range checks, Plausibility checks	I/O set to incorrect values
	Channel redundancies and plausi-bility checks at application level, Complementary data redundancy, Memory protection, Range checks, Plausibility checks	Getting wrong peripheral notifications (e.g Adc/Dio/Pwm)
	Channel redundancies and plausi-bility checks at application level, Complementary data redundancy, Memory protection, Range checks, Plausibility checks	Wrong I/O HwAbs peripherals status on hw failure “Short to GND”
	Channel redundancies and plausi-bility checks at application level, Complementary data redundancy, Memory protection, Range checks, Plausibility checks	Wrong I/O HwAbs peripherals status on hw failure “Short to VDD”
	Channel redundancies and plausi-bility checks at application level, Complementary data redundancy, Memory protection, Range checks, Plausibility checks	Wrong I/O HwAbs peripherals status on hw failure “Open Load”

8 Conclusions and Outlook

This document provides architecture and gap analysis with respect to AUTOSAR specifications 4.0.3. It provides use cases in terms of low level safety measures to be used in I/O intensive systems.

The actual AUTOSAR basic services allow for controlling the flow and data integrity, but the actual I/O drivers and data integrity are less covered in the AUTOSAR specification, including also the integrity with respect to HW errors. Therefore it seems necessary that AUTOSAR requirements, architecture and specifications are to be improved with respect to such requirements.

The E2E protection can in principle cover local communication, but it is not optimized for it. The protection of bus information above the RTE has disadvantages if the protected data units are not identical with the signals provided by COM, which unfortunately is quite often the case. The reason for this is that the protected data units then must be reconstructed on top of the RTE and signals have to be resynchronized on application level. This implies a waste of resources, which increases with increasing number of safety relevant signals as part of the communication matrix or implies use of the alternative callout solution, which is possible but not defined in detail..

Current ideas for potential follow-on developments beyond the project scope aim at implementing safety partitions, which are not only well protected but can also run autonomously (independent from each other). In order to support such approach, some of the services need to be constructed in a way that they can “survive” and remain fully disposable despite other parts/functions of the system are terminated and closed down. For example, this can refer to OS/task scheduling, persistent memory or I/Os. Multi-core based systems will support this in a very promising manner. Referring to AUTOSAR, all cores are supposed to start in a synchronized manner, which currently contradicts the approach. Nevertheless, such approach needs to be further discussed despite the conflict with AUTOSAR requirements. Thus this is an issue to be solved in later developments beyond the SAFE project in order to make the solution viable for highly safety-relevant applications such as in ADAS or autonomous driving.

9 References

[1] International Organization for Standardization: ISO 26262 Road vehicles - Functional safety. (2011)

[2] Safe-E Eurostars project, D2.1 and D2.2 deliverable D2_1_Eurostars-E6095-Safe-E-Mixed_Criticality_Platform_Approach_final.pdf and D2_2_Safe-E-Eurostas-6095-Deliverable-Requirements_HW_Safety_ECU_final.pdf

[3] www.autosar.org, list of basic software module AUTOSAR_TR_BSWModuleList.pdf

[4] AUTOSAR_TR_Safety_ConceptreportStatus.pdf etc.…

10 Acknowledgments

This document is based on the SAFE project in the framework of the ITEA2, EUREKA cluster programme Σ! 3674. The work has been funded by the German Ministry for Education and Research (BMBF) under the funding ID 01IS11019, and by the French Ministry of the Economy and Finance (DGCIS). The responsibility for the content rests with the authors.