Document Title	Requirements on Core Test
Document Category	Requirement specification (RS)
Document UID	258
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• No content changes
2025-11-27	R25-11	AUTOSAR Release Management	• No content changes
2024-11-27	R24-11	AUTOSAR Release Management	• No content changes
2023-11-23	R23-11	AUTOSAR Release Management	• No content changes
2022-11-24	R22-11	AUTOSAR Release Management	• No content changes
2021-11-25	R21-11	AUTOSAR Release Management	• No content changes
2020-11-30	R20-11	AUTOSAR Release Management	• No content changes
2019-11-28	R19-11	AUTOSAR Release Management	• No content changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• Editorial changes

△

2016-11-30	4.3.0	AUTOSAR Release Management	• Editorial changes
2014-10-31	4.2.1	AUTOSAR Release Management	• Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Editorial changes • Renamed "RS_BSWAndRTEFeatures" into "RS_Features"
2013-03-15	4.1.1	AUTOSAR Administration	• Formal update of the document template • Add traceability to features • Clarification of one requirement
2011-12-22	4.0.3	AUTOSAR Administration	• Added a new requirement for foreground test • Clarification of some requirements
2010-09-30	3.1.5	AUTOSAR Administration	• Initial Release

Disclaimer

This work (specification and/or software implementation) 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 work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work 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 work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Scope of Document 5

2 Conventions to be used 6

3 Acronyms and abbreviations 7

4 Requirements Specification 8

4.1 Functional Overview . . . 8
4.1.1 Definition of Core . . . 8
4.1.2 Multicore Support . . . 10
4.1.3 Architectural Prerequisites . . . 11
4.1.3.1 Resource Allocation . . . 11
4.1.3.2 Test Concept . . . 11
4.1.3.3 Limitations . . . 12
4.2 Functional Requirements . . . 12
4.2.1 Configuration . . . 12
4.2.2 Normal Operation . . . 13
4.2.3 Initialisation . . . 20
4.2.4 Shutdown Operation . . . 21
4.3 Non-Functional Requirements (Qualities) . . . 21

5 References 24

A Change history of AUTOSAR traceable items 25

A.1 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 25
A.1.1 Added Requirements in R26-11 . . . 25
A.1.2 Changed Requirements in R26-11 . . . 25
A.1.3 Deleted Requirements in R26-11 . . . 25
A.2 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 25
A.2.1 Added Requirements in R25-11 . . . 25
A.2.2 Changed Requirements in R25-11 . . . 25
A.2.3 Deleted Requirements in R25-11 . . . 25

1 Scope of Document

This document defines general rules and requirements for Core Test specification in AUTOSAR. It shall be used as a basis for each requirements document.

Care has been taken to insure consistency between the Core test, RAM test and Flash test SRS documents.

2 Conventions to be used

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see [1, Standardization Template].

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see [1, Standardization Template].

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as follows.

Note that the requirement level of the document in which they are used modifies the force of these words.

- MUST: This word, or the adjective “LEGALLY REQUIRED”, means that the definition is an absolute requirement of the specification due to legal issues.
- MUST NOT: This phrase, or the phrase “MUST NOT”, means that the definition is an absolute prohibition of the specification due to legal issues.
- SHALL: This phrase, or the adjective “REQUIRED”, means that the definition is an absolute requirement of the specification.
- SHALL NOT: This phrase means that the definition is an absolute prohibition of the specification.
- SHOULD: This word, or the adjective “RECOMMENDED”, means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase “NOT RECOMMENDED”, means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY: This word, or the adjective “OPTIONAL”, means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item.

An implementation, which does not include a particular option, SHALL be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, SHALL be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides).

3 Acronyms and abbreviations

The glossary below includes acronyms and abbreviations relevant to CoreTest that are not included in the AUTOSAR Glossary [2].

Abbreviation / Acronym:	*Description:*
CPU	Central Processing Unit
MPU	Memory Protection Unit
L1	1st level memory
L2	2nd level memory
MCU	Microcontroller Unit
BIST	Built in Self Test
IRQ	Interrupt Request
Core	A CPU plus closely located functional resources
Atomic sequence/ atomic part	Sequence of software code execution which must not be interrupted at any time
Partial test	A partial test is defined as the test of one or more 'hardware resources'. (A partial test is interruptible because it is executed in background mode).
PCB	Printed Circuit Board
External device	A physical external entity; e.g. a second microcontroller
Resource	A core internal unit which executes a unique functionality (e.g. IRQ-controller)
Checksum/ signature	A numerical representation of the result of a test execution or atomic sequence of a test execution.
Caller/calling entity	The caller/calling entity is located on a higher AUTOSAR or ISO layer. It is the user of the API call.

Table 3.1: Acronyms and abbreviations used in the scope of this Document

Term:	*Description:*
Background test	Background test is called periodically by a SW-scheduler.
Foreground test	Foreground test is called via users call.
Golden (Ref.) Value	Reference value used for comparison (e.g. Checksum/Signature)
Good Case	The execution finished without reporting an error

Table 3.2: Terms used in the scope of this Document

As this is a document from professionals for professionals, all other terms are expected to be known.

4 Requirements Specification

This chapter describes all requirements driving the work to define the CoreTest.

4.1 Functional Overview

This module describes the requirements for an API specifying test cases in accordance with the automotive norm. It covers periodic as well as start-up tests. This is meant to be integrated in the overall safety concept and will not give the required diagnostic coverage on its own.

The test may be run in background or foreground mode.

In background mode the test is called periodically by a scheduler, and is interruptible on completion of the current atomic sequence which is a part of the core test. One complete test may consist of many atomic sequences which test functionality of the core entities. This complete test is split up over many atomic test parts to take care of real time operating system requirements in case of scheduling of tasks.

In foreground mode the tests can be used to test the whole core functionality or selected blocks, e.g. prior to running a critical task.

It shall be allowed to cancel the background mode and start a foreground mode. It shall not be possible to have both modes being executed at the same time. If a background task is running and a foreground task is requested, the background task should be cancelled (e.g. at the end of an atomic sequence) before calling the foreground task.

The complete test consists of 2 steps:

1. Run dedicated instruction sequences to stimulate gates and flops and compute a checksum/signature as the result representation.
2. Provides compared checksum/signature - or - Compare computed checksum with reference value ("Golden reference value") and decides whether the test is passed or failed - or - stores computed checksum and provide it on demand to an external caller.

The test computes both steps and returns pass/fail status, or just computes the checksum and provide a notification of completion to the calling entity. This is to allow a higher degree of flexibility in the implementation of test and supervision concepts.

The caller can also be a software component running on a different CPU or an external device.

4.1.1 Definition of Core

The Core is defined as the central processing unit (CPU), all dedicated memory and bus interfaces (TCM, L1, L2 cache, system bus, etc.) and all dedicated supporting

functionality (e.g. interrupt controller, debug, etc.). Throughout this document the expression 'Core' is used for referencing to this definition. A very generic block diagram is shown below. Cores which implement more than one generic CPU should have more than one core test entity.

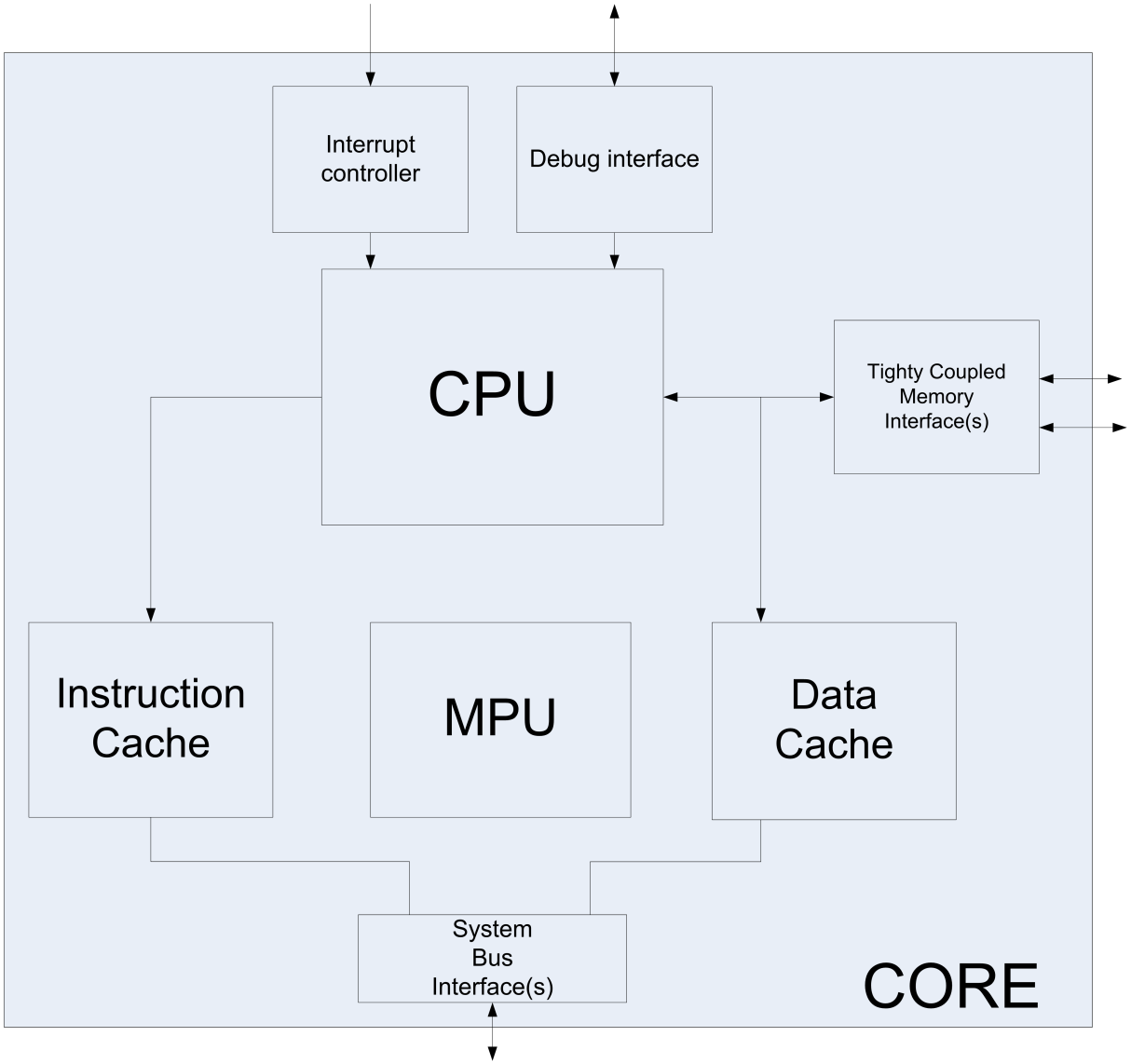


Figure 4.1

The requirements are derived from the automotive standards. Busses have to be tested including arbitration, MMU/MPU, caches, tightly coupled memories, general purpose and dedicated registers, numerical execution units, including address generation and interrupts plus exception handling.

The corresponding tests are listed in the automotive standard. APIs are foreseen for techniques defined as test by the automotive standard, with exception of boundary scan test which will not be in the scope of this document.

Not covered are permanent monitoring techniques nor redundant hardware techniques (e.g. lock-step CPU). If present and requiring software support, they may have to be addressed by MCAL complex drivers.

Note: The Core test initiates diagnostic events only. It shall be used to detect static hardware errors at runtime. Transient faults and intermittent faults are not covered and cannot be detected by dedicated test-software support.

Note: A Core test reports errors in all dedicated memory and bus interfaces (TCM, L1, L2 cache, system bus, etc.) and all dedicated supporting functionality (e.g. interrupt controller, debug, etc.) to the diagnostic event manager (DEM) [3]. For the CPU (e.g. ALU, Prefetch queue) inside a core - only a successful execution of a test or an atomic part of the test ("Good case") can be reported. The errors cases for the CPU inside the core cannot be reliably reported to DEM. Events at DEM have to be defined accordingly. Results/errors are reported though the DEM API (BSW, Dem_SetEventStatus()).

Note: The Core test implementation shall be focused to test the core itself with no interference to the application implementation itself. Anyhow some performance and timing effort shall be considered due to core test computation needs. Anyhow, this needs to be handled by upper entities/layers as seen relative to the Core test driver executing on lower layers or any caller of MCAL core test driver and therefore is out of scope of a driver implementation.

4.1.2 Multicore Support

It shall be possible to execute a Core test on every equal instance of a core inside a silicon device. The Core test itself and the API do not have to be aware of the system architecture itself due to its fact of being a driver located on lower AUTOSAR layers [4].

Additionally a Core test have not to be aware of the number of cores which co-exist in the overall system architecture and is only focused on one single core entity (i.e. if there are multi-cores, then the user-application has to schedule multiple entities of the same test for each core).

Therefore there has to be a clear distinction between the expressions 'Multi-microcontroller' and 'multi-core'. Multi-Microcontroller system designs are out of scope of a core test and its driver API due to the nature of being a driver and a driver API.

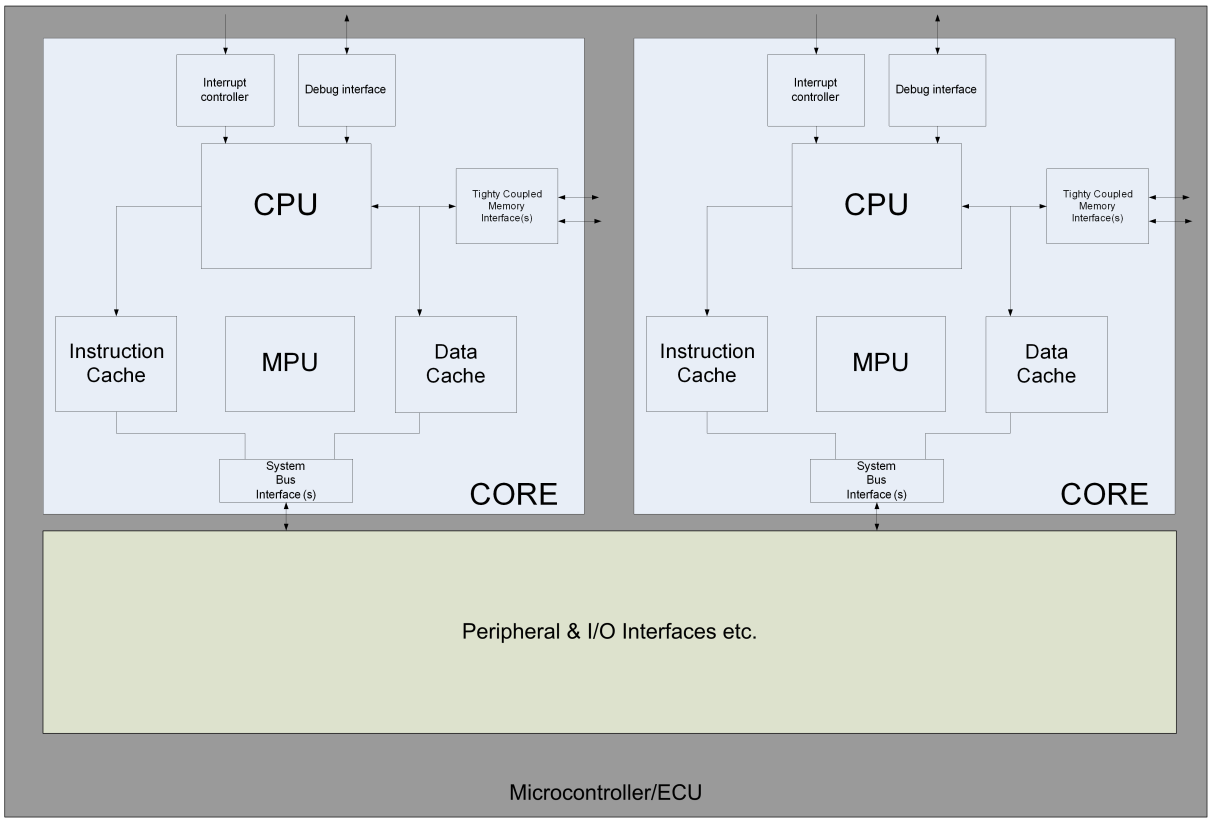


Figure 4.2

As a summary, the core test is a local MCAL driver and as such it has no horizontal view to the system architecture design as well as to other microcontrollers or upper layer services.

4.1.3 Architectural Prerequisites

4.1.3.1 Resource Allocation

There is no resource managing entity available in AUTOSAR upper layers (e.g. ISO 7-layer model - session management). It is necessary to temporarily free a local core resource (e.g. IRQ controller) from application usage to avoid unwanted behavior and interference between test and application during runtime. There is no managing entity available within AUTOSAR architecture to actively handle this requirement prior starting to execute a core test (Feb/2008, R3.0). An MCAL driver cannot handle resource management due to its state of being a driver located in lower AUTOSAR layers. The ECU state manager [5] might be extended to handle this as an additional state or mode.

4.1.3.2 Test Concept

Today AUTOSAR does not support runtime testing; therefore no test managing entity is available in AUTOSAR upper layers. Due to the intentionally missing ability of an

MCAL driver to directly access test results being executed on other cores (e.g. Multi-microcontroller systems), a test managing entity is needed in upper AUTOSAR layers architecture [4] to handle test result processing (local, external) and related reactions of the overall system architecture.

4.1.3.3 Limitations

Due to 4.1.3.1 and 4.1.3.2, a Core test implementation might be limited to be executed during power-up/start-up time where core resources are not shared among different active application tasks or entities (e.g. IRQ-controller, DMA) -OR- might be limited to test resources with are not shared during runtime (e.g. CPU itself).

4.2 Functional Requirements

4.2.1 Configuration

[SRS_CoreTst_14101] The Core Test Shall Be Configurable ⌈

Description:	The Core functionality to be tested and the atomic tests to be run shall be configurable.
Rationale:	The new Cores are highly configurable at synthesis. Caches, MPU, Tightly Coupled Memories/Internal Memories and other functionality is implementation specific and optional/configurable. The tests need to reflect the configuration of the Core on which they will be finally executed
Use Case:	Reuse same test software with different versions of core and select configuration.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14102] Link Time Configuration Shall Be Supported ⌈

Description:	The Core functionality to be tested and the atomic tests to be run shall be configured at link time by object libraries
Rationale:	The core test shall be available as object library. No runtime (post build) configuration is required as core functionality is fixed and not dependent on software variants and use cases.
Use Case:	Reuse same test software with different versions of core and select configuration.
Dependencies:	–
Supporting Material:	–

⌋

4.2.2 Normal Operation

[SRS_CoreTst_14104] Core Register Test Shall Be Available ⌈

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:	–

⌋

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

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:	–

⌋

[SRS_CoreTst_14106] Core ALU Test Shall Be Available ⌈

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:	–

⌋

[SRS_CoreTst_14107] Core Address Generator Test Shall Be Available ⌈

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:	–

⌋

[SRS_CoreTst_14108] Core Memory Interfaces Test Shall Be Available ⌈

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:	–

⌋

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

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:	–

⌋

[SRS_CoreTst_14110] Cache Controller Test Shall Be Available ⌈

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:	–

⌋

[SRS_CoreTst_14111] The Core Test Shall Be Divided into Atomic Sequences ⌈

Description:	The Core test module shall be divided into a sequence of atomic tests. The execution time as well as the code length of an atomic test shall be as short as practically feasible. The implementer shall provide the runtime in number of cycles.
Rationale:	In order not to corrupt the state a Core test, it cannot be interrupted and resumed and will have to run to completion of at least a single atomic sequence. To avoid increasing the interrupt latency beyond acceptable levels, the different building blocks of a Core shall be tested separately in individual atomic tests.
Use Case:	–
Dependencies:	–

▽

△

Supporting Material:	–

⌋

[SRS_CoreTst_14112] There Shall Be a Single API for the Core Test Service ⌈

Description:	There shall be a single API calling the atomic Core tests in sequence. The implementer shall state the sequence and dependencies if required.
Rationale:	Ease of implementation: single entry point for multiple tests (expected to be to most common use)
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14113] The API Shall Have a Parameter to Select Which Component Shall Be Tested ⌈

Description:	There shall be a parameter to select which component of the core to test. The following components shall be testable separately e.g.: • CPU a as whole • External and annex modules to the CPU like e.g. cache, MPU, interrupt controller individually Any kind of combination of tests can be selected, but a least one test have to be selected as a minimum.
Rationale:	–
Use Case:	OS can test components individually prior to re-initialisation or mode change or all available Core component tests in a one go sequence during start up phase/time.
Dependencies:	[SRS_CoreTst_14112]
Supporting Material:	–

⌋

[SRS_CoreTst_14114] A Main Function for the Core Test Shall Be Available ⌈

Description:	There shall be a main-processing function for the Core Test (which has a different meaning compared to the main() function call in a C-language representation). Though the main function the test sequence can be executed without any handling overhead of the Core test execution internals by the application.
Rationale:	The Core test may be called by the BSW scheduler in background mode.
Use Case:	Cyclic background core test.

▽

△

Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14115] Test Metrics Shall Be Available to Caller ⌈

Description:	The checksum result of each partial test shall be stored in an internal variable. This variable hold the last result for the call to read out, no history buffer is foreseen.
Rationale:	The caller will compare with 'a golden value' and decide whether the test is passed or failed, The caller could be a SW component running on the tested, a separate on chip CPU or an external device.
Use Case:	Having the detailed results for each part of the core will allow a higher flexibility in the implementation of recovery mechanisms. E.g. if the MPU is detected to be faulted, the OS could run in an unprotected mode, if the cache is faulty, the system could run with reduced performance and functionality, etc.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14116] A Service shall be provided which returns a checksum/signature as test result ⌈

Description:	The test first computes a checksum/signature as test result representation. The comparison with the golden reference value to decide whether it is passed or failed is left over to an external/higher entity.
Rationale:	This service is needed because the check of a pass or fail criteria shall be done from a different entity.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14131] A Service shall be provided which returns a Pass/Fail Status Representation as a test result ⌈

Description:	The test first computes an algorithm to test the core module and then compares the test result with the golden reference value to decide whether it is passed or failed. The representation value for 'pass' or 'fail' is returned to the calling entity.
Rationale:	Provide a different reporting method for smaller ECU systems.
Use Case:	–
Dependencies:	–

▽

△

Supporting Material:	–

⌋

[SRS_CoreTst_14117] Faults Shall Be Treated as Production Errors ⌈

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:	–

⌋

[SRS_CoreTst_14118] The results of the Core test module shall be provided to the user ⌈

Description:	The results of the Core test module shall be provided to the user. User shall have the possibility to get the status of the Core test at any time. This shall be implemented as a get-status-interface and shall be configurable during compile time. This function shall be optional.
Rationale:	Consistency with RAM test
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14119] A Notification of Completion Shall Be Provided ⌈

Description:	The system or caller shall be notified the test has run to completion.
Rationale:	See description of core test usage in section 5.1
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14126] It Shall Be Possible to Cancel a Running Test ⌈

Description:	It shall be possible to stop the test after completion of current atomic sequence.
Rationale:	Requirement for stopping the service from running due to change running mode. If you change the ECU mode it should be possible to stop the running coretest by software.
Use Case:	It shall be allowed to cancel the background mode and start a foreground mode. It shall not be possible to have both modes being executed at the same time. If a background task is running and a foreground task is requested, the background task should be cancelled (e.g. at the end of an atomic sequence) before calling the foreground task.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14130] Destructive Test Shall Restore Original State of tested Entity ⌈

Description:	A core test shall restore the state of the tested entity as it was before the test execution was started.
Rationale:	In case of destructive tests, values will be modified during core test and this will cause interference with the application.
Use Case:	E.g. test of core register set or interrupt controller configuration
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14133] Each Core Test interval shall have an identifier ⌈

Description:	Each Core Test interval shall have an identifier which shall be incremented by each start of a new test interval in background mode. This value of the Core Test interval shall be provided to upper layers. The end value of the identifier shall be configurable.
Rationale:	Assign test result or test signature to a test interval on order to monitor test flow from upper software layers.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14134] A Foreground Core Test Shall be Available (open) ⌈

Description:	A service shall be available to test a core entity in foreground mode.
Rationale:	Test core entity during start-up phase. Test core entity before critical operations or core-mode changes
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14128] Core Test shall Not Interfere With the Application (rejected) ⌈

Description:	A core test shall be independently implemented from the application running on the core. The Core test implementation shall be focused to test the Core itself with no modification to the application task. Timing influences to the application shall be considered due to core test computation effort and scheduling.
Rationale:	The core test shall be transparent to the application. A core test has to be provided by the core designer due to very specific test algorithms and complex core structures.
Use Case:	Testing of the core functionality during run time operation in foreground or background mode.
Dependencies:	[SRS_CoreTst_14121], [SRS_CoreTst_14123]
Supporting Material:	–

⌋

[SRS_CoreTst_14129] Multimicrocontroller Support (rejected) ⌈

Description:	Both ECUs supervise each other either themselves or by a third external decision-making unit. If there is more than one Core implemented on an ECU, the calling application or the calling OS shall be able to assign core test to a certain core within the ECU. Core test explicitly does no core assignment.
Rationale:	Applications which require enhanced safety and/or high data throughput.
Use Case:	Testing of ECUs during run time operation.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14127] The Test Shall Have the Ability to Request a Checksum From An External Entity (rejected) ⌈

Description:	The checksum result of an atomic test executed on an external entity shall be requested. This received checksum shall be compared to the internally processed checksum. There is no history buffer foreseen for received checksums.
Rationale:	The core will compare a received checksum with its own final core test result and therefore will be able to decide whether its own test is passed or failed. The external checksum providing entity could be a SW component running on a separate CPU, a monitoring MCU or just an external storage device.
Use Case:	WPII-1.3 "Multi-Microcontroller Support" document proposes a flexible architecture approach where all scales of external monitoring from a simple watchdog until dual core MCU architectures are covered.
Dependencies:	–
Supporting Material:	WPII-1.3, "Multi-Microcontroller Support" document, V1.0, sept/26/2007

⌋

4.2.3 Initialisation

[SRS_CoreTst_14103] An init Function for the Core Test Shall Be Available (rejected) ⌈

Description:	Shall support test according to the automotive standard. • Select a dedicated test coverage level • Activate dedicated diagnostic hardware (if available) • Activate Core internal test and diagnostic modes (if available)
Rationale:	For high coverage levels a hardware support is likely to be needed to achieve related test coverage requirements. An API is needed to initialise the dedicated diagnostic hardware (if available)
Dependencies:	–
Supporting Material:	–

⌋

4.2.4 Shutdown Operation

[SRS_CoreTst_14120] A DeInit Function for the Core Test Shall Be Available (rejected) ⌈

Description:	• Stop dedicated diagnostic hardware (if available) • Disable Core internal test and diagnostic modes (if available)
Rationale:	For the automotive standard additional hardware support is likely to be needed. An API is needed to reset the dedicated diagnostic hardware.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

4.3 Non-Functional Requirements (Qualities)

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

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:	–

⌋

[SRS_CoreTst_14121] Timing Requirements (rejected) ⌈

Description:	Test duration of a Core test software
Rationale:	Run time execution of a Core test is identified a vital requirement and therefore will be a debug event during development phase. Maximum execution time shall not be exceeded to avoid conflicts with OS and/or application software as well as Core performance requirements.
Use Case:	–
Dependencies:	–

▽

△

Supporting Material:	–

⌋

[SRS_CoreTst_14122] An Interface to the DET shall be Available (rejected) ⌈

Description:	The Core test shall provide an interface to the DET for monitoring critical parameters during development.
Rationale:	Critical parameter such as timing budget overrun, or test in progress, should be monitored during development.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14125] Diagnostic Coverage (rejected) ⌈

Description:	Diagnostic coverage of 60%, 90% and 99% shall be proven; the diagnostic coverage refers to the Core. In addition, transient and intermittent errors will have to be detected. It is questionable whether coverage levels higher than 60% can be achieved in SW only, without support of dedicated additional Core test hardware; a software test would not catch the faults as required by the fault model for 90% and 99%.
Rationale:	Mandated by the automotive standard
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_CoreTst_14124] The implementation of the Core test shall have to comply with the IEC61508 (rejected) ⌈

Description:	The implementation of the Core test will have to comply with the IEC61508 Software Requirements to achieve certification. This affects both the development process and the programming techniques.
Rationale:	Mandated by IEC61508
Use Case:	–
Dependencies:	–
Supporting Material:	BRF 00001 - 00100 (ID)

⌋

Note: Questionable whether sufficient test coverage levels can be achieved in SW only, without support of dedicated additional Core test hardware; a software test would not catch the all faults as required by common fault models (e.g. transient faults).

5 References

[1] Standardization Template
AUTOSAR_FO_TPS_StandardizationTemplate

[2] Glossary
AUTOSAR_FO_TR_Glossary

[3] Specification of Diagnostic Event Manager
AUTOSAR_CP_SWS_DiagnosticEventManager

[4] Layered Software Architecture
AUTOSAR_CP_EXP_LayeredSoftwareArchitecture

[5] Specification of ECU State Manager
AUTOSAR_CP_SWS_ECUStateManager

A Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

A.1 Traceable item history of this document according to AUTOSAR Release R26-11

A.1.1 Added Requirements in R26-11

none

A.1.2 Changed Requirements in R26-11

none

A.1.3 Deleted Requirements in R26-11

none

A.2 Traceable item history of this document according to AUTOSAR Release R25-11

A.2.1 Added Requirements in R25-11

none

A.2.2 Changed Requirements in R25-11

none

A.2.3 Deleted Requirements in R25-11

none