Document Title	Specification of RAM Test
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	076
Document Classification	Standard
Document Version	2.1.1
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History

Date	Version	Changed by	Change Description
31.03.2014	2.1.1	AUTOSAR Release Management	• Editorial changes • Updated traceability
07.10.2013	2.1.0	AUTOSAR Release Management	• Removed timing attribute of requirement SWS_RamTst_00110 • Editorial changes • Removed chapter(s) on change documentation
26.02.2013	2.0.0	AUTOSAR Administration	• Alignment to the new SWS_BSWGeneral document • Updated the document for Extended Production Errors • Alignment to official naming in other Autosar documents • Adjustment to ISO 26262: major • Clarification of some requirements
27.09.2011	1.5.0	AUTOSAR Administration	• Clarification of some requirements. • Typos correction. • Added a new requirement for DET error reporting
14.10.2010	1.4.0	AUTOSAR Administration	• Clarification on some configuration parameters • Clarification of some types used in API • Improvement of error reporting
30.11.2009	1.3.0	AUTOSAR Administration	• Foreground tests added • Allow more than one configuration per test algorithm • Further maintenance for R4.0 • Legal disclaimer revised
23.06.2008	1.2.2	AUTOSAR Administration	Legal disclaimer revised
22.01.2008	1.2.1	AUTOSAR Administration	• Correction of figures in Chapter 1 and Chapter 9.

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
11.12.2007	1.2.0	AUTOSAR Administration	• RAM test concept documented and included; • Requirements tables updated; • Wording/grammar changes; • Sequence diagram changes; • Generated content corrected/modified. • Document meta information extended • Small layout adaptations made
24.01.2007	1.1.1	AUTOSAR Administration	• “Advice for users” revised • “Revision Information” added
15.12.2006	1.1.0	AUTOSAR Administration	• File include structure updated • “Modified Hamming code” test removed • RamTst_Stop() & RamTst_Continue() changed to “asynchronous” • Dem API updated • Configuration description corrected • descriptions optimized • Legal disclaimer revised
18.05.2006	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

Advice for users

AUTOSAR Specification Documents 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 Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, 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

1 Introduction and Functional Overview

This document specifies the functionality, API and configuration of the AUTOSAR Basic Software module “RAM Test”.

The RAM Test is a test of the physical health of the RAM cells. It is not intended to test the contents of the RAM. RAM used for registers is also tested.

Within this document, a RAM cell is understood as the unit of memory, which can be individually addressed by the processor. Thus the cell size in bits is for example 16 for a 16-bit processor.

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. ISO 26262 only establishes a distinction between three basic coverage levels Low (60%), Medium (90%) and High (99%) [11]. This basic distinction is also used in the AUTOSAR specification.

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).

A RAM Test may be called synchronously by the test environment (hereafter called “foreground test”) or may be called in a cyclic manner by an OS task or other cyclic calling method (hereafter called “background test”). The test environment may select test parameters, start and stop the test, and get status reports. Development errors are reported to the Development Error Tracer (DET) and production errors are reported to the Diagnostic Event Manager (DEM).

The RamTst module consists of a RamTst_MainFunction() for background testing, the API's for foreground testing, several configuration and status API's (Application Programming Interface), and several configuration containers.

TEST FUNCTION API's	***DEFINITION***
RamTst_Init	Prepare resources for testing as necessary. Initialize the test execution state as necessary. Proceed to "test stopped" state after initialization is complete.
RamTst_DeInit	Reset all used registers to reset values, and release all used resources.
RamTst_Allow	Permit the RamTst_MainFunction() to perform testing at its next scheduled call.
RamTst_Stop	Prohibit the RamTst_MainFunction() from performing tests at its next scheduled call. When RamTst_Stop is called, testing stops after the current atomic sequence. Test status is retained, but test parameters (block number, loop count, etc.) are discarded.
RamTst_Suspend	Temporarily prohibit the RamTst_MainFunction() from performing tests at its next scheduled call. When RamTst_Suspend is called, testing stops after the current atomic sequence. Test status and test parameters are retained.
RamTst_Resume	Permits the RamTst_MainFunction() to continue testing at the point where it was suspended, at its next scheduled call. Testing continues according to the saved test parameters.
RamTst_RunFullTest	Test the entire RAM space without interruption. RamTst_Stop must be called prior to calling this API.
RamTst_RunPartialTest	Test the portion of the RAM defined by the API. RamTst_Stop or RamTst_Suspend must be called prior to calling this API.

TEST PARAMETER AND FEEDBACK API's
RamTst_GetVersionInfo
RamTst_GetExecutionStatus
RamTst_GetTestResult
RamTst_GetTestResultPerBlock
RamTst_GetAlgParams
RamTst_GetTestAlgorithm
RamTst_GetNumberOfTestedCells
RamTst_SelectAlgParams
RamTst_ChangeNumberOfTestedCells

RamTst_MainFunction() is the scheduled function for background testing.

- For background testing, RamTst_MainFunction() is called periodically by a scheduler, and is interruptible. One complete test consists of testing with one algorithm over the memory space defined by the currently selected configuration. This complete test is split up over many scheduled calls.

- For foreground testing, RamTst_RunFullTest() or RamTst_RunPartialTest() is called once, and is not interruptible by routines which access the tested memory area (this has to be controlled by the test environment). It tests with one algorithm over the memory space (or a subset in case of partial test) defined by the selected configuration.

The state chart below shows the various states of the test execution.

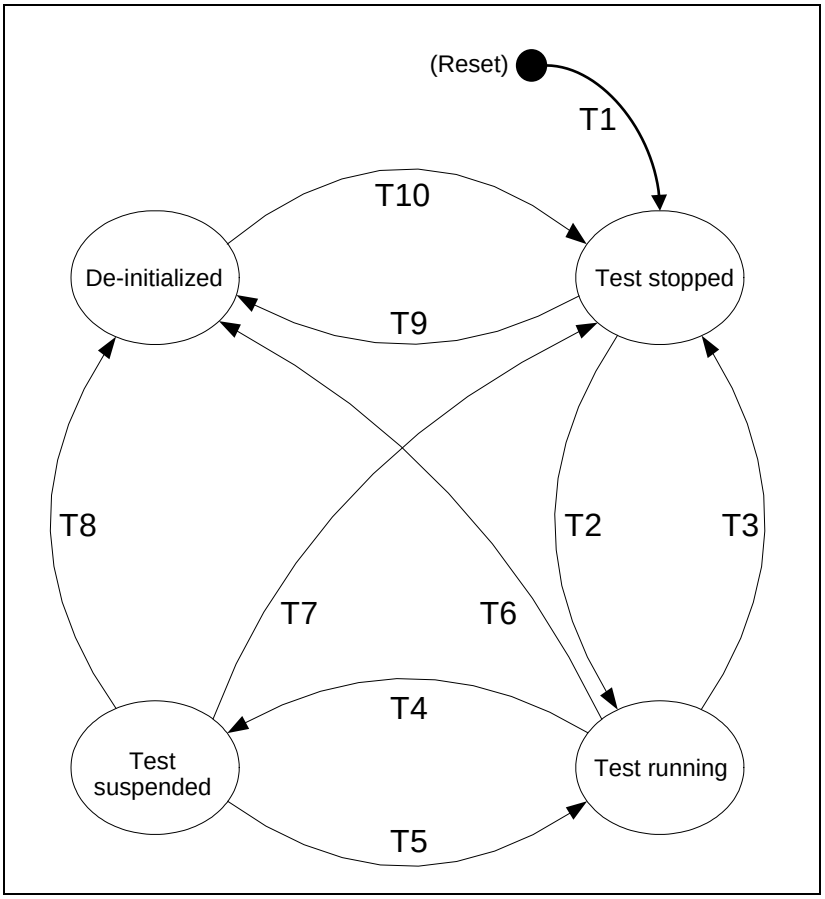


Figure 1 - Phases of Ram Test module

Event	Event Trigger
T1	API: RamTst_Init
T2	API: RamTst_RunFullTest API: RamTst_RunPartialTest API: RamTst_Allow
T3	API: RamTst_Stop (or end of RamTst_RunFullTest) (or end of RamTst_RunPartialTest)
T4	API: RamTst_Suspend (or end of RamTst_RunPartialTest)
T5	API: RamTst_Resume API: RamTst_RunPartialTest
T6	API: RamTst_DeInit
T7	API: RamTst_Stop
T8	API: RamTst_DeInit
T9	API: RamTst_DeInit
T10	API: RamTst_Init

Note: The state “test running” does not necessarily mean that testing is continuously being performed. For foreground testing, it does mean that the test is directly performed by an API call and RamTst_MainFunction() is not scheduled. For background testing, it only means that RamTst_MainFunction() is permitted to test a small portion of the RAM when it is called periodically by the scheduler.

In the actual specification, this state is further divided into “test allowed” and “test running”. The state “test allowed” is only used in the initial phase of a background test; for the big picture given in this overview this difference has been neglected

All API’s and configuration variables are fully defined elsewhere within this document.

The following table shows, which APIs are allowed to be called in each state. For any cell in the table where there is an “N”, there should be a corresponding DET error assigned.

API: Application Programming Interface

	API allowable in this State?			
API’s which cause a change of state in the state chart	**Test Stopped**	**Test Running (or Allowed)**	**Test Suspended**	**Test De-initialized**
RamTst_Init	N	N	N	Y
RamTst_RunFullTest	Y	N	N	N
RamTst_RunPartialTest	Y	N	Y	N
RamTst_Suspend [1]	N	Y	N	N
RamTst_Resume	N	N	Y	N
RamTst_Stop	N	Y	Y	N
RamTst_Allow [2]	Y	N	N	N
RamTst_DeInit	Y	Y	Y	N

	API allowable in this State?			
API's which do not cause a change of state	**Test Stopped**	**Test Running (or Allowed)**	**Test Suspended**	**Test De-initialized**
RamTst_GetVersionInfo	Y	Y	Y	Y
RamTst_GetExecutionStatus	Y	Y	Y	N
RamTst_GetTestResult	Y	Y	Y	N
RamTst_GetTestResultPerBlock	Y	Y	Y	N
RamTst_GetAlgParams	Y	Y	Y	N
RamTst_GetTestAlgorithm	Y	Y	Y	N
RamTst_GetNumberOfTestedCells	Y	Y	Y	N
RamTst_SelectAlgParams [3]	Y	N	N	N
RamTst_ChangeNumberOfTestedCells [4]	Y	N	N	N

NOTES:

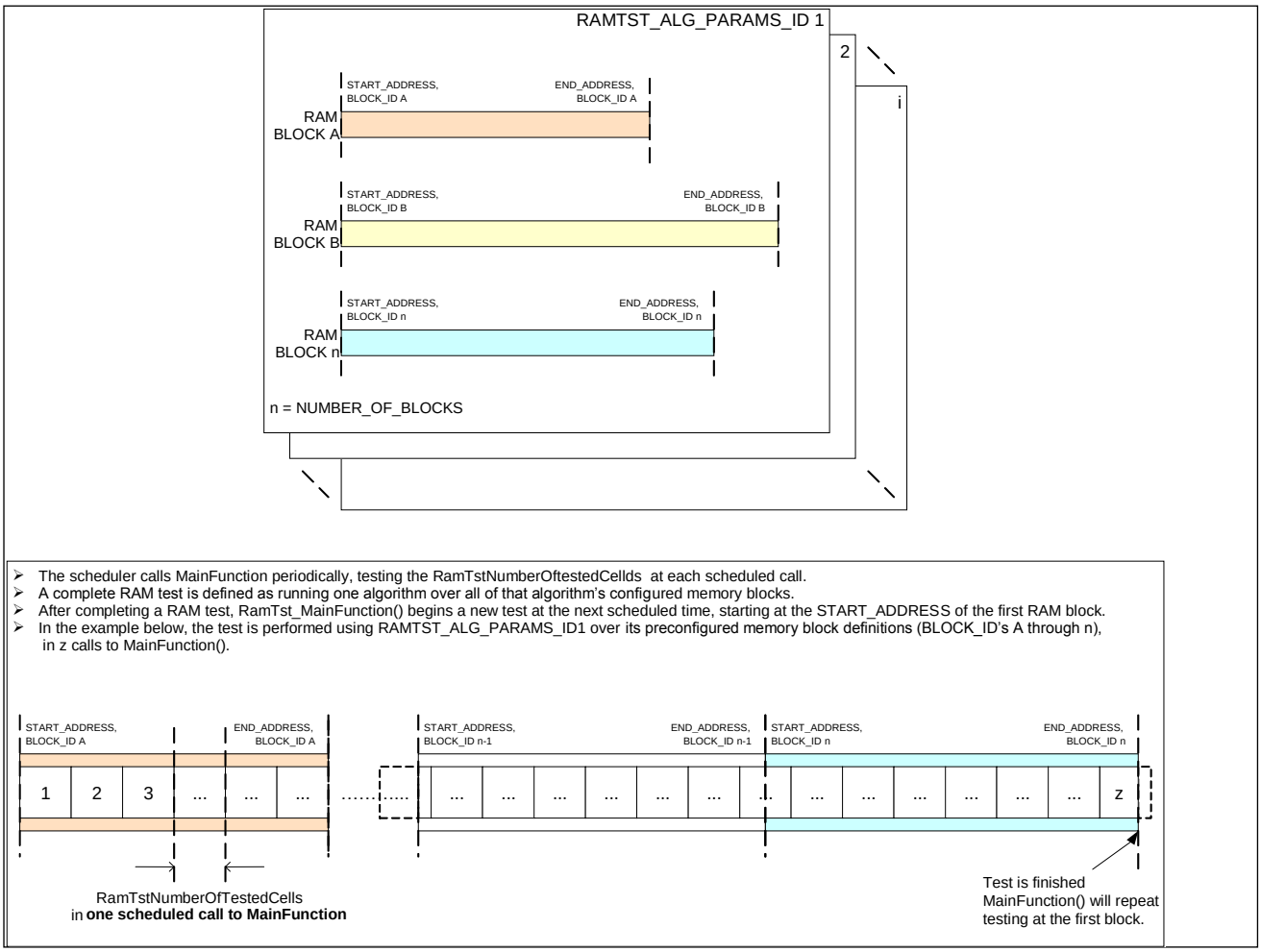
[1] RamTst_Suspend causes a state change to "test suspended" at the end of the current RamTst_MainFunction() atomic sequence if RamTst_MainFunction() is actively testing.

[2] RamTst_Allow is called to permit the RamTst_MainFunction() to test when called, it does not initiate any test itself.

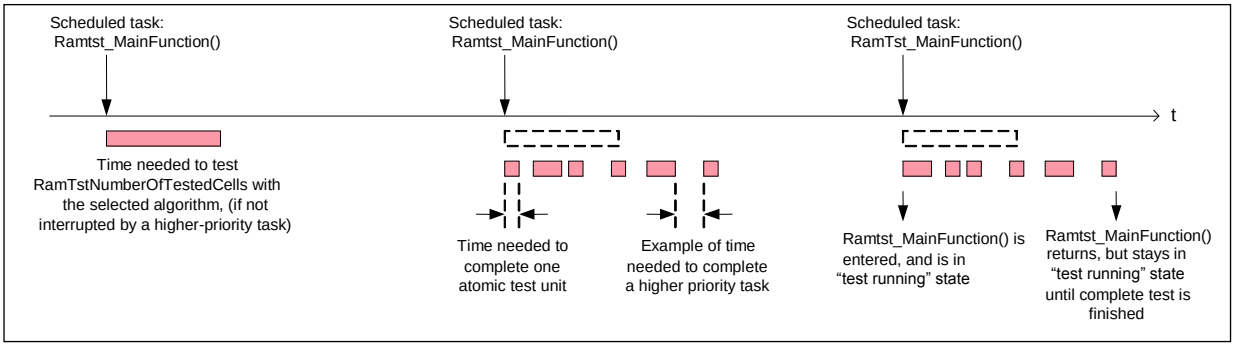
[3] RamTst_Stop must first be called before selecting another configuration parameter set by RamTst_SelectAlgParams.

[4] RamTst_ChangeNumberOfTestedCells operates at the end of the current RamTst_MainFunction() atomic sequence if RamTst_MainFunction() is actively testing. For a foreground test, RamTst_ChangeNumberOfTestedCells is not relevant.

The following figure shows how blocks are configured for an algorithm, and how RamTst_MainFunction() then tests the memory cells for each block in a background test.



The following figure shows how RamTst_MainFunction() is called by the scheduler, and how it can be interrupted between atomic pieces by higher priority tasks.



RamTstNumberOfTestedCells

The RamTstNumberOfTestedCells default is set by configuration (pre-compile or link) in the RamTstAlgParams container and applies to every block defined within an algorithm, but can be different for each RamTstAlgParams, thus can be different for different algorithms or for different parameter sets for the same algorithm. RamTstNumberOfTestedCells can be changed during runtime using the API RamTst_ChangeNumberOfTestedCells. This capability, for example, could be used to reduce the duration of the RAM test task before running some other high-bandwidth task in order to prevent task overruns. Such a situation could occur when unusual conditions in a vehicle cause a normally dormant special algorithm to become active.

RamTstNumberOfTestedCells is only applicable to background testing.

RamTstNumberOfTestedCells may not exceed RamTstMaxNumberOfTestedCells.

The absolute maximum size of RamTstNumberOfTestedCells for a given RamTstAlgParams container is defined and documented by the implementer. This maximum should be equal to the sum of the block sizes as defined by the block descriptions. The integrator sets RamTstExtNumberOfTestedCells to this absolute maximum value (pre-compile or link) in the RamTstAlgParams container. RamTstExtNumberOfTestedCells is not changeable during run time.

The integrator also configures (pre-compile or link) the RamTstMaxNumberOfTestedCells for each RamTstAlgParams container. The integrator must carefully select RamTstMaxNumberOfTestedCells such that it puts an upper limit on the run time of RamTst_MainFunction() in a background task according to the system needs for throughput. In no case should RamTstMaxNumberOfTestedCells be set to a value greater than RamTstExtNumberOfTestedCells. RamTstMaxNumberOfTestedCells is not changeable during run time.

The minimum value of RamTstNumberOfTestedCells is defined and documented by the implementer. The minimum should be defined as one cell unless there is some physical reason for a larger minimum. The integrator configures (pre-compile or link) the RamTstMinNumberOfTestedCells to be greater than or equal to the minimum defined by the implementer. RamTstMinNumberOfTestedCells applies to the entire RAM test module, and not to individual algorithms or parameter sets. It is configured in the RamTstConfigParams container. RamTstMinNumberOfTestedCells is not changeable during run time.

The cell size (in terms of bits) is also defined by the implementer and cannot be changed at integration time, as it should be a fixed value for a given processor. Therefore the corresponding parameter is specified as a published parameter (see chapter10.3).

No matter how many blocks or partial blocks are tested in one RamTst_MainFunction() scheduled call, test status information must be maintained for each block separately.

RamTst_MainFunction() .
A **background** test is performed by the scheduler periodically calling the RamTst_MainFunction() to test a RamTstNumberOfTestedCells of memory using the selected algorithm, until the entire defined area of RAM is tested. This RamTst_MainFunction() can be interrupted at the end of each atomic sequence during a scheduled call.

RamTst_MainFunction():

- Is made up of one or more atomic (i.e. uninterruptable) pieces of code. The number of cells that can be tested in one atomic sequence is considered as implementation specific, thus it is not determined by any (standardized) configuration parameter. However, it is expected that at least RamTstMinNumberOfTestedCells are completely tested during one atomic sequence. It should be noted, that in general the detection of coupling faults between cells is limited to those cells which are tested together in the same atomic sequence.
- At the end of each atomic piece, internal flags are checked to see if an OS task has changed any parameter of the state chart, and to respond to question-type API's.
- Knows inherently:
  - which algorithm it is using;
  - which memory blocks must be tested for this algorithm,
  - start and end addresses of each block;
  - number of cells to test at each call
  - further parameters for the test (see chapter 7.6)
- Remembers:
  - which block it is in;
  - which address to start at in the next call;
  - status of the test;
  - overall test results;
  - test results for each block.
- When RamTstNumberOfTestedCells is reached, RamTst_MainFunction() ends testing for that scheduled call, and starts testing in the next scheduled call at the next (saved) address.
- When the end of a block is reached during a scheduled call, RamTst_MainFunction() continues testing at the beginning of the next block, and continues until RamTstNumberOfTestedCells is reached. (Note: The atomic test sequence should be careful to take into account any issues regarding crossing into the next block.)
- When all blocks are fully tested, RamTst_MainFunction() issues a notification and repeats testing at the first block.
- If there is an error during testing, RamTst_MainFunction() issues a notification (if configured) and continues testing.

RamTst_RunFullTest, RamTst_RunPartialTest
"Full" and "Partial" refers to full or partial memory, and **not** the full or partial set of algorithms over the memory space. The test is performed over the specified memory area using only one algorithm. The desired parameter set (which includes the

algorithm) is selected by calling the API RamTst_SelectAlgParams before calling the foreground test API.

Note that due to the possibility of testing larger memory areas without interruption the fault coverage of foreground tests is in general better than of background test for the same algorithm.

RamTst_RunFullTest API:
The user calls RamTst_RunFullTest with no arguments (the test parameter set is selected before). This test is normally used for a full RAM check at system startup or shutdown.

Sequence:
RamTst_Stop
RamTst_SelectAlgParams to chose the desired parameter set
RamTst_RunFullTest

RamTst_RunPartialTest API:
The user calls RamTst_RunPartialTest with one argument specifying the desired block to be tested. This test is used for example to check a specified memory section immediately before using that memory. This capability is to enable a system safety concept.

Sequence:
RamTst_Stop
RamTst_SelectAlgParams to chose the desired parameter set
RamTst_RunPartialTest (ChosenBlock)

or if background test shall continue afterwards:
RamTst_Suspend
RamTst_RunPartialTest (ChosenBlock)

2 Acronyms and Abbreviations

Abbreviation / Acronym:	*Description:*
API	Application Programming Interface
CRC	Cyclic Redundancy Check
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DMA	Direct Memory Access
ECC	Error Correction Code
NMI	Non Maskable Interrupt
RAM	Random Access Memory

Definitions
Note: These definition are copied from the AUTOSAR_TR_Glossary.pdf

Synchronous: A communication is synchronous when the calling software entity is blocked until the called operation is evaluated. The calling software entity continues its operation by getting the result. Synchronous communication between distributed functional units has to be implemented as remote procedure call.

Asynchronous: Asynchronous communication does not block the sending software entity. The sending software entity continues its operation without getting a response from the communication partner(s). There could be an acknowledgement by the communication system about the sending of the information. A later response to the sending software entity is possible.

3 Related Documentation

3.1 Input Documents

[1] List of Basic Software Modules
AUTOSAR_TR_BSWModuleLis.pdf

[2] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[4] General Requirements on SPAL
AUTOSAR_SRS_SPALGeneral.pdf

[5] Requirements on RAM Test
AUTOSAR_SRS_RAMTest.pdf

[6] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[7] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

3.2 Related Standards and Norms

[8] D1.5-General Architecture; ITEA/EAST-EEA, Version 1.0; chapter 3, page 72 et seq.

[9] D2.1-Embedded Basic Software Structure Requirements; ITEA/EAST-EEA, Version 1.0 or higher.

[10] D2.2-Description of existing solutions; ITEA/EAST-EEA, Version 1.0 or higher.

[11] ISO 26262-5:2011: Road vehicles – Functional safety – Part 5: Product development at the hardware level.

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software modules [7] (SWS BSW General), which is also valid for RAM Test.

Thus, the specification SWS BSW General shall be considered as additional and required specification for RAM Test.

4 Constraints and Assumptions

Note: To achieve ISO 26262 compliance, the software implementation must be according to the requirements of ISO 26262 for the required safety integrity level (ASIL A, ASIL B, ASIL C or ASIL D) of the safety goals of the system.

4.1 Limitations

[SWS_RamTst_00002] 「 During the execution of a RAM test algorithm, no other software shall be allowed to modify the RAM area under test. 」()

In case of background test, the testing code shall be implemented in small atomic pieces in order to accomplish this.

In case of foreground test, it is assumed that the test environment provides the conditions for exclusive access to the tested RAM area.

The rationale behind this requirement is the incapability of the RAM test module to ensure data consistency (e.g. during an NMI, or during a DMA transfer).

[SWS_RamTst_00082] 「 The implementer shall provide integration hints for each algorithm, e.g. “do not use in parallel with a DMA”. 」()

When testing shared memory in a multi-core system it might not be possible to get exclusive access to more than one memory cell via interrupt locking. In this case, the usage of a test configuration for shared memory blocks must be restricted to foreground tests and to specific ECU states, see 3 Related Documentation and SWS_RamTst_00203 for additional information.

In a multi-core system, disabling the interrupts does not guarantee atomicity for more than a single memory access. Since a RAM test operation consists of more than a single memory access, a more sophisticated mechanism is needed to realize atomicity. Therefore, different solutions for shared and non-shared RAM are required.

4.2 Full RAM Test

A full test shall be executed when only a single core is running. In a Master-Slave system, this is possible during the initial boot phase while only the master core is active. Additionally full tests can be performed during ECU sleep mode. This allows the EcuM to delay the sleep state of one of the cores to perform a RAM tests on that core.

Full RAM tests shall be allowed whenever atomicity across cores can be guaranteed, known moments are,

1. before the master core has activated any other core.
2. when all cores except one have entered sleep mode.

4.3 Partial RAM Test

During normal operation, the memory is split into non-shared and shared parts. The integrator has to specify for each `ALGORITHM_ID` the memory areas on which the algorithm works. A non-shared area is owned by a specific core and can be tested by the code running on that core as in the single core case. Lack of atomicity in MC causes problems for shared memory.

4.4 Applicability to Car Domains

No restrictions.

5 Dependencies to Other Modules

An actual selected parameter set for a RAM test basically consists of a set of RAM bocks and a test algorithm.

The available parameter sets for the RAM blocks and test algorithms must be configured at pre-compile time. The software responsible for monitoring the RAM state of health must then select an appropriate parameter set, (it can also switch between several ones at runtime), according to the results of the ECU safety analysis.

Within each parameter set, the detailed definition of the blocks to be tested, e.g. their start/end address, must be configured at pre-compile or link time. Further parameters controlling the details of the test are explained later in the document (see 7.6).

If the test environment calls a RAM Test API to test all or part of the RAM immediately (in the foreground), then the test environment is responsible to mask interrupts as desired or to call the test in a particular situation, where the tested blocks are not accessed by other modules.

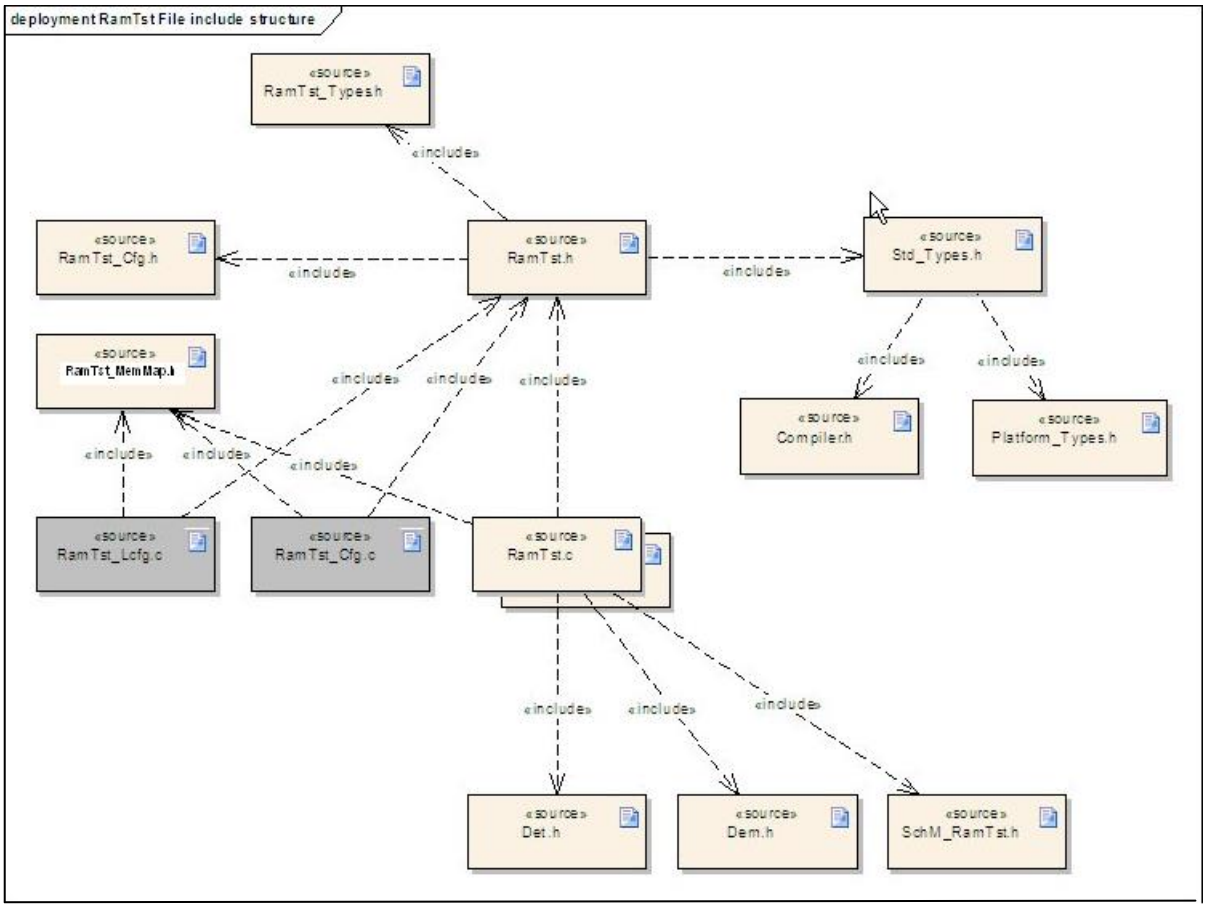
For background testing, the ECU State Manager or the BSW Scheduler must schedule the RAM Test main function. The number of cells tested in one cycle is set as a default at pre-compile or link time based upon the needs of the scheduler. This size may be changed during runtime to accommodate a change in the schedule. In addition, the parameter set used for the background test may be switched during runtime, so that e.g. certain critical blocks can be tested in certain ECU states with higher coverage than in other ECU states or uncritical blocks can be excluded from tests in certain ECU states.

In development mode the error-hook function of module DET will be called.

5.1 File Structure

5.1.1 Header File Structure

[SWS_RamTst_00003] ⌈ The include structure for the source code of the RAM Test module shall be as follows:



」(SRS_BSW_00346, SRS_BSW_00415, SRS_BSW_00435, SRS_BSW_00436, SRS_BSW_00447)

6 Requirements Traceability

Requirement	Description	Satisfied by
-	-	SWS_RamTst_00002
-	-	SWS_RamTst_00005
-	-	SWS_RamTst_00009
-	-	SWS_RamTst_00014
-	-	SWS_RamTst_00018
-	-	SWS_RamTst_00021
-	-	SWS_RamTst_00034
-	-	SWS_RamTst_00047
-	-	SWS_RamTst_00069
-	-	SWS_RamTst_00082
-	-	SWS_RamTst_00085
-	-	SWS_RamTst_00089
-	-	SWS_RamTst_00094
-	-	SWS_RamTst_00096
-	-	SWS_RamTst_00098
-	-	SWS_RamTst_00100
-	-	SWS_RamTst_00101
-	-	SWS_RamTst_00106
-	-	SWS_RamTst_00112
-	-	SWS_RamTst_00147
-	-	SWS_RamTst_00148
-	-	SWS_RamTst_00149
-	-	SWS_RamTst_00150
-	-	SWS_RamTst_00167
-	-	SWS_RamTst_00168
-	-	SWS_RamTst_00169
-	-	SWS_RamTst_00171
-	-	SWS_RamTst_00173
-	-	SWS_RamTst_00174
-	-	SWS_RamTst_00188
-	-	SWS_RamTst_00189
-	-	SWS_RamTst_00190
-	-	SWS_RamTst_00191
-	-	SWS_RamTst_00193
-	-	SWS_RamTst_00194
-	-	SWS_RamTst_00195
-	-	SWS_RamTst_00196

-	-	SWS_RamTst_00197
-	-	SWS_RamTst_00198
-	-	SWS_RamTst_00202
-	-	SWS_RamTst_00203
-	-	SWS_RamTst_00204
-	-	SWS_RamTst_00205
-	-	SWS_RamTst_00206
-	-	SWS_RamTst_00207
-	-	SWS_RamTst_00211
-	-	SWS_RamTst_00212
-	-	SWS_RamTst_00215
-	-	SWS_RamTst_00217
-	-	SWS_RamTst_00222
-	-	SWS_RamTst_00223
-	-	SWS_RamTst_00227
BSW00434	-	SWS_RamTst_00999
BSW00443	-	SWS_RamTst_00999
BSW00444	-	SWS_RamTst_00999
RS_SPAL_12448	-	SWS_RamTst_00095
SRS_BSW_00005	Modules of the æC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_RamTst_00999
SRS_BSW_00006	The source code of software modules above the æC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_RamTst_00999
SRS_BSW_00009	All Basic SW Modules shall be documented according to a common standard.	SWS_RamTst_00999
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	SWS_RamTst_00999
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_RamTst_00007, SWS_RamTst_00099
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	SWS_RamTst_00999
SRS_BSW_00162	The AUTOSAR Basic Software	SWS_RamTst_00999

	shall provide a hardware abstraction layer	
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_RamTst_00999
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_RamTst_00999
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_RamTst_00999
SRS_BSW_00172	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	SWS_RamTst_00999
SRS_BSW_00301	All AUTOSAR Basic Software Modules shall only import the necessary information	SWS_RamTst_00999
SRS_BSW_00302	All AUTOSAR Basic Software Modules shall only export information needed by other modules	SWS_RamTst_00999
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_RamTst_00999
SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	SWS_RamTst_00999
SRS_BSW_00309	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	SWS_RamTst_00999
SRS_BSW_00312	Shared code shall be reentrant	SWS_RamTst_00999
SRS_BSW_00314	All internal driver modules shall separate the interrupt frame definition from the service routine	SWS_RamTst_00221
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_RamTst_00999
SRS_BSW_00323	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	SWS_RamTst_00033, SWS_RamTst_00037, SWS_RamTst_00039, SWS_RamTst_00040, SWS_RamTst_00095, SWS_RamTst_00097, SWS_RamTst_00170, SWS_RamTst_00172, SWS_RamTst_00210, SWS_RamTst_00214
SRS_BSW_00325	The runtime of interrupt service	SWS_RamTst_00221

	routines and functions that are running in interrupt context shall be kept short	
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_RamTst_00999
SRS_BSW_00331	All Basic Software Modules shall strictly separate error and status information	SWS_RamTst_00102, SWS_RamTst_00103
SRS_BSW_00333	For each callback function it shall be specified if it is called from interrupt context or not	SWS_RamTst_00999
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_RamTst_00999
SRS_BSW_00337	Classification of development errors	SWS_RamTst_00067
SRS_BSW_00339	Reporting of production relevant error status	SWS_RamTst_00011, SWS_RamTst_00067, SWS_RamTst_00071, SWS_RamTst_00111, SWS_RamTst_00213, SWS_RamTst_00216, SWS_RamTst_01001
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_RamTst_00999
SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_RamTst_00026, SWS_RamTst_00027
SRS_BSW_00345	BSW Modules shall support pre-compile configuration	SWS_RamTst_00058
SRS_BSW_00346	All AUTOSAR Basic Software Modules shall provide at least a basic set of module files	SWS_RamTst_00003
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall be in place	SWS_RamTst_00999
SRS_BSW_00353	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	SWS_RamTst_00999
SRS_BSW_00358	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	SWS_RamTst_00099
SRS_BSW_00361	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	SWS_RamTst_00999
SRS_BSW_00370	-	SWS_RamTst_00999
SRS_BSW_00373	The main processing function of each AUTOSAR Basic Software Module shall be	SWS_RamTst_00110

	named according the defined convention	
SRS_BSW_00374	All Basic Software Modules shall provide a readable module vendor identification	SWS_RamTst_00999
SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_RamTst_00999
SRS_BSW_00376	-	SWS_RamTst_00110
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_RamTst_00999
SRS_BSW_00379	All software modules shall provide a module identifier in the header file and in the module XML description file.	SWS_RamTst_00999
SRS_BSW_00383	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	SWS_RamTst_00999
SRS_BSW_00385	List possible error notifications	SWS_RamTst_00067, SWS_RamTst_01001
SRS_BSW_00386	The BSW shall specify the configuration for detecting an error	SWS_RamTst_00999
SRS_BSW_00399	Parameter-sets shall be located in a separate segment and shall be loaded after the code	SWS_RamTst_00999
SRS_BSW_00400	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	SWS_RamTst_00999
SRS_BSW_00402	Each module shall provide version information	SWS_RamTst_00081
SRS_BSW_00404	BSW Modules shall support post-build configuration	SWS_RamTst_00999
SRS_BSW_00405	BSW Modules shall support multiple configuration sets	SWS_RamTst_00999
SRS_BSW_00406	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	SWS_RamTst_00006, SWS_RamTst_00012, SWS_RamTst_00013
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_RamTst_00109
SRS_BSW_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_RamTst_00999
SRS_BSW_00414	The init function may have parameters	SWS_RamTst_00093

SRS_BSW_00415	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	SWS_RamTst_00003
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_RamTst_00999
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	SWS_RamTst_00999
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the DEM	SWS_RamTst_00999
SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_RamTst_00999
SRS_BSW_00424	BSW module main processing functions shall not be allowed to enter a wait state	SWS_RamTst_00999
SRS_BSW_00425	The BSW module description template shall provide means to model the defined trigger conditions of schedulable objects	SWS_RamTst_00999
SRS_BSW_00426	BSW Modules shall ensure data consistency of data which is shared between BSW modules	SWS_RamTst_00999
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_RamTst_00221
SRS_BSW_00428	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	SWS_RamTst_00999
SRS_BSW_00429	BSW modules shall be only allowed to use OS objects and/or related OS services	SWS_RamTst_00999
SRS_BSW_00432	Modules should have separate main processing functions for read/receive and write/transmit data path	SWS_RamTst_00999
SRS_BSW_00435	-	SWS_RamTst_00003
SRS_BSW_00436	-	SWS_RamTst_00003
SRS_BSW_00437	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	SWS_RamTst_00999
SRS_BSW_00438	Configuration data shall be defined in a structure	SWS_RamTst_00999
SRS_BSW_00439	Enable BSW modules to	SWS_RamTst_00221

	handle interrupts	
SRS_BSW_00440	The callback function invocation by the BSW module shall follow the signature provided by RTE to invoke servers via Rte_Call API	SWS_RamTst_00999
SRS_BSW_00441	Naming convention for type, macro and function	SWS_RamTst_00999
SRS_BSW_00442	The AUTOSAR architecture shall support standardized debugging and tracing features	SWS_RamTst_00159, SWS_RamTst_00161, SWS_RamTst_00162, SWS_RamTst_00163, SWS_RamTst_00164
SRS_BSW_00447	Standardizing Include file structure of BSW Modules Implementing Autosar Service	SWS_RamTst_00003
SRS_BSW_00449	BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType	SWS_RamTst_00999
SRS_BSW_00450	A Main function of a un-initialized module shall return immediately	SWS_RamTst_00175
SRS_RamTst_13800	The number of tested cells shall be changeable at runtime	SWS_RamTst_00036, SWS_RamTst_00107
SRS_RamTst_13802	Multiple RAM areas shall be configurable at post build/ link time	SWS_RamTst_00026
SRS_RamTst_13803	A subset of available RAM Test algorithms shall be selectable at pre-compile time	SWS_RamTst_00026, SWS_RamTst_00027, SWS_RamTst_00063, SWS_RamTst_00224, SWS_RamTst_00225, SWS_RamTst_00226
SRS_RamTst_13804	A subset of the pre-compile time selected RAM Check test algorithms shall be selectable at runtime	SWS_RamTst_00083
SRS_RamTst_13809	It shall be possible to divide the RAM test execution into smaller pieces	SWS_RamTst_00008, SWS_RamTst_00026, SWS_RamTst_00059, SWS_RamTst_00107, SWS_RamTst_00108
SRS_RamTst_13810	Current status of RAM test execution per block shall be available through a get status interface	SWS_RamTst_00010, SWS_RamTst_00011, SWS_RamTst_00019, SWS_RamTst_00024, SWS_RamTst_00038, SWS_RamTst_00104
SRS_RamTst_13811	The RAM test module shall be able to perform its tests in a non-destructive manner	SWS_RamTst_00060, SWS_RamTst_00061, SWS_RamTst_00200
SRS_RamTst_13812	The RAM test module shall be able to perform its tests in a destructive manner	SWS_RamTst_00061, SWS_RamTst_00201
SRS_RamTst_13816	Effects of Instruction / Data queue shall be taken into account	SWS_RamTst_00062
SRS_RamTst_13820	RAM test execution status shall be provided by a notification mechanism	SWS_RamTst_00043, SWS_RamTst_00044, SWS_RamTst_00045, SWS_RamTst_00046, SWS_RamTst_00113, SWS_RamTst_00114
SRS_RamTst_13822	A Test algorithm with low	SWS_RamTst_00224

	coverage shall be available	
SRS_RamTst_13823	A Test algorithm with medium coverage shall be available	SWS_RamTst_00225
SRS_RamTst_13824	A Test algorithm with high coverage shall be available	SWS_RamTst_00226
SRS_SPAL_00157	All drivers and handlers of the AUTOSAR Basic Software shall implement notification mechanisms of drivers and handlers	SWS_RamTst_00043, SWS_RamTst_00044, SWS_RamTst_00045, SWS_RamTst_00046
SRS_SPAL_12056	All driver modules shall allow the static configuration of notification mechanism	SWS_RamTst_00043, SWS_RamTst_00044
SRS_SPAL_12057	All driver modules shall implement an interface for initialization	SWS_RamTst_00007, SWS_RamTst_00026, SWS_RamTst_00027
SRS_SPAL_12063	All driver modules shall only support raw value mode	SWS_RamTst_00999
SRS_SPAL_12064	All driver modules shall raise an error if the change of the operation mode leads to degradation of running operations	SWS_RamTst_00999
SRS_SPAL_12067	All driver modules shall set their wake-up conditions depending on the selected operation mode	SWS_RamTst_00999
SRS_SPAL_12068	The modules of the MCAL shall be initialized in a defined sequence	SWS_RamTst_00999
SRS_SPAL_12069	All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason	SWS_RamTst_00999
SRS_SPAL_12075	All drivers with random streaming capabilities shall use application buffers	SWS_RamTst_00999
SRS_SPAL_12078	The drivers shall be coded in a way that is most efficient in terms of memory and runtime resources	SWS_RamTst_00999
SRS_SPAL_12092	The driver's API shall be accessed by its handler or manager	SWS_RamTst_00999
SRS_SPAL_12125	All driver modules shall only initialize the configured resources	SWS_RamTst_00999
SRS_SPAL_12129	The ISRs shall be responsible for resetting the interrupt flags and calling the according notification function	SWS_RamTst_00221
SRS_SPAL_12163	All driver modules shall	SWS_RamTst_00146

	implement an interface for de-initialization	
SRS_SPAL_12263	The implementation of all driver modules shall allow the configuration of specific module parameter types at link time	SWS_RamTst_00026, SWS_RamTst_00027
SRS_SPAL_12265	Configuration data shall be kept constant	SWS_RamTst_00999
SRS_SPAL_12267	Wakeup sources shall be initialized by MCAL drivers and/or the MCU driver	SWS_RamTst_00999
SRS_SPAL_12448	All driver modules shall have a specific behavior after a development error detection	SWS_RamTst_00033, SWS_RamTst_00037, SWS_RamTst_00039, SWS_RamTst_00040, SWS_RamTst_00084
SRS_SPAL_12461	Specific rules regarding initialization of controller registers shall apply to all driver implementations	SWS_RamTst_00999
SRS_SPAL_12462	The register initialization settings shall be published	SWS_RamTst_00999
SRS_SPAL_12463	The register initialization settings shall be combined and forwarded	SWS_RamTst_00999

7 Functional Specification

7.1 Requirements

[SWS_RamTst_00005] ⌈ The RAM Test module shall provide the background RAM test as an asynchronous service. ⌋()

[SWS_RamTst_00206] ⌈ The RAM Test module shall provide the foreground RAM test as an synchronous service. ⌋()

[SWS_RamTst_00063] ⌈ The configuration process for the RAM Test module shall allow the selection of a subset of different RAM Test algorithms during pre-compile time. ⌋(SRS_RamTst_13803)

This subset is to be chosen from the different RAM Test algorithms as specified in SWS_RamTst_00224, SWS_RamTst_00225, SWS_RamTst_00226, SWS_RamTst_00204

[SWS_RamTst_00060] ⌈ If non-destructive RAM Test is chosen, the RAM Test module shall save the RAM area to be tested before the module modifies it. The RAM Test module shall execute the complete procedure (saving, changing, restoring) without interruption. ⌋(SRS_RamTst_13811)

Note: “Saving” and “restoring” does not necessarily mean explicit copying actions. If the test algorithm is “transparent” it restores the original content in the tested cells after the test without needing additional memory for saving.

[SWS_RamTst_00061] ⌈ For both the destructive and non-destructive options, the RAM Test module shall ensure that the test algorithm does not overwrite the RAM Test internal variables. ⌋(SRS_RamTst_13811, SRS_RamTst_13812)

[SWS_RamTst_00062] ⌈ After writing to a cell and before reading back, the RAM Test module shall provide the possibility to inject instruction(s) forcing the controller to clear its CPU internal cache. ⌋(SRS_RamTst_13816)

[SWS_RamTst_00224] ⌈ The RAM Test module shall provide a test algorithm with low coverage as stated in [11], Table D.1. ⌋(SRS_RamTst_13803, SRS_RamTst_13822)

[SWS_RamTst_00225] ⌈ The RAM Test module shall provide a test algorithm with medium coverage as stated in [11], Table D.1. ⌋(SRS_RamTst_13803, SRS_RamTst_13823)

[SWS_RamTst_00226] ⌈ The RAM Test module shall provide a test algorithm with high coverage as stated in [11], Table D.1. ⌋(SRS_RamTst_13803, SRS_RamTst_13824)

[SWS_RamTst_00204] ⌈ If appropriate, the RAM Test module may provide additional vendor or hardware specific test algorithms or different variants of the algorithms listed above. These algorithms must be clearly documented by the implementer, especially their fault coverage (for vendor specific configuration parameters see SWS_RamTst_00205). ⌋()

[SWS_RamTst_00221] ⌈ A processor specific test algorithm is allowed to make use of hardware macros and/or interrupts supporting the detection of data loss (like CRC, ECC) if appropriate. The implementer must describe any interrupt routine in the Basic Software Module Description and the implementation must follow the general requirements for interrupt handling (see SRS_BSW_00427, SRS_SPAL_12129, SRS_BSW_00325, SRS_BSW_00439, SRS_BSW_00314). ⌋(SRS_BSW_00427, SRS_BSW_00314, SRS_BSW_00325, SRS_BSW_00439, SRS_SPAL_12129)

7.2 Error Classification

7.2.1 Development Errors

[SWS_RamTst_00067] ⌈ The RAM Test shall detect following development errors:

Type or error	*Relevance*	*Related error code*	*Value [hex]*	*Requirement*
A particular API is called in an unexpected state	Development	RAMTST_E_STATUS_FAILURE	0x01	SWS_RAMTST_00033 SWS_RAMTST_00037 SWS_RAMTST_00095 SWS_RAMTST_00097 SWS_RamTst_00170 SWS_RamTst_00172 SWS_RamTst_00210 SWS_RamTst_00214
API parameter out of specified range	Development	RAMTST_E_OUT_OF_RANGE	0x02	SWS_RAMTST_00039 SWS_RAMTST_00040 SWS_RAMTST_00084 SWS_RamTst_00223
API service used without module initialization	Development	RAMTST_E_UNINIT	0x03	SWS_RAMTST_00089
API service called with a NULL pointer	Development	RAMTST_E_PARAM_POINTER	0x04	SWS_RamTst_00222

⌋(SRS_BSW_00337, SRS_BSW_00339, SRS_BSW_00385)

7.2.2 Production Errors

This module does not specify any production errors.

7.2.3 Extended Production Errors

[SWS_RamTst_01001]⌈ The RAM Test shall detect following extended production errors:

Type or error	*Related error code*	*Value*
RAM failure during tests. See SWS_RAMTST_00011 SWS_RAMTST_00213 SWS_RAMTST_00216	RAMTST_E_RAM_FAILURE	Assigned by the DEM

⌋(SRS_BSW_00339, SRS_BSW_00385)

7.3 Error Detection

[SWS_RamTst_00089] ⌈ The function RamTst_Init shall be called first before calling any other RAM test functions. If this sequence is not respected, the error code RAMTST_E_UNINIT shall be reported to the Development Error Tracer (if development error detection is enabled). ⌋()

7.4 Error Notification

[SWS_RamTst_00069] ⌈ Additional errors that are detected because of specific implementation and/or specific hardware properties shall be added in the RAM Test device specific implementation specification. The classification and enumeration shall be compatible to the errors listed above in SWS_RamTst_00067. ⌋()

[SWS_RamTst_00071] ⌈ Production errors shall be reported to Diagnostic Event Manager (DEM) via the Dem_ReportErrorStatus API. ⌋(SRS_BSW_00339)

7.5 Debugging

[SWS_RamTst_00159] ⌈ The internal variable holding the current test execution status shall be available for debugging. ⌋(SRS_BSW_00442)

[SWS_RamTst_00161] ⌈ The internal variable holding the current overall test result shall be available for debugging. ⌋(SRS_BSW_00442)

[SWS_RamTst_00162] ⌈ The internal variables holding the current test results per block shall be available for debugging. ⌋(SRS_BSW_00442)

[SWS_RamTst_00163] ⌈ The internal variable holding the ID for the currently selected test algorithm parameter set shall be available for debugging. ⌋(SRS_BSW_00442)

[SWS_RamTst_00164] ⌈ The internal variable holding the current number of cells to be tested per cycle shall be available for debugging. ⌋(SRS_BSW_00442)

7.6 General Test Behavior

This sections contains detailed specifications items which hold for the foreground test and the background test as well.

Both foreground of background tests are controlled by the currently selected parameter set `RamTstAlgParams` which defines the test algorithm, the set of memory blocks to be tested and several attributes controlling the behavior of the test.

Note that the same test algorithm can be used as part of several different `RamTstAlgParams`, that none of the `RamTstAlgParams` must necessarily contain all memory blocks and that (in general) for foreground and background tests different `RamTstAlgParams` can be selected. This allows for a flexible approach of usings the tests differently in specific ECU modes or for different types of memory.

[SWS_RamTst_00200] ⌈ If the configuration parameter `RamTstTestPolicy` for a block is set to RAMTEST_NON_DESTRUCTIVE, the test algorithm shall restore the original memory content of the tested cells of this block after the test (given that no error is detected). ⌋(SRS_RamTst_13811)

Hint: For a transparent test algorithm, this behavior is automatically fulfilled without additional overhead. For a non-transparent test algorithm, this option means overhead in runtime/memory in order to save and restore the content.

[SWS_RamTst_00201] ⌈ If the configuration parameter `RamTstTestPolicy` for a block is set to RAMTEST_DESTRUCTIVE, the test algorithm shall fill the tested cells after the test with the bit pattern defined for this block by parameter `RamTst_FillPattern` (given that no error is detected). ⌋(SRS_RamTst_13812)

This requirement shall ensure reproducible behavior.

Hint: For a transparent test algorithm, specifying this option would mean runtime overhead. For a non-transparent algorithm, the runtime overhead can be minimized, if the fill pattern corresponds to a constant value left behind by the algorithm anyhow.

[SWS_RamTst_00202] ⌈ The overall test result – for the set of blocks in the current `RamTstAlgParams` – shall be

- `RAMTST_RESULT_NOT_TESTED` if no test was started yet (after reset or de-init).

- RAMTST_RESULT_UNDEFINED if a test was started, not all blocks have yet been tested and no block result is RAMTST_RESULT_NOT_OK.
- RAMTST_RESULT_OK if all blocks have been tested with result status RAMTST_RESULT_OK.
- RAMTST_RESULT_NOT_OK if at least one block test result is RAMTST_RESULT_NOT_OK regardless whether all blocks have been already tested or not. 」()

[SWS_RamTst_00207] 「 The test result for a specific block (identified in the given RamTstAlgParams) – shall be

- RAMTST_RESULT_NOT_TESTED if this block is considered as not yet tested.
- RAMTST_RESULT_UNDEFINED if a test on this block is running.
- RAMTST_RESULT_OK if all memory cells in this block have been tested sucessfully.
- RAMTST_RESULT_NOT_OK if a failure has been detected for at least one memory cell in this block. 」()

For a given processor type, memory layout and fault model, not all possible combinations of test algorithms, block configurations and their attributes make sense. For example:

- The implementer might want to exclude a certain combination of test algorithm and RamTstTestPolicy.
- A certain test algorithm might have to be excluded from background tests due to performance reason.
- Some memory blocks might have to be excluded from background tests due to performance reasons or because an exclusive access cannot be guaranteed under normal operation (e.g. for shared memory).

This leads to the following requirement:

[SWS_RamTst_00203] 「 The implementer shall document possible restrictions for the combination of configuration parameters and for their usage in background/foreground tests. Where applicable, he shall support this by the definition of predefined or recommended configuration parameter values attached to the BSW Module Description. 」()

8 API Specification

8.1 Imported Types

This chapter lists data type definitions for the included variables and constants.

[SWS_RamTst_00098] ⌈

Module	*Imported Type*
Dem	Dem_EventIdType
	Dem_EventStatusType
Std_Types	Std_ReturnType
	Std_VersionInfoType

⌋()

8.2 Type Definitions

8.2.1 RamTst_ExecutionStatusType

[SWS_RamTst_00189] ⌈

Name:	RamTst_ExecutionStatusType	
Type:	Enumeration	
Range:	RAMTST_EXECUTION_UNINIT	The RAM Test is not initialized or not usable.
	RAMTST_EXECUTION_STOPPED	The RAM Test is stopped and ready to be started in foreground or to be allowed in background.
	RAMTST_EXECUTION_RUNNING	The RAM Test is currently running.
	RAMTST_EXECUTION_SUSPENDED	The background RAM Test is waiting to be resumed.
Description:	This is a status value returned by the API service RamTst_GetExecutionStatus().	

⌋()

[SWS_RamTst_00006] ⌈ For the type RamTst_ExecutionStatusType, the enumeration value RAMTST_EXECUTION_UNINIT shall be the default value after a reset. This enumeration value shall have the numeric value 0. ⌋(SRS_BSW_00406)

8.2.2 RamTst_TestResultType

[SWS_RamTst_00190] ⌈

Name:	RamTst_TestResultType	
Type:	Enumeration	
Range:	RAMTST_RESULT_NOT_TESTED	The RAM Test is not executed.
	RAMTST_RESULT_OK	The RAM Test has been tested with OK result
	RAMTST_RESULT_NOT_OK	The RAM Test has been tested with NOT-OK result.
	RAMTST_RESULT_UNDEFINED	The RAM Test is currently running.
Description:	This is a status value returned by the API service RamTst_GetTestResult().	

⌋()

[SWS_RamTst_00012] ⌈ For the type `RamTst_TestResultType` (of the overall test result), the enumeration value `RAMTST_RESULT_NOT_TESTED` shall be the default value after a reset. This enumeration value shall have the numeric value 0. ⌋(SRS_BSW_00406)

For more details on the usage of this status see chapter 7.6.

8.2.3 RamTst_AlgParamsIdType

[SWS_RamTst_00191] ⌈

Name:	`RamTst_AlgParamsIdType`		
Type:	`uint8`		
Range:	`0...255`	--	--
Description:	Data type used to identify a set of configuration parameters for a test algorithm.		

⌋()

[SWS_RamTst_00188] ⌈ For the type `RamTst_AlgParamsIdType`, the value 0 shall indicate, that no test parameters (and thus no test algorithm) is selected. This shall be the default value of the corresponding variable after reset. ⌋()

8.2.4 RamTst_AlgorithmType

[SWS_RamTst_00227] ⌈

Name:	`RamTst_AlgorithmType`	
Type:	`Enumeration`	
Range:	`RAMTST_ALGORITHM_UNDEFINED`	Undefined algorithm (uninitialized value)
	`RAMTST_CHECKERBOARD_TEST`	Checkerboard test algorithm
	`RAMTST_MARCH_TEST`	March test algorithm
	`RAMTST_WALK_PATH_TEST`	Walk path test algorithm
	`RAMTST_GALPAT_TEST`	Galpat test algorithm
	`RAMTST_TRANSP_GALPAT_TEST`	Transparent Galpat test algorithm
	`RAMTST_ABRAHAM_TEST`	Abraham test algorithm
Description:	This is a value returned by the API service RamTst_GetTestAlgorithm().	

⌋()

[SWS_RamTst_00013] ⌈ For the type `RamTst_AlgorithmType`, the enumeration value `RAMTST_ALGORITHM_UNDEFINED` shall be the default value after reset. This enumeration value shall have the numeric value 0. ⌋(SRS_BSW_00406)

[SWS_RamTst_00058] ⌈ The type `RamTst_AlgorithmType` shall contain only the enumerations of the algorithms selected at pre-compile time. ⌋(SRS_BSW_00345)

Note that if vendor specific algorithms were defined (see SWS_RamTst_00205), the enumeration fields of `RamTst_AlgorithmType` should be extended accordingly by the implementer (or by a code generator).

8.2.5 RamTst_NumberOfTestedCellsType

[SWS_RamTst_00173] ⌈

Name:	RamTst_NumberOfTestedCellsType		
Type:	uint32		
Range:	1...(2^32-1)	--	--
Description:	Data type of number of tested RAM cells		

⌋()

8.2.6 RamTst_NumberOfBlocksType

[SWS_RamTst_00174] ⌈

Name:	RamTst_NumberOfBlocksType		
Type:	uint16		
Range:	1...65535	--	--
Description:	Data type used to identify or count RAM blocks given in the test configuration parameters.		

⌋()

8.3 Function Definitions

This is a list of functions provided for upper layer modules.

8.3.1 RamTst_Init

[SWS_RamTst_00099] ⌈

Service name:	RamTst_Init
Syntax:	void RamTst_Init(void)
Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service for RAM Test initialization.

⌋(SRS_BSW_00101, SRS_BSW_00358)

Note: See also **[SWS_RamTst_00093]** in 10.2.1 Variants.

[SWS_RamTst_00007] ⌈ The function RamTst_Init shall initialize all RAM Test relevant registers and global variables and change the execution status to RAMTST_EXECUTION_STOPPED. The test is initialized to use the default test parameter set (RamTstDefaultAlgParamsId) as configured by its RamTstAlgParams container. ⌋(SRS_BSW_00101, SRS_SPAL_12057)

[SWS_RamTst_00096] ⌈ If the DET is enabled and the execution status of the RAM Test is not `RAMTST_EXECUTION_UNINIT`, the function `RamTst_Init` shall report the error value `RAMTST_E_STATUS_FAILURE` to the DET, and then immediately return. ⌋()

8.3.2 RamTst_DeInit

[SWS_RamTst_00146:] ⌈

Service name:	RamTst_DeInit
Syntax:	`void RamTst_DeInit( void )`
Service ID[hex]:	0x0c
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service for RAM Test deinitialization.

⌋(SRS_SPAL_12163)

[SWS_RamTst_00147] ⌈ The function `RamTst_DeInit` shall deinitialize all RAM Test relevant registers and global variables that were initialized by `RamTst_Init` and change the execution status to `RAMTST_EXECUTION_UNINIT`. ⌋()

If the RAM Test is in the `RAMTST_EXECUTION_UNINIT` state after a `RamTst_DeInit` call, a call to any RamTst Module function (except `RamTst_Init`) may result in unknown software behavior.

8.3.3 RamTst_Stop

[SWS_RamTst_00100] ⌈

Service name:	RamTst_Stop
Syntax:	`void RamTst_Stop( void )`
Service ID[hex]:	0x02
Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service for stopping the RAM Test.

⌋()

[SWS_RamTst_00014] ⌈ When the RamTst_Stop function is called, RamTst_MainFunction shall still finish the current atomic sequence (if it is executing), and afterward the status shall be set to RAMTST_EXECUTION_STOPPED. The test result is retained, but test parameters and loop data are not. ⌋()

[SWS_RamTst_00148] ⌈ After a RamTst_Stop call, RamTst_MainFunction shall not begin testing again when called by the scheduler until after a RamTst_Allow call. ⌋()

[SWS_RamTst_00033] ⌈ If the DET is enabled and the execution status of the RAM Test is not RAMTST_EXECUTION_RUNNING or RAMTST_EXECUTION_SUSPENDED, the function RamTst_Stop shall report the error value RAMTST_E_STATUS_FAILURE to the DET, and then immediately return. ⌋(SRS_BSW_00323, SRS_SPAL_12448)

The RamTst_Stop API can be enabled or disabled by the configuration parameter RamTstStopApi within the container RamTstCommon.

8.3.4 RamTst_Allow

[SWS_RamTst_00149] ⌈

Service name:	RamTst_Allow

Syntax:	`void RamTst_Allow(`
	`  void`
	`)`

Service ID[hex]:	0x03
Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service for continuing the RAM Test after calling 'RamTst_Stop.

⌋()

[SWS_RamTst_00169] ⌈ The function RamTst_Allow shall permit the RamTst_MainFunction to perform testing at its next scheduled call, if it had been stopped. Therefore, it shall change the execution status to RAMTST_EXECUTION_RUNNING, if it has been RAMTST_EXECUTION_STOPPED. ⌋()

[SWS_RamTst_00170] ⌈ If DET is enabled and the execution status is not RAMTST_EXECUTION_STOPPED, the function RamTst_Allow shall report the error

value RAMTST_E_STATUS_FAILURE to the DET, and then immediately return. ⌋(SRS_BSW_00323)

The RamTst_Allow API can be enabled or disabled by the configuration parameter RamTstAllowApi within the container RamTstCommon.

8.3.5 RamTst_Suspend

[SWS_RamTst_00150] ⌈

Service name:	RamTst_Suspend

Syntax:	`void RamTst_Suspend(`
	`  void`
	`)`

Service ID[hex]:	0x0d
Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service for suspending current operation of background RAM Test, until RESUME is called.

⌋()

[SWS_RamTst_00171] ⌈ The function RamTst_Suspend shall temporarily prohibit the RamTst_MainFunction from performing tests at its next scheduled call. When RamTst_Suspend is called and the execution status is RAMTST_EXECUTION_RUNNING, testing stops after the current atomic sequence, test result and current test states are retained and the execution status is changed to RAMTST_EXECUTION_SUSPENDED. ⌋()

[SWS_RamTst_00172] ⌈ If DET is enabled and the execution status is not RAMTST_EXECUTION_RUNNING, the function RamTst_Suspend shall report the error value RAMTST_E_STATUS_FAILURE to the DET, and then immediately return. ⌋(SRS_BSW_00323)

The RamTst_Suspend API can be enabled or disabled by the configuration parameter RamTstSuspendApi within the container RamTstCommon.

8.3.6 RamTst_Resume

[SWS_RamTst_00101] ⌈

Service name:	RamTst_Resume

Syntax:	`void RamTst_Resume(`
	`  void`
	`)`

Service ID[hex]:	0x0e

Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service for allowing to continue the background RAM Test at the point is was suspended.

」()

[SWS_RamTst_00018] ⌈ The function `RamTst_Resume` shall permit the `RamTst_MainFunction` to continue testing at the point where it was suspended, at its next scheduled call. Testing continues according to the saved test states. The function `RamTst_Resume` shall change the execution status to `RAMTST_EXECUTION_RUNNING` if it has been `RAMTST_EXECUTION_SUSPENDED`. 」()

[SWS_RamTst_00037] ⌈ If DET is enabled and the execution status of the RAM Test module is not `RAMTST_EXECUTION_SUSPENDED`, the function `RamTst_Resume` shall report the error value `RAMTST_E_STATUS_FAILURE` to the DET, and then immediately return. 」(SRS_BSW_00323, SRS_SPAL_12448)

The `RamTst_Resume` API can be enabled or disabled by the configuration parameter `RamTstResumeApi` within the container `RamTstCommon`.

8.3.7 RamTst_GetExecutionStatus
[SWS_RamTst_00102]

⌈

Service name:	RamTst_GetExecutionStatus	
Syntax:	`RamTst_ExecutionStatusType RamTst_GetExecutionStatus( void )`	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	RamTst_ExecutionStatusType	See type definition
Description:	Service returns the current RAM Test execution status.	

」(SRS_BSW_00331)

[SWS_RamTst_00019] ⌈ The function `RamTst_GetExecutionStatus` shall return the current RAM Test execution status. 」(SRS_RamTst_13810)

The `RamTst_GetExecutionStatus` API can be enabled or disabled by the configuration parameter RamTst `GetExecutionStatusApi` within the container `RamTstCommon`.

8.3.8 RamTst_GetTestResult

[SWS_RamTst_00103] ⌈

Service name:	RamTst_GetTestResult	
Syntax:	`RamTst_TestResultType RamTst_GetTestResult( void )`	
Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	RamTst_TestResultType	See type definition
Description:	Service returns the current RAM Test result.	

⌋(SRS_BSW_00331)

[SWS_RamTst_00024] ⌈ The function `RamTst_GetTestResult` shall return the current RAM test result. ⌋(SRS_RamTst_13810)

The test result is determined according to SWS_RamTst_00202.

The `RamTst_GetTestResult` API can be enabled or disabled by the configuration parameter `RamTstGetTestResultApi` within the container `RamTstCommon`.

8.3.9 RamTst_GetTestResultPerBlock

[SWS_RamTst_00104] ⌈

Service name:	RamTst_GetTestResultPerBlock	
Syntax:	`RamTst_TestResultType RamTst_GetTestResultPerBlock( RamTst_NumberOfBlocksType BlockID )`	
Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	BlockID	Identifies the block
Parameters (inout):	None	
Parameters (out):	None	
Return value:	RamTst_TestResultType	See type definition
Description:	Service returns the current RAM Test result for the specified block.	

⌋(SRS_RamTst_13810)

[SWS_RamTst_00038] ⌈ The function `RamTst_GetTestResultPerBlock` shall return the current RAM test result for the specified block. ⌋(SRS_RamTst_13810)

The test result per block is determined according to SWS_RamTst_00207.

[SWS_RamTst_00039] ⌈ If DET is enabled and the BlockID is out of range, the function `RamTst_GetTestResultPerBlock` shall report the error value `RAMTST_E_OUT_OF_RANGE` to the DET and return the test result value `RAMTST_RESULT_UNDEFINED`. ⌋(SRS_BSW_00323, SRS_SPAL_12448)

Hint: "Out of range" means here, that the BlockID does not match to one of the values configured for the currently selected `RamTstAlgParams/ RamtstBlockParams/ RamTstBlockId`, see ECUC_RamTst_00143.

The `RamTst_GetTestResultPerBlock` API can be enabled or disabled by the configuration parameter `RamTstGetTestResultPerBlockApi` within the container `RamTstCommon`.

8.3.10 RamTst_GetVersionInfo

[SWS_RamTst_00109]⌈

Service name:	RamTst_GetVersionInfo	
Syntax:	`void RamTst_GetVersionInfo( Std_VersionInfoType* versioninfo )`	
Service ID[hex]:	0x0a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to the location / address where to store the version information of this module.
Return value:	None	
Description:	Service returns the version information of this module.	

⌋(SRS_BSW_00407)

[SWS_RamTst_00222] ⌈ If the function `RamTst_GetVersionInfo` is called with a NULL pointer as parameter, it shall return immediately without any further action, and If DET is enabled, this function shall report the error value `RAMTST_E_PARAM_POINTER` to the DET module. ⌋()

8.3.11 RamTst_GetAlgParams

[SWS_RamTst_00193] ⌈

Service name:	RamTst_GetAlgParams
Syntax:	`RamTst_AlgParamsIdType RamTst_GetAlgParams(`

	void)
Service ID[hex]:	0x12
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	RamTst_AlgParamsIdType — See type definition.
Description:	Service returns the ID of the current RAM Test algorithm parameter set.

」()

[SWS_RamTst_00194] 「 The function `RamTst_GetAlgParams` shall return the ID of the currently selected test algorithm parameter set (i.e. the ID of the currently selected `RamTstAlgParams` in the container `RamTstConfigParams`). 」()

The `RamTst_GetAlgParams` API can be enabled or disabled by the configuration parameter `RamTstGetAlgParamsApi` within the container `RamTstCommon`.

8.3.12 RamTst_GetTestAlgorithm

[SWS_RamTst_00106] 「

Service name:	RamTst_GetTestAlgorithm
Syntax:	RamTst_AlgorithmType RamTst_GetTestAlgorithm(void)
Service ID[hex]:	0x07
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	RamTst_AlgorithmType — See type definition
Description:	Service returns the current RAM Test algorithm.

」()

[SWS_RamTst_00021] 「 The function `RamTst_GetTestAlgorithm` shall return the current RAM Test algorithm. 」()

The `RamTst_GetTestAlgorithm` API can be enabled or disabled by the configuration parameter `RamTstGetTestAlgorithmApi` within the container `RamTstCommon`.

8.3.13 RamTst_GetNumberOfTestedCells

[SWS_RamTst_00108]「

Service name:	RamTst_GetNumberOfTestedCells	
Syntax:	`RamTst_NumberOfTestedCellsType RamTst_GetNumberOfTestedCells( void )`	
Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	RamTst_NumberOfTestedCellsType	Number of currently tested cells per cycle.
Description:	Service returns the current number of tested cells per main-function cycle.	

」(SRS_RamTst_13809)

[SWS_RamTst_00034] 「 The function `RamTst_GetNumberofTestedCells` shall read the current number of tested cells per cycle. 」()

The `RamTst_ GetNumberOfTestedCells` API can be enabled or disabled by the configuration parameter `RamTstGetNumberOfTestedCellsApi` within the container `RamTstCommon`.

8.3.14 RamTst_SelectAlgParams

RamTst105: 「

Service name:	RamTst_SelectAlgParams	
Syntax:	`void RamTst_SelectAlgParams( RamTst_AlgParamsIdType NewAlgParamsId )`	
Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	NewAlgParamsId	Identifies the parameter set to be used.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service used to set the test algorithm and its parameter set.	

」(SRS_RamTst_13804)

[SWS_RamTst_00083] 「 The function `RamTst_SelectAlgParams` shall select the test parameter set (i.e. one of the `RamTstAlgParams` in the container `RamTstConfigParams`) to be used by the RAM Test module. 」(SRS_RamTst_13804)

Note: Depending on the configured content of `RamTstAlgParams`, this function may be used to select a different test algorithm. But the function may also be used to

select a different set of blocks (e.g. for foreground testing) for the same test algorithm.

[SWS_RamTst_00085] ⌈ The function `RamTst_SelectAlgParams` shall re-initialize all RAM Test relevant registers and global variables with the values for the "`NewAlgParamsId`". ⌋()

[SWS_RamTst_00084] ⌈ If DET is enabled and the parameter "`NewAlgParamsId`" is out of range, the function `RamTst_SelectAlgParams` shall report the error value `RAMTST_E_OUT_OF_RANGE` to the DET, leaving the current `RamTstAlgParams` unchanged. ⌋(SRS_SPAL_12448)

Hint: "Out of range" means, that the "`NewAlgParamsId`" does not match to one of the configured values for `RamTstAlgParams/RamTstAlgParamsId`, see ECUC_RamTst_00179.

[SWS_RamTst_00097] ⌈ If DET is enabled and the execution status of the RAM Test module is not `RAMTST_EXECUTION_STOPPED`, the function `RamTst_SelectAlgParams` shall report the error value `RAMTST_E_STATUS_FAILURE` to the DET, then immediately return from the function. ⌋(SRS_BSW_00323)

[SWS_RamTst_00094] ⌈ The function `RamTst_SelectAlgParams` shall initialize the test result status (according to SWS_RamTst_00207 and SWS_RamTst_00202). ⌋()

Hint: It makes no sense to keep the previous test results at this point (as it was specified in former version of this document), since the block structure might change due to the selected parameter set, so the previous result could no longer be interpreted. If the test environment wants to save the previous results, it can easily retrieve them via `RamTst_GetTestResult` or `RamTst_GetTestResultPerBlock` before calling `RamTst_SelectAlgParams`.

The function `RamTst_SelectAlgParams` also requires the configuration parameter `RamTstSelectAlgParams Api` within the container `RamTstCommon`.

8.3.15 RamTst_ChangeNumberOfTestedCells

[SWS_RamTst_00107] ⌈

Service name:	RamTst_ChangeNumberOfTestedCells
Syntax:	`void RamTst_ChangeNumberOfTestedCells( RamTst_NumberOfTestedCellsType NewNumberOfTestedCells )`
Service ID[hex]:	0x08
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant

Parameters (in):	NewNumberOfTestedCells	See type definition
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service changes the current number of tested cells.	

」(SRS_RamTst_13800, SRS_RamTst_13809)

[SWS_RamTst_00036] 「 The function `RamTst_ChangeNumberOfTestedCells` shall change the current number of tested cells (for background tests). 」(SRS_RamTst_13800)

[SWS_RamTst_00040] 「 If DET is enabled and the parameter `NewNumberOfTestedCells` is out of range 」(SRS_BSW_00323, SRS_SPAL_12448)
(min= `MinNumberOfTestedCells` / max = `MaxNumberOfTestedCells`),
the function `RamTst_ChangeNumberOfTestedCells` shall report the error value `RAMTST_E_OUT_OF_RANGE` to the DET. The function shall leave the number of tested cells unchanged.

[SWS_RamTst_00095] 「 If the execution status of the RAM Test module is not in the status `RAMTST_EXECUTION_STOPPED`, the function `RamTst_ChangeNumberOfTestedCells` shall not change the current number of tested cells and (if DET is enabled) shall report the error value `RAMTST_E_STATUS_FAILURE` to the DET. 」(SRS_BSW_00323, RS_SPAL_12448)

The `RamTst_ ChangeNumberOfTestedCells` API can be enabled or disabled by the configuration parameter `RamTstChangeNumOfTestedCellsApi` within the container `RamTstCommon`.

8.3.16 RamTst_RunFullTest

[SWS_RamTst_00195] 「

Service name:	RamTst_RunFullTest
Syntax:	`void RamTst_RunFullTest( void )`
Service ID[hex]:	0x10
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service for executing the full RAM Test in the foreground.

」()

[SWS_RamTst_00196] ⌈ If the RAM Test execution status is `RAMTST_EXECUTION_STOPPED`, the function `RamTst_RunFullTest` shall test all RAM blocks defined in the selected `RamTstAlgParams`. ⌋()

[SWS_RamTst_00210] ⌈ If DET is enabled and the execution status of the RAM Test module is not `RAMTST_EXECUTION_STOPPED`, the function `RamTst_RunFullTest` shall report the error value `RAMTST_E_STATUS_FAILURE` to the DET, and then immediately return. ⌋(SRS_BSW_00323)

[SWS_RamTst_00211] ⌈ If the RAM Test execution status is `RAMTST_EXECUTION_STOPPED`, the function `RamTst_RunFullTest` shall set the execution status to `RAMTST_EXECUTION_RUNNING` during the test and set it back to `RAMTST_EXECUTION_STOPPED` before returning. ⌋()

[SWS_RamTst_00212] ⌈ The function `RamTst_RunFullTest` shall update the test result status of single blocks according to SWS_RamTst_00207. ⌋()

[SWS_RamTst_00213] ⌈ The function `RamTst_RunFullTest` shall update the overall test result status according to SWS_RamTst_00202. If at least one block test result is `RAMTST_RESULT_NOT_OK`, then the function shall report the production error `RAMTST_E_RAM_FAILURE` to the DEM. ⌋(SRS_BSW_00339)

The function `RamTst_RunFullTest` requires the configuration `parameter RamTstRunFullTestApi in the container RamTstCommon`.

Destruction or restoration of the memory content are handled according to the requirements SWS_RamTst_00200 and SWS_RamTst_00201.

For pre-conditions on the function `RamTst_RunFullTest`, see requirement SWS_RamTst_00002.

Implementation Hints:

For reasons of effiency and optimum fault coverage, the implementation of `RamTst_RunFullTest` shall assume, that it has exclusive access to all memory blocks contained in its `RamTstAlgParams` during the call. This allows to apply the test algorithm on a wider range of memory than in background test, which increases the fault coverage especially for coupling faults.

Thus it is the responsibility of the test environment, to either provide appropriate resource locking, or to call the function in an ECU mode, where the memory blocks of the selected `RamTstAlgParams` are not in use. The test environment must also ensure, that `RamTst_MainFunction` is not scheduled during the foreground test.

A test algorithm usually requires various write and read cycles over a given range of memory. Some algorithms also require multiple walks through this range. It is up to the implementation, whether such a tested range corresponds to one block (which means, that the full test is split into several ranges) or even includes several or all blocks. This depends on performance issues and the assumed fault model. In any case, the test behavior must be clearly documented.

8.3.17 RamTst_RunPartialTest

[SWS_RamTst_00197] ⌈

Service name:	RamTst_RunPartialTest
Syntax:	`void RamTst_RunPartialTest( RamTst_NumberOfBlocksType BlockId )`
Service ID[hex]:	0x11
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	BlockId: Identifies the single RAM block to be tested in the selected set of RamTstAlgParams.
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Service for testing one RAM block in the foreground.

⌋()

[SWS_RamTst_00198] ⌈ If the RAM Test execution status is `RAMTST_EXECUTION_STOPPED` or `RAMTST_EXECUTION_SUSPENDED`, the function `RamTst_RunPartialTest` shall test the specified RAM block. ⌋()

[SWS_RamTst_00214] ⌈ If DET is enabled and the RAM test execution status is neither `RAMTST_EXECUTION_STOPPED` nor `RAMTST_EXECUTION_SUSPENDED`, the function `RamTst_RunPartialTest` shall report the error value `RAMTST_E_STATUS_FAILURE` to the DET, and then immediately return. ⌋(SRS_BSW_00323)

[SWS_RamTst_00215] ⌈ If the RAM Test execution status is `RAMTST_EXECUTION_STOPPED` or `RAMTST_EXECUTION_SUSPENDED`, the function `RamTst_RunPartialTest` shall set the execution status to `RAMTST_EXECUTION_RUNNING` during the test, and after the test shall set it back to the previous state (the state when the function was called). ⌋()

[SWS_RamTst_00216] ⌈ The function `RamTst_RunPartialTest` shall update the test result status of the tested block according to SWS_RamTst_00207. If this block test result is `RAMTST_RESULT_NOT_OK`, then the function shall report the production error `RAMTST_E_RAM_FAILURE` to the DEM. ⌋(SRS_BSW_00339)

[SWS_RamTst_00217] ⌈ A successful partial foreground test shall set the block specific result to `RAMTST_RESULT_OK`. It shall not modify the overall test result. A failing partial foreground test shall set both, the block specific as well as the overall test result to `RAMTST_RESULT_NOT_OK`. ⌋()

[SWS_RamTst_00223] ⌈ If DET is enabled and the `BlockId` is out of range, the function `RamTst_RunPartialTest` shall report the error value `RAMTST_E_OUT_OF_RANGE` to the DET and then immediately return. ⌋()

Notes:

'Out of range' in [**SWS_RamTst_00223**] means that the `BlockId` does not match to one of the configured values for the currently selected `Block Identifier` (RamTstAlgParams/ RamtstBlockParams/ RamTstBlockId).

Updating the test results will overwrite the result from a previous test of this block and the overall test result in case of failure, including the result from a suspended background test.

The function `RamTst_RunPartialTest` requires the configuration `parameter RamTstRunPartialTestApi in the container RamTstCommon`.

Destruction or restoration of the memory content is handled according to the requirements SWS_RamTst_00200 and SWS_RamTst_00201.

For pre-conditions on the function `RamTst_RunPartialTest`, see requirement SWS_RamTst_00002.

Implementation Hints:

The implementation hints given for `RamTst_RunFullTest` also apply here, as far as applicable to one single block.

8.4 Callback Notifications

Since the RAM Test is a driver module, it does not implement any callback functions from lower layer modules.

8.5 Scheduled Functions

For details refer to the chapter 8.5 "Scheduled functions" in *SWS_BSWGeneral*

.

8.5.1 RamTst_MainFunction

[SWS_RamTst_00110] ⌈

Service name:	RamTst_MainFunction

Syntax:	`void RamTst_MainFunction(`
	`  void`
	`)`

Service ID[hex]:	0x01

Description:	Scheduled function for executing the RAM Test in the background.

⌋(SRS_BSW_00373, SRS_BSW_00376)

[SWS_RamTst_00008] ⌈ If the RAM Test execution status is `RAMTST_EXECUTION_RUNNING`, the function `RamTst_MainFunction` shall continue to test the RAM blocks defined in the selected `RamTstAlgParams`. ⌋(SRS_RamTst_13809)

[SWS_RamTst_00009] ⌈ If the RAM Test execution status is `RAMTST_EXECUTION_RUNNING` and if no blocks have yet been tested (first call of the function), then the function `RamTst_MainFunction` shall start testing with the first configured RAM block in the selected `RamTstAlgParams`. ⌋()

[SWS_RamTst_00175] ⌈ If the RAM Test execution status is not `RAMTST_EXECUTION_RUNNING` when this API is called, the function `RamTst_MainFunction` shall return immediately without any actions. ⌋(SRS_BSW_00450)

[SWS_RamTst_00010] ⌈ The function `RamTst_MainFunction` shall update the test result status of single blocks according to SWS_RamTst_00207. ⌋(SRS_RamTst_13810)

[SWS_RamTst_00011] ⌈ The function `RamTst_MainFunction` shall update the overall test result status according to SWS_RamTst_00202. If at least one block test result is `RAMTST_RESULT_NOT_OK`, then the function shall report the production error `RAMTST_E_RAM_FAILURE` to the DEM. ⌋(SRS_BSW_00339, SRS_RamTst_13810)

[SWS_RamTst_00047] ⌈ After the function `RamTst_MainFunction` has completed testing all RAM blocks configured in the selected `RamTstAlgParams`, the next call of the function `RamTst_MainFunction` shall restart the test from the beginning. ⌋()

[SWS_RamTst_00059] ⌈ The function `RamTst_MainFunction` shall test the defined number of RAM cells within one call. The defined number is specified by the function `RamTst_ChangeNumberOfTestedCells` or by initialization. ⌋(SRS_RamTst_13809)

Notes:

Updating the test results will overwrite the result from a previous test of the current block and the overall test result, including the case that the background test was resumed after a partial foreground test of the current block.

Destruction or restoration of the memory content are handled according to the requirements SWS_RamTst_00200 and SWS_RamTst_00201.

For pre-conditions on the function `RamTst_MainFunction`, see requirement SWS_RamTst_00002.

Implementation Hints:

In general, the actual test algorithm within one call of `RamTst_MainFunction` must be performed within one or more atomic sequences. Only within one atomic sequence, the memory written by the algorithm is allowed to be corrupted during the test. This means, that the algorithm can be applied only to those cells accessed within one atomic sequence, so that the detection of coupling faults between cells (by background test) is restricted to those cells which are included in one atomic sequence.

An atomic sequence can either be declared as exclusive area via the BSW module description (see [3], SRS_BSW_00426), leaving the actual locking method to the BSW Scheduler, or be directly implemented via interrupt locking (see [3], SRS_BSW_00429). The latter is allowed, because the RAM test module belongs to the MCAL layer.

8.6 Expected Interfaces

In this chapter, all interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

This chapter defines all interfaces which are required to fulfill the core functionality of the module.

[SWS_RamTst_00111] ⌈

API function	*Description*
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by BSW modules). The interface has an asynchronous behavior, because the processing of the event is done within the Dem main function. OBD Events Suppression shall be ignored for this computation.

⌋(SRS_BSW_00339)

8.6.2 Optional Interfaces

This chapter defines all interfaces, which are required to fulfill an optional functionality of the module.

[SWS_RamTst_00112] ⌈

API function	*Description*
Det_ReportError	Service to report development errors.

」()

8.6.3 Configurable Interfaces

In this chapter, all interfaces are listed where the target function could be configured. The target function is usually a callback function.

Terms and definitions:
Reentrant: interface is expected to be reentrant

Don't care: reentrancy of interface not relevant for this module (in general, it is in this case not reentrant).

[SWS_RamTst_00043] 「 The callback notifications shall have no parameters and no return value. 」(SRS_SPAL_00157, SRS_SPAL_12056, SRS_RamTst_13820)

[SWS_RamTst_00044] 「 If a callback notification is configured as null pointer, the RAM Test module shall not execute the callback. 」(SRS_SPAL_00157, SRS_SPAL_12056, SRS_RamTst_13820)

8.6.3.1 RamTst_TestCompletedNotification

[SWS_RamTst_00113]「

Service name:	RamTst_TestCompletedNotification

Syntax:	`void RamTst_TestCompletedNotification(`
	`  void`
	`)`

Sync/Async:	--
Reentrancy:	Don't care
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	The function RamTst_TestCompleted shall be called every time when all RAM blocks of the current test configuration have been tested in the background.

」(SRS_RamTst_13820)

[SWS_RamTst_00045] 「 The RAM Test module shall call the callback `RamTst_TestCompletedNotification` every time when it has tested all RAM blocks of the selected parameter set in a background test. 」(SRS_SPAL_00157, SRS_RamTst_13820)

The callback notification `RamTst_TestCompleted` requires configuration of the parameter `RamTstTestCompletedNotification` within the container `RamTstConfigParams`.

8.6.3.2 RamTst_ErrorNotification

[SWS_RamTst_00114] ⌈

Service name:	RamTst_ErrorNotification
Syntax:	`void RamTst_ErrorNotification( void )`
Sync/Async:	--
Reentrancy:	Don't care
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	The function RamTst_Error shall be called every time when a RAM failure has been detected by the selected test algorithm in the background.

⌋(SRS_RamTst_13820)

[SWS_RamTst_00046] ⌈ The RAM test module shall call the callback `RamTst_ErrorNotification` every time when the test algorithm of the selected parameter set has detected a RAM failure in a background test.

⌋(SRS_SPAL_00157, SRS_RamTst_13820)

The callback notification `RamTst_Error_Notification` requires configuration of the parameter `RamTstTestErrorNotification` within the container `RamTstConfigParams`.

9 Sequence Diagrams

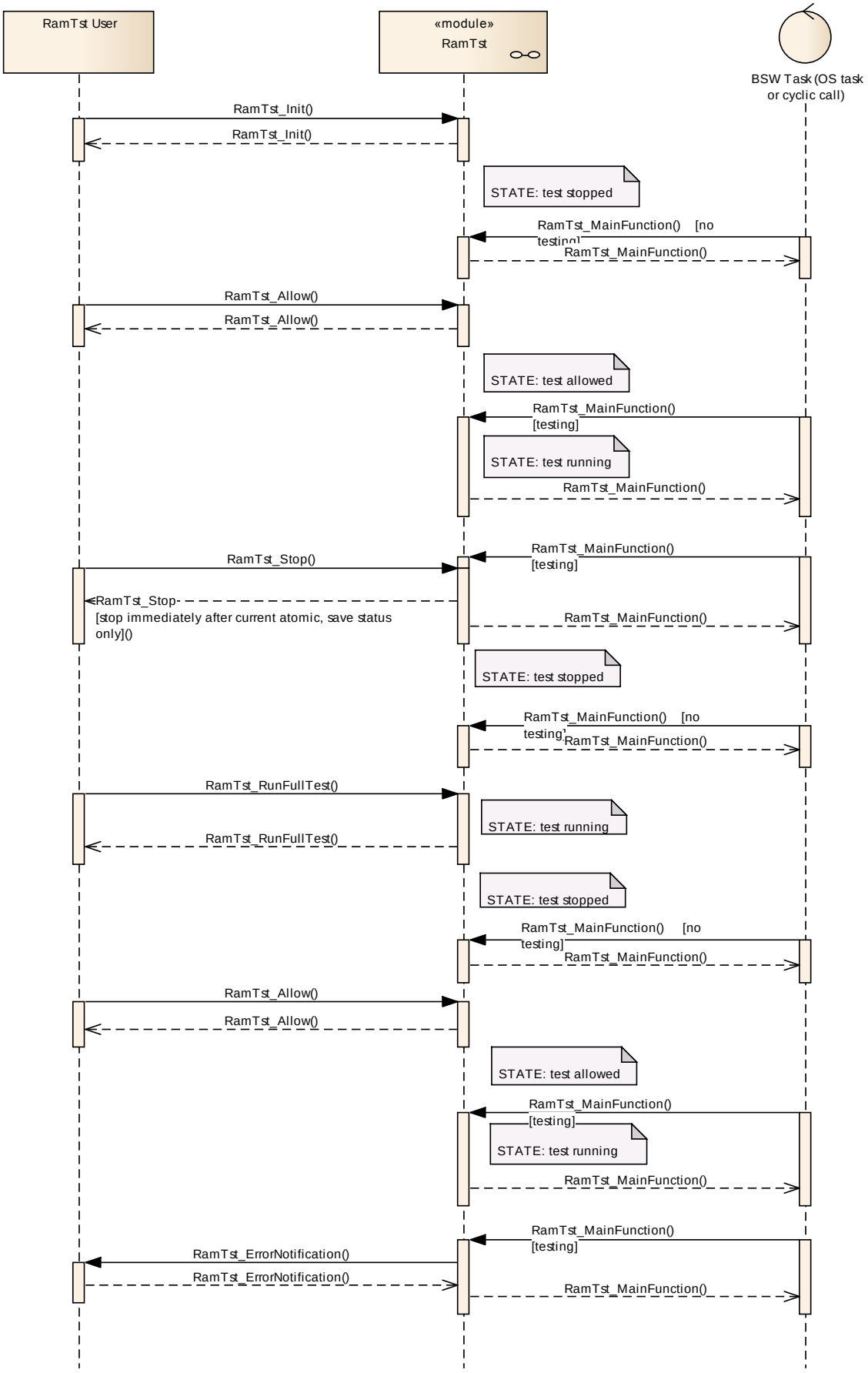
9.1 RamTst_MainFunction (Examples)

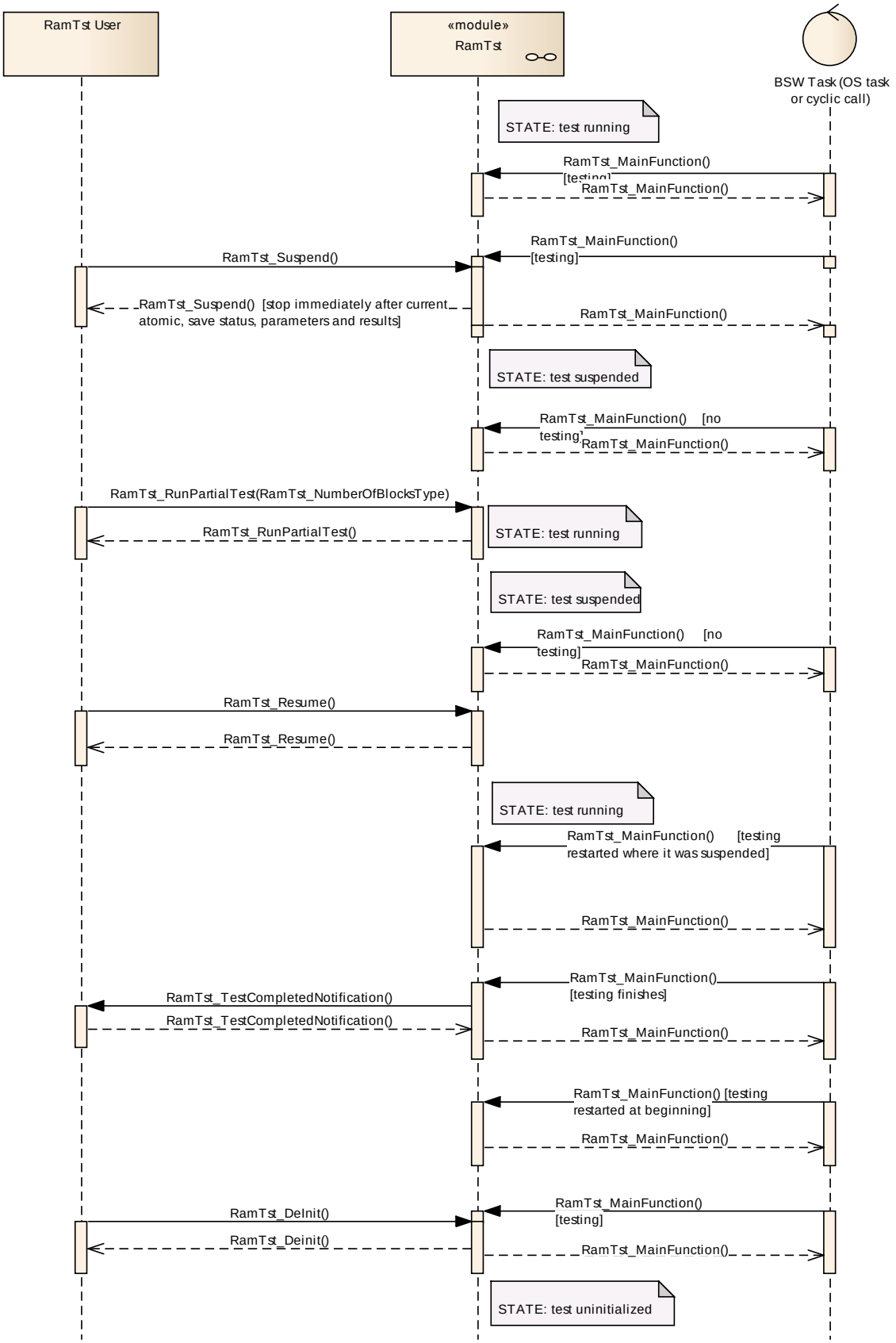
The first example sequence shows the initialization of the RAM Test module, a foreground Run Full Test request, error notification, and the cyclic call background testing.

A cyclic background task called by a scheduler consists of several small atomic sequences in succession. At the end of each atomic sequence, the command variables are checked to see if any command has been received, and corresponding actions are taken.

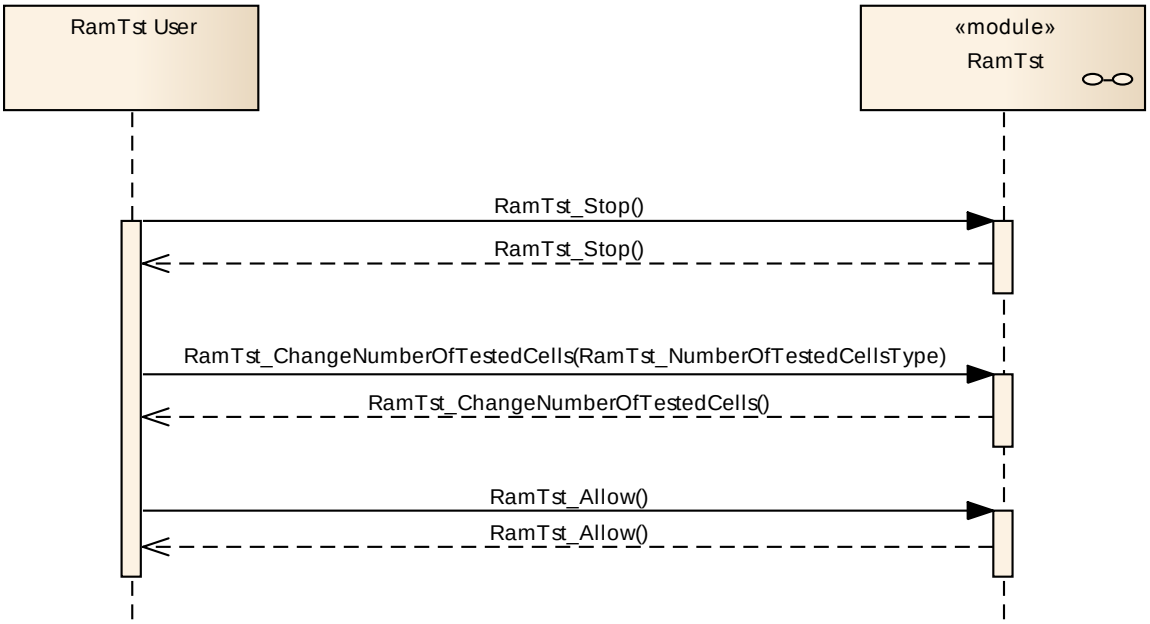
The stop request is handled following the currently running atomic sequence of the main routine, or at the next cyclic call of the main routine if it is not currently running. The allow request is handled at the next cyclic call of the main routine.

The second example shows the Suspend and Resume commands, a foreground Run Partial Test request, a Test Completed notification, and the DeInit command.

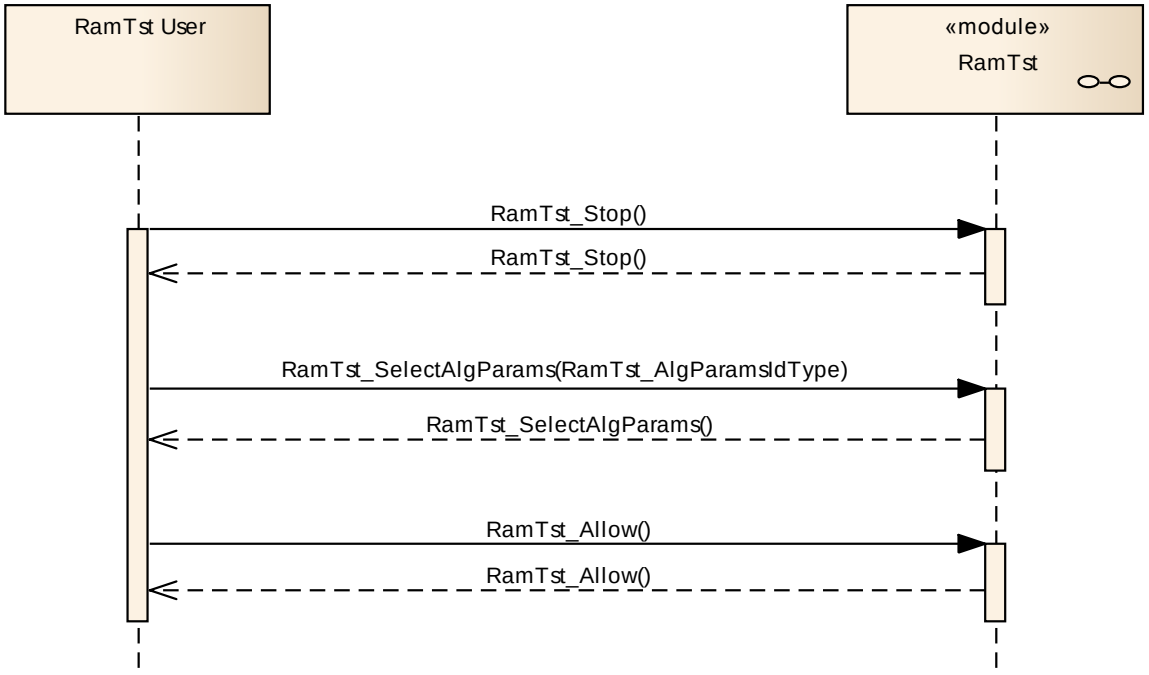

RamTst User
«module»
RamTst
BSW Task (OS task or cyclic call)
RamTst_Init()
RamTst_Init()
STATE: test stopped
RamTst_MainFunction() [no testing]
RamTst_MainFunction()
RamTst_Allow()
RamTst_Allow()
STATE: test allowed
RamTst_MainFunction() [testing]
STATE: test running
RamTst_MainFunction()
RamTst_Stop()
RamTst_Stop- [stop immediately after current atomic, save status only]()
STATE: test stopped
RamTst_RunFullTest()
RamTst_RunFullTest()
STATE: test running
STATE: test stopped
RamTst_Allow()
RamTst_Allow()
STATE: test allowed
RamTst_MainFunction() [testing]
STATE: test running
RamTst_MainFunction()
RamTst_ErrorNotification()
RamTst_ErrorNotification()


RamTst User
«module»
RamTst
BSW Task (OS task or cyclic call)
STATE: test running
RamTst_MainFunction() [testing]
RamTst_MainFunction()
RamTst_MainFunction() [testing]
RamTst_Suspend()
RamTst_Suspend() [stop immediately after current atomic, save status, parameters and results]
RamTst_MainFunction()
STATE: test suspended
RamTst_MainFunction() [no testing]
RamTst_MainFunction()
RamTst_RunPartialTest(RamTst_NumberOfBlocksType)
RamTst_RunPartialTest()
STATE: test running
STATE: test suspended
RamTst_MainFunction() [no testing]
RamTst_MainFunction()
RamTst_Resume()
RamTst_Resume()
STATE: test running
RamTst_MainFunction() [testing restarted where it was suspended]
RamTst_MainFunction()
RamTst_MainFunction() [testing finishes]
RamTst_TestCompletedNotification()
RamTst_TestCompletedNotification()
RamTst_MainFunction()
RamTst_MainFunction() [testing restarted at beginning]
RamTst_MainFunction()
RamTst_DeInit()
RamTst_Deinit()
RamTst_MainFunction() [testing]
RamTst_MainFunction()
STATE: test uninitialized

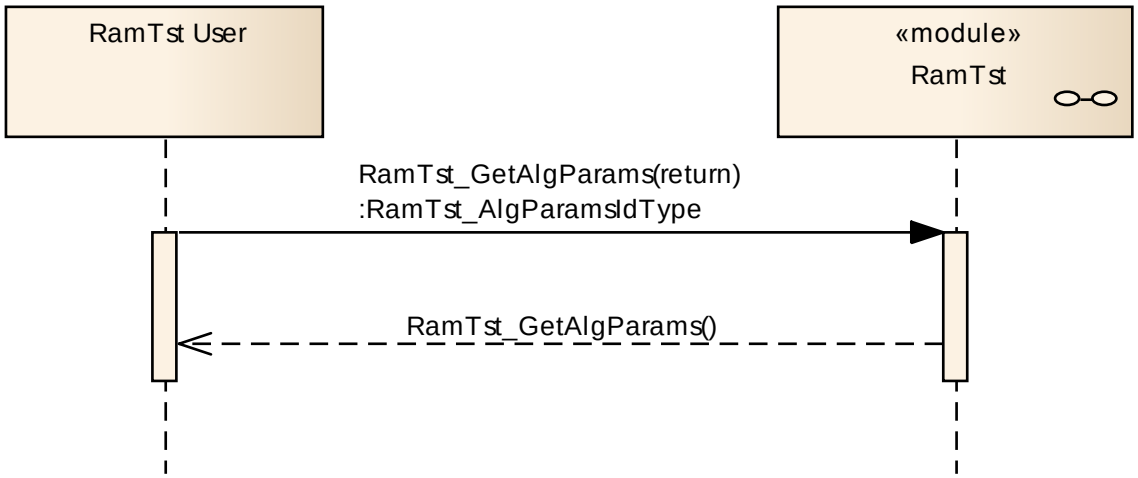
9.2 RamTst_ChangeNumberOfTestedCells



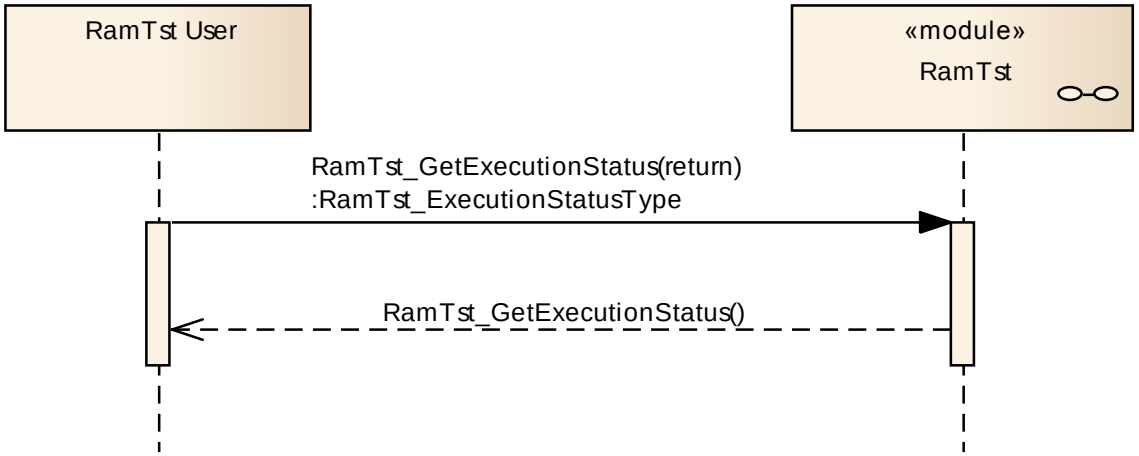
9.3 RamTst_SelectAlgParams



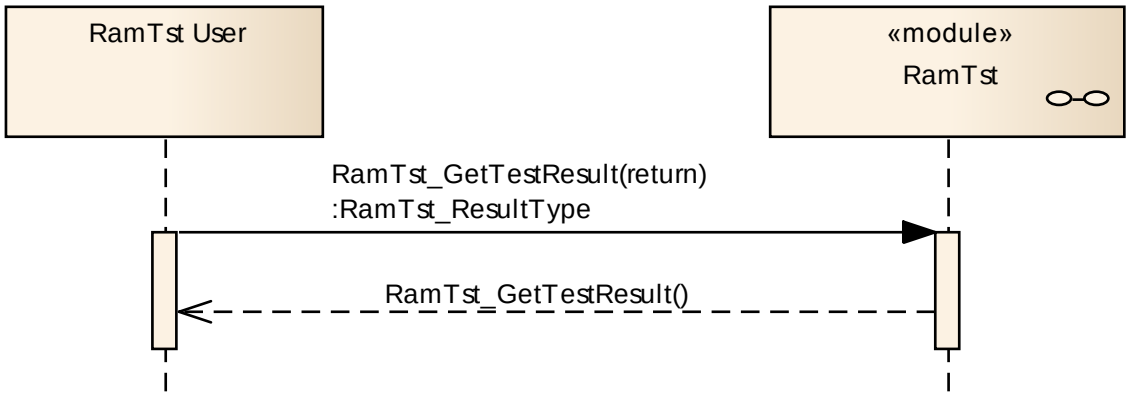
9.4 RamTst_GetAlgParams



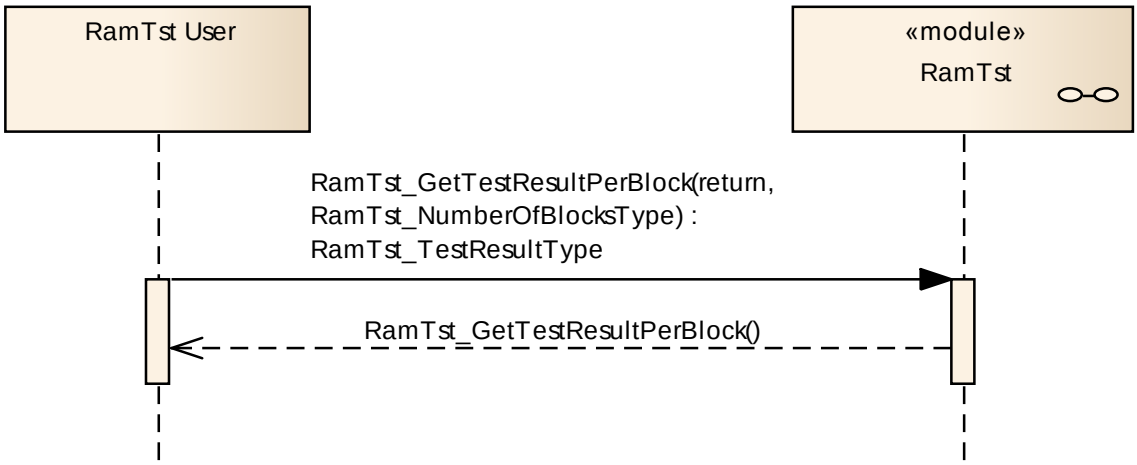
9.5 RamTst_GetExecutionStatus



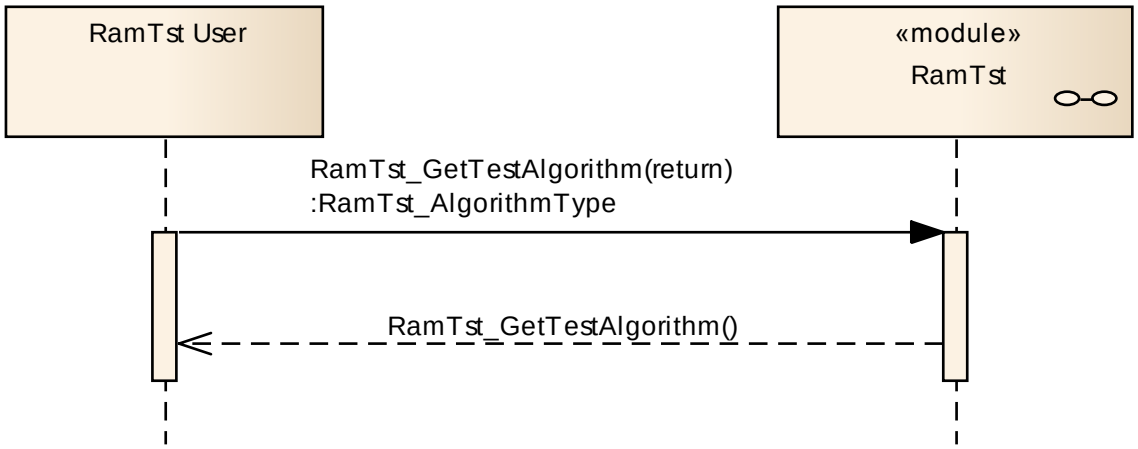
9.6 RamTst_GetTestResult



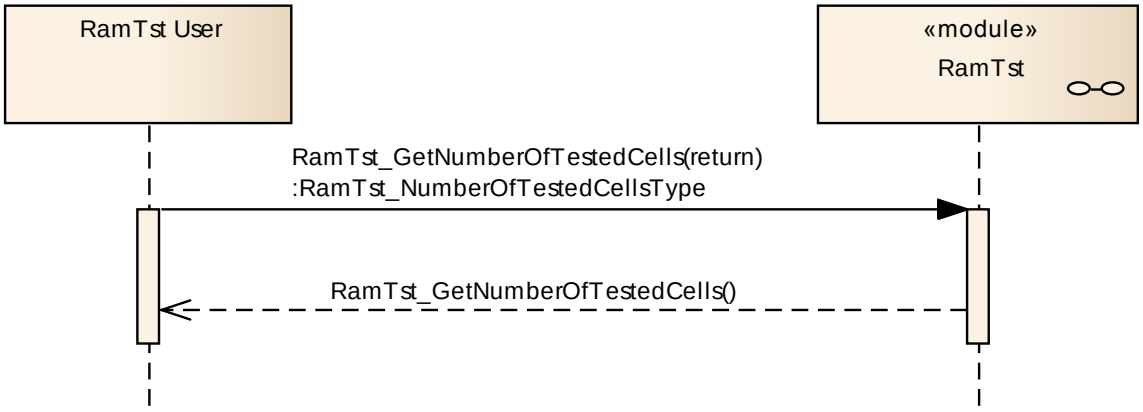
9.7 RamTst_GetTestResultPerBlock



9.8 RamTst_GetTestAlgorithm



9.9 RamTst_GetNumberOfTestedCells



10 Configuration Specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification, Chapter 10.1 describes fundamentals. It also specifies a template (table) that shall be used for the parameter specification. It is intended to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module RAM Test.

Chapter 10.3 specifies published information of the module RAM Test.

Chapter 10.4 contains additional information for the module RAM Test.

10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in SWS_BSWGeneral

10.2 Containers and Configuration Parameters

The following sections summarize the containers of RAM Test configuration parameters. The detailed descriptions of the configuration parameters are described in Chapter 8 API Specification.

[SWS_RamTst_00026] ⌈ Within the configuration data for the RAM Test module, there shall be a set of configuration containers named `RamTstAlgParams`. Each one defines a possible test parameter set to be selected at runtime, which includes an algorithm. The algorithms included in `RamTstAlgParams` are restricted to those that were pre-compile selected to be available to the user via the container `RamTstAlgorithms`.⌋(SRS_BSW_00344, SRS_SPAL_12057, SRS_SPAL_12263, SRS_RamTst_13802, SRS_RamTst_13803, SRS_RamTst_13809)

[SWS_RamTst_00027] ⌈ Within the configuration data for the RAM Test module, each parameter set of type `RamTstAlgParams` (as required in SWS_RamTst_00026) shall also have memory block configuration containers. The memory block configuration container is defined in `RamTstBlockParams`. The number of memory block configuration containers is defined by the integrator according to the RAM test strategy. ⌋(SRS_BSW_00344, SRS_SPAL_12057, SRS_SPAL_12263, SRS_RamTst_13803)

10.2.1 Variants

[SWS_RamTst_00167] ⌈ This module shall support the configuration variant VARIANT-PRE-COMPILE. Only parameters with "Pre-compile time" configuration are allowed in this variant. The intention of this variant is to optimize the parameter configuration for a source code delivery. See SRS_BSW_00397. ⌋()

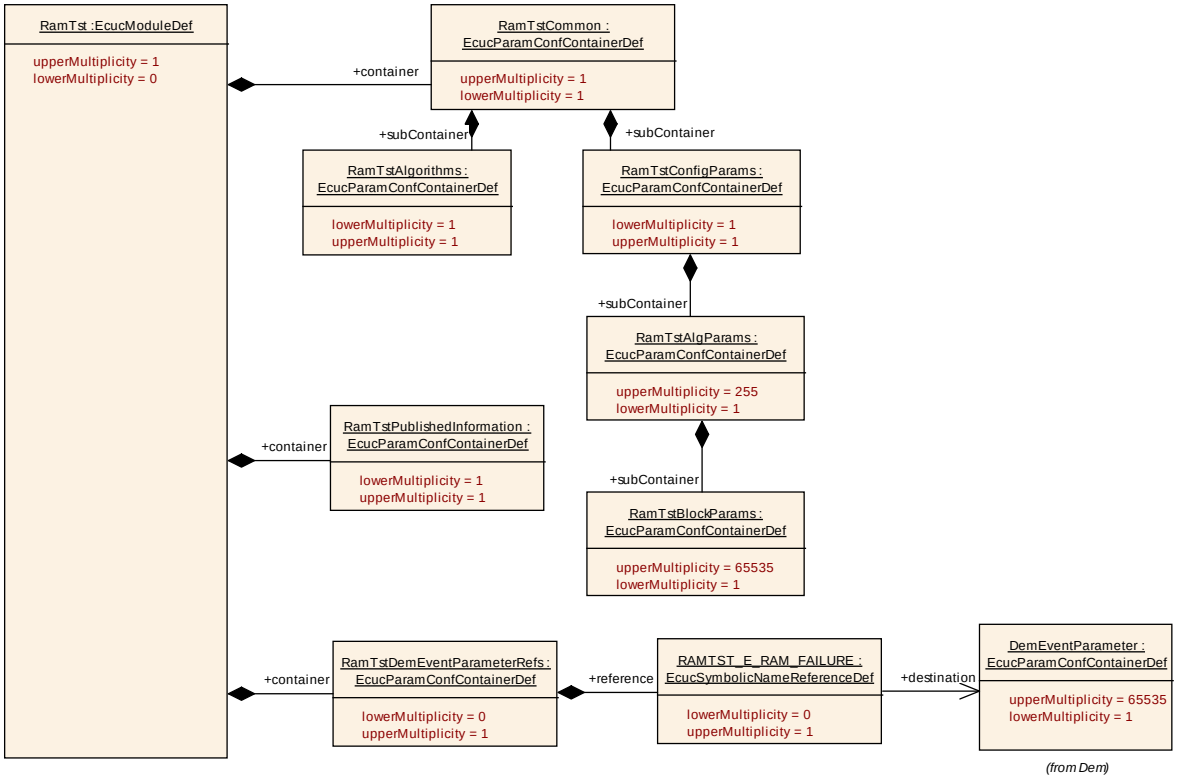
[SWS_RamTst_00168] ⌈ This module shall support the configuration variant VARIANT-LINK-TIME. Parameters with "Pre-compile time" and "Link time" are allowed in this variant. The intention of this variant is to optimize the parameter configuration for an object code delivery. See SRS_BSW_00398. ⌋()

[SWS_RamTst_00093] ⌈ The initialization function of this module shall always have a “void” as parameter. This means that, in contradiction to SRS_BSW_00414 only one interface for initialization shall be implemented and it shall not depend on the modules configuration which interface the calling software module shall use. ⌋(SRS_BSW_00414)

10.2.2 RamTst

SWS Item	**ECUC_RamTst_00150 :**
Module Name	*RamTst*
Module Description	Configuration of the RamTst module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
RamTstCommon	1	This container holds a list of all available functions in the RamTst module. Each function is turned ON or OFF before compiling so that only the desired functions and test algorithms are in the compiled code.
RamTstDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor-specific error references.
RamTstPublishedInformation	1	Container holding all RamTst specific published information parameter.



10.2.3 RamTstDemEventParameterRefs

SWS Item	**ECUC_RamTst_00188 :**
Container Name	RamTstDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor-specific error references.
Configuration Parameters	

SWS Item	**ECUC_RamTst_00189 :**		
Name	RAMTST_E_RAM_FAILURE		
Description	Reference to the DemEventParameter which shall be issued when the error "RAM failure" has occurred.		
Multiplicity	0..1		
Type	Symbolic name reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.4 RamTstCommon

SWS Item	**ECUC_RamTst_00070 :**
Container Name	RamTstCommon{RamTst_Common}
Description	This container holds a list of all available functions in the RamTst module. Each function is turned ON or OFF before compiling so that only the desired functions and test algorithms are in the compiled code.

Configuration Parameters			

SWS Item	**ECUC_RamTst_00120 :**		
Name	RamTstAllowApi {RAMTST_ALLOW_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_Allow".		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00118 :**		
Name	RamTstChangeNumOfTestedCellsApi {RAMTST_CHANGE_NUMBER_OF_TESTED_CELLS_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_ChangeNumberOfTestedCells".		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00121 :**		
Name	RamTstDevErrorDetect {RAMTST_DEV_ERROR_DETECT}		
Description	Preprocessor switch to select the development error tracer (DET) ON or OFF		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00183 :**		
Name	RamTstGetAlgParamsApi {RAMTST_GET_ALG_PARAMS_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_GetAlgParams".		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00122 :**		
Name	RamTstGetExecutionStatusApi {RAMTST_GET_EXECUTION_STATUS_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_GetExecutionStatus"		
Multiplicity	1		

Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00123 :**		
Name	RamTstGetNumberOfTestedCellsApi {RAMTST_GET_NUMBER_OF_TESTED_CELLS_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_GetNumberOfTestedCells".		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00124 :**		
Name	RamTstGetTestAlgorithmApi {RAMTST_GET_TEST_ALGORITHM_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_GetTestAlgorithm"		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00125 :**		
Name	RamTstGetTestResultApi {RAMTST_GET_TEST_RESULT_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_GetTestResult"		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00126 :**		
Name	RamTstGetTestResultPerBlockApi {RAMTST_GET_TEST_RESULT_PER_BLOCK_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_GetTestResultPerBlock"		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_RamTst_00128 :		
Name	RamTstGetVersionInfoApi {RAMTST_GET_VERSION_INFO_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_GetVersionInfo"		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_RamTst_00155 :		
Name	RamTstResumeApi {RAMTST_RESUME_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_Resume".		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_RamTst_00184 :		
Name	RamTstRunFullTestApi {RAMTST_RUN_FULL_TEST_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_RunFullTest"		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_RamTst_00185 :		
Name	RamTstRunPartialTestApi {RAMTST_RUN_PARTIAL_TEST_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_RunPartialTest"		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

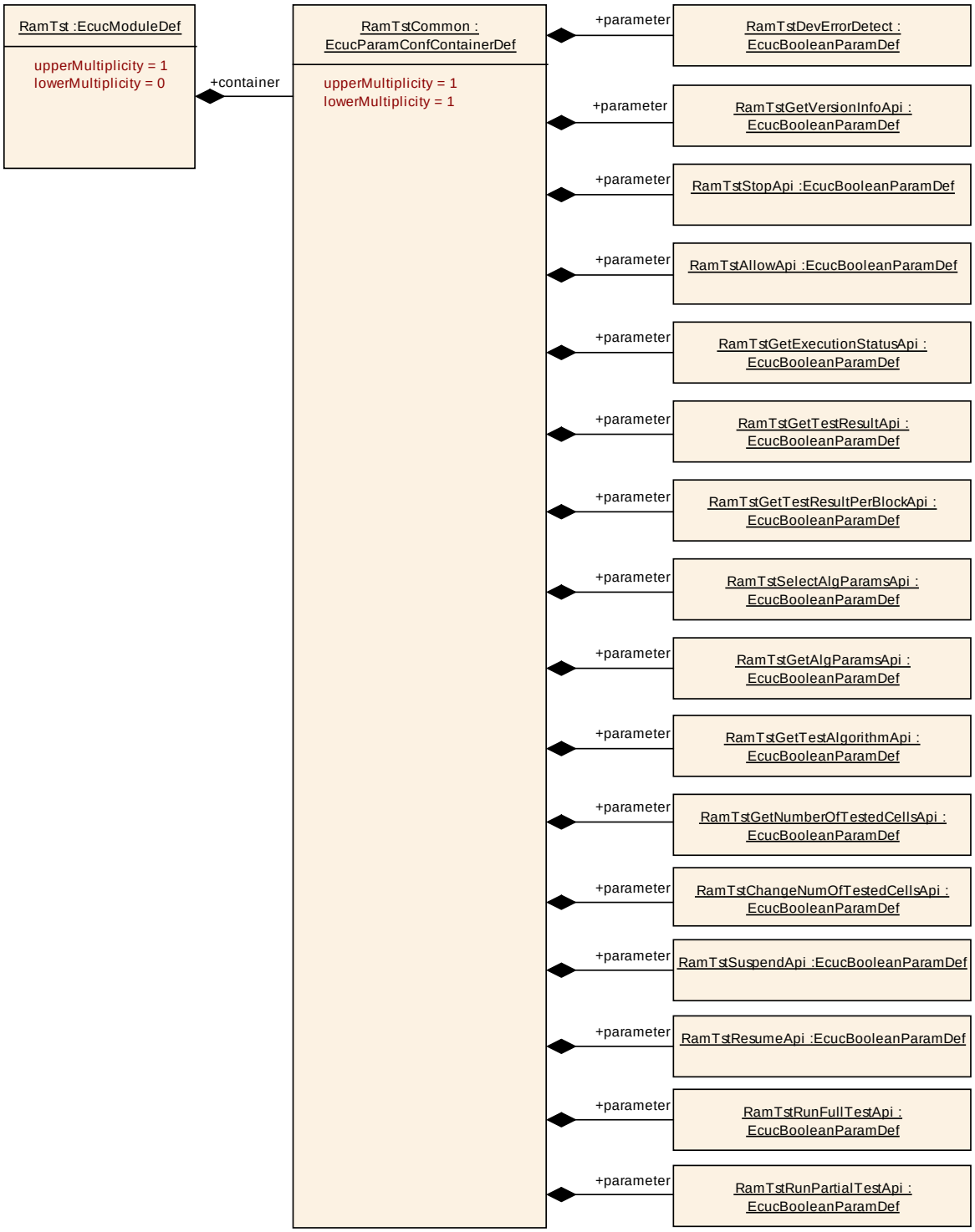
SWS Item	ECUC_RamTst_00182 :		
Name	RamTstSelectAlgParamsApi {RAMTST_SELECT_ALG_PARAMS_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_SelectAlgParams".		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	

	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00127 :**		
Name	RamTstStopApi {RAMTST_STOP_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_Stop"		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00156 :**		
Name	RamTstSuspendApi {RAMTST_SUSPEND_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_Suspend".		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
RamTstAlgorithms	1	This container holds all of the available test algorithms for the specific microcontroller. Each test algorithm is selected ON or OFF before compiling so that only the desired test algorithms are in the compiled code.
RamTstConfigParams	1	This container specifies configuration parameters which are set at pre-compile or link time.



10.2.5 RamTstAlgorithms

SWS Item	**ECUC_RamTst_00065 :**
Container Name	RamTstAlgorithms{RamTst_Algorithms}
Description	This container holds all of the available test algorithms for the specific microcontroller. Each test algorithm is selected ON or OFF before compiling so that only the desired test algorithms are in the compiled code.
Configuration Parameters	

SWS Item	**ECUC_RamTst_00129 :**

Name	RamTstAbrahamTestSelected {RAMTST_ABRAHAM_TEST_SELECTED}		
Description	Preprocessor switch to select the Abraham Test ON or OFF		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00130 :**		
Name	RamTstCheckerboardTestSelected {RAMTST_CHECKERBOARD_TEST_SELECTED}		
Description	Preprocessor switch to select the Checkerboard Test ON or OFF		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

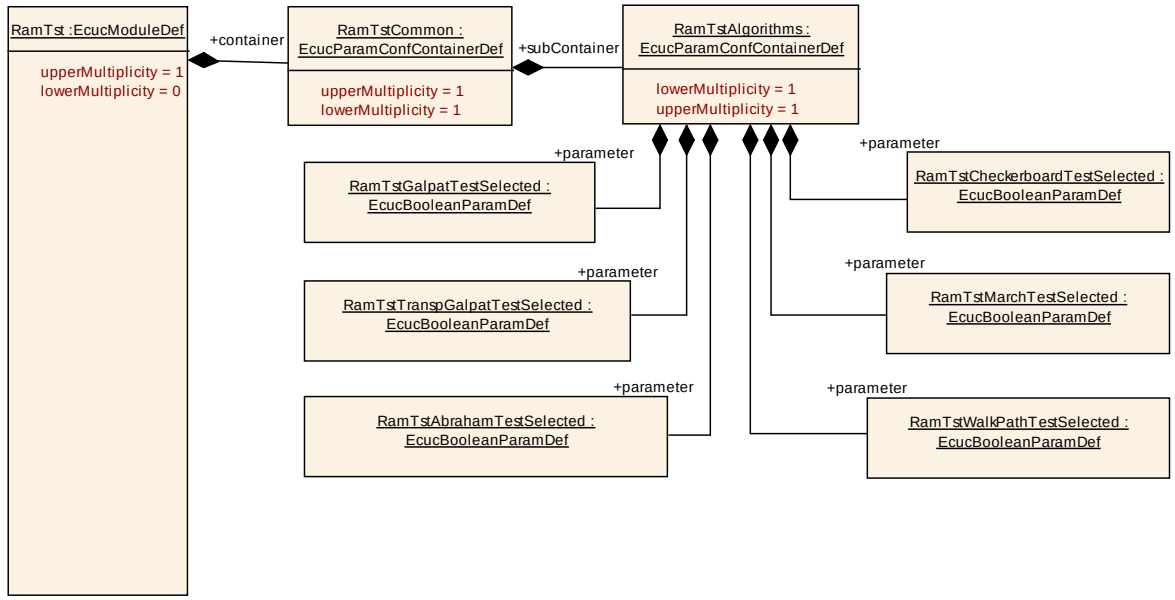
SWS Item	**ECUC_RamTst_00132 :**		
Name	RamTstGalpatTestSelected {RAMTST_GALPAT_TEST_SELECTED}		
Description	Preprocessor switch to select the Galpat Test ON or OFF		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00133 :**		
Name	RamTstMarchTestSelected {RAMTST_MARCH_TEST_SELECTED}		
Description	Preprocessor switch to select the March Test ON or OFF		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00134 :**		
Name	RamTstTranspGalpatTestSelected {RAMTST_TRANSP_GALPAT_TEST_SELECTED}		
Description	Preprocessor switch to select the Transparent Galpat Test ON or OFF		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	ECUC_RamTst_00135 :		
Name	RamTstWalkPathTestSelected {RAMTST_WALK_PATH_TEST_SELECTED}		
Description	Preprocessor switch to select the Walking Path Test ON or OFF		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers



10.2.6 RamTstConfigParams

SWS Item	ECUC_RamTst_00066 :
Container Name	RamTstConfigParams{RamTst_ConfigParams}
Description	This container specifies configuration parameters which are set at pre-compile or link time.
Configuration Parameters	

SWS Item	ECUC_RamTst_00181 :		
Name	RamTstDefaultAlgParamsId {RAMTST_DEFAULT_ALG_PARAMS_ID}		
Description	This is the identifier for the default "RamTstAlgParams" valid after the "RamTst_Init(..)" function. It is the initial value for a RAM variable which could be changed by the function "RamTst_SelectAlgParams".		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00154 :**		
Name	RamTstMinNumberOfTestedCells {RAMTST_MIN_NUMBER_OF_TESTED_CELLS}		
Description	Minimum number of tested cells for one cycle of a background test, as defined by implementer.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

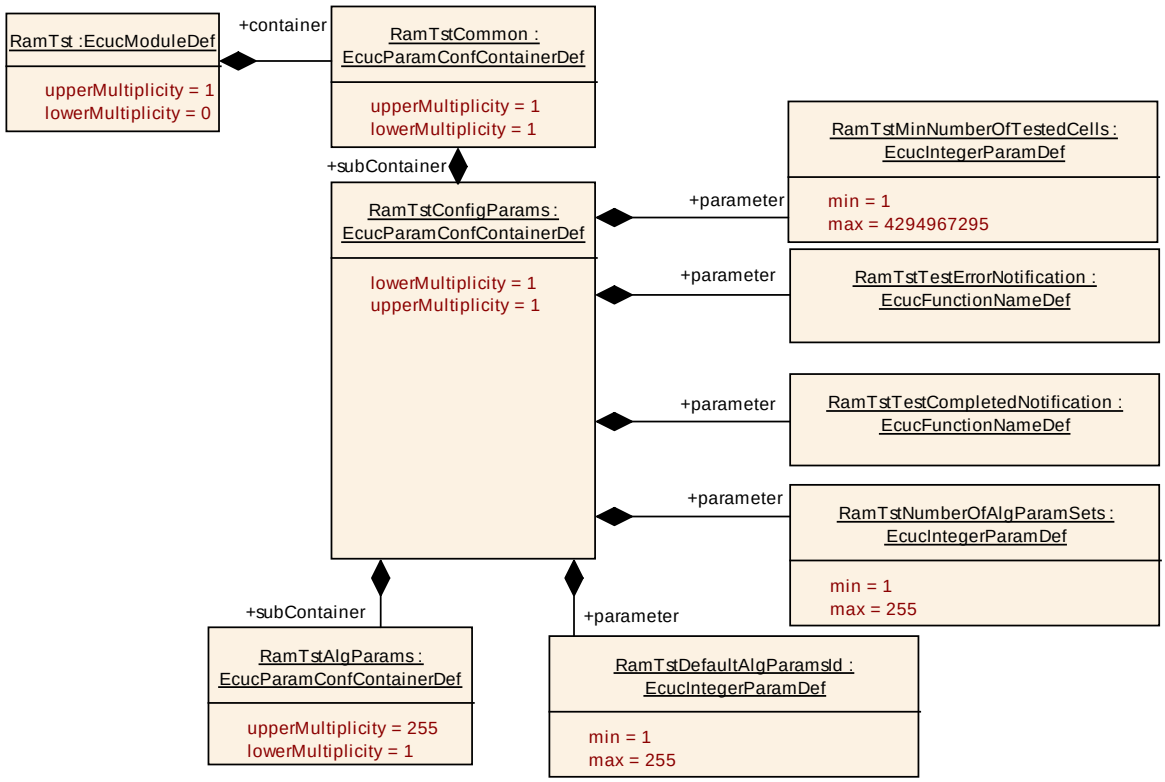
SWS Item	**ECUC_RamTst_00180 :**		
Name	RamTstNumberOfAlgParamSets {RAMTST_NUMBER_OF_ALG_PARAM_SETS}		
Description	Number of configured parameter sets for the available test algorithms. calculationFormula = count of the container RamTst_AlgParams		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local dependency: "RamTstNumberOfAlgParamSets" is derived by the count of "RamTstAlgParams" which is part of the same subContainer and has a multiplicity of 1 to 255.		

SWS Item	**ECUC_RamTst_00138 :**		
Name	RamTstTestCompletedNotification {RAMTST_TEST_COMPLETED_NOTIFICATION}		
Description	This function will be called from a background test after finishing the RAM test without an error.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00139 :**
Name	RamTstTestErrorNotification {RAMTST_TEST_ERROR_NOTIFICATION}
Description	This function will be called from a background test if an error occurs during the RAM test.
Multiplicity	1
Type	EcucFunctionNameDef
Default value	--
maxLength	--
minLength	--
regularExpression	--

ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
RamTstAlgParams	1..255	This container holds parameters for configuring an algorithm. For each algorithm selected in the RamTst_Algorithms container there can be one or more RamTstAlgParams containers. The multiplicity of the included container RamTstBlockParams depends on the number of separate blocks of RAM which are defined for the particular test configuration.



10.2.7 RamTstAlgParams

SWS Item	**ECUC_RamTst_00090 :**
Container Name	RamTstAlgParams{RamTst_AlgParams}
Description	This container holds parameters for configuring an algorithm. For each algorithm selected in the RamTst_Algorithms container there can be one or more RamTstAlgParams containers. The multiplicity of the included container RamTstBlockParams depends on the number of separate blocks of RAM which are defined for the particular test configuration.
Configuration Parameters	

SWS Item	**ECUC_RamTst_00179 :**
Name	RamTstAlgParamsId {RAMTST_ALG_PARAMS_ID}
Description	This is the identifier by which this RamTstAlgParams set can be selected.
Multiplicity	1

Type	EcucIntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00178 :**		
Name	RamTstAlgorithm {RAMTST_ALGORITHM}		
Description	This is the algorithm for which this RamTstAlgParams set is defined. Note that the same algorithm can be used in more than one RamTstAlgParams. Constraint: Only the algorithms selected by RamTstCommon/ RamTstAlgorithms can be used.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	RAMTST_ABRAHAM_TEST	--	
	RAMTST_CHECKERBOARD_TEST	--	
	RAMTST_GALPAT_TEST	--	
	RAMTST_MARCH_TEST	--	
	RAMTST_TRANSP_GALPAT_TEST	--	
	RAMTST_WALK_PATH_TEST	--	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00152 :**		
Name	RamTstExtNumberOfTestedCells {RAMTST_EXT_NUMBER_OF_TESTED_CELLS}		
Description	This is the absolute maximum value for the number of cells that NUMBER_OF_TESTED_CELLS and MAX_NUMBER_OF_TESTED_CELLS can be.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

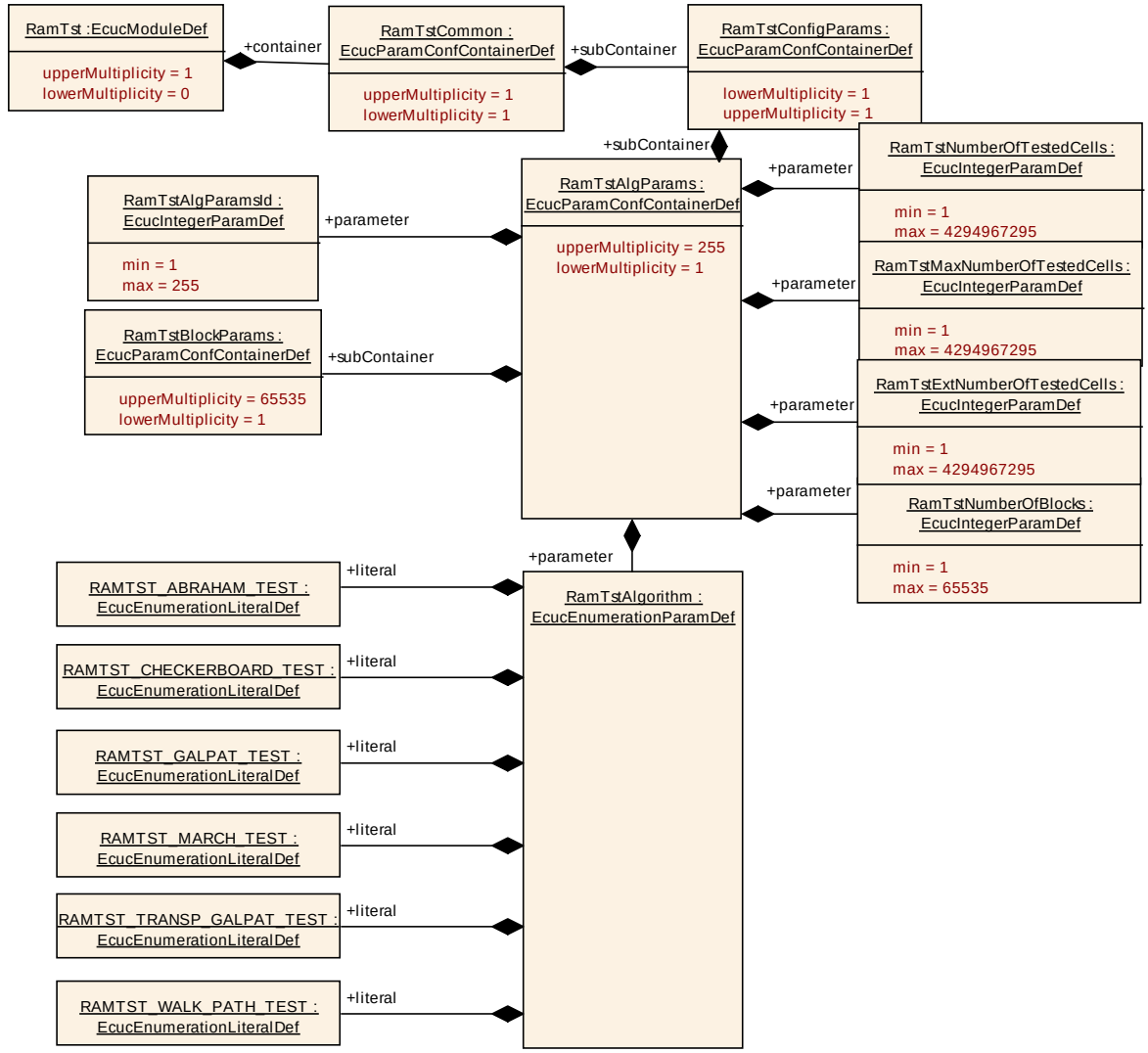
SWS Item	**ECUC_RamTst_00153 :**		
Name	RamTstMaxNumberOfTestedCells {RAMTST_MAX_NUMBER_OF_TESTED_CELLS}		
Description	This is the maximum value for the number of cells that can be tested in one cycle of a background test.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00141 :**

Name	RamTstNumberOfBlocks {RAMTST_NUMBER_OF_BLOCKS}		
Description	Number of RAM blocks configured using the container "RamTst_BlockParams" calculationFormula = Count of RamTstBlockParams contained in this RamTstAlgParams.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local dependency: "RamTstNumberOfBlocks" is derived by the count of the number of "RamTstBlockParams" containers which are part of the same subcontainer and have a multiplicity of 0..65535.		

SWS Item	**ECUC_RamTst_00142 :**		
Name	RamTstNumberOfTestedCells {RAMTST_NUMBER_OF_TESTED_CELLS}		
Description	This is the initial value for a RAM variable, which can be changed by the function "RamTst_ChangeNumberOfTestedCells"		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
RamTstBlockParams	1..65535	This container holds the description for one block of RAM. For each RAM block to be tested by a given algorithm, there is one container which describes the block. Multiple instances of this container are included in each container RamTst_AlgParams.



10.2.8 RamTstBlockParams

SWS Item	**ECUC_RamTst_00091 :**
Container Name	RamTstBlockParams{RamTst_BlockParams}
Description	This container holds the description for one block of RAM. For each RAM block to be tested by a given algorithm, there is one container which describes the block. Multiple instances of this container are included in each container RamTst_AlgParams.
Configuration Parameters	

SWS Item	**ECUC_RamTst_00143 :**		
Name	RamTstBlockId {RAMTST_BLOCK_ID}		
Description	ID of the RAM block		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00144 :**

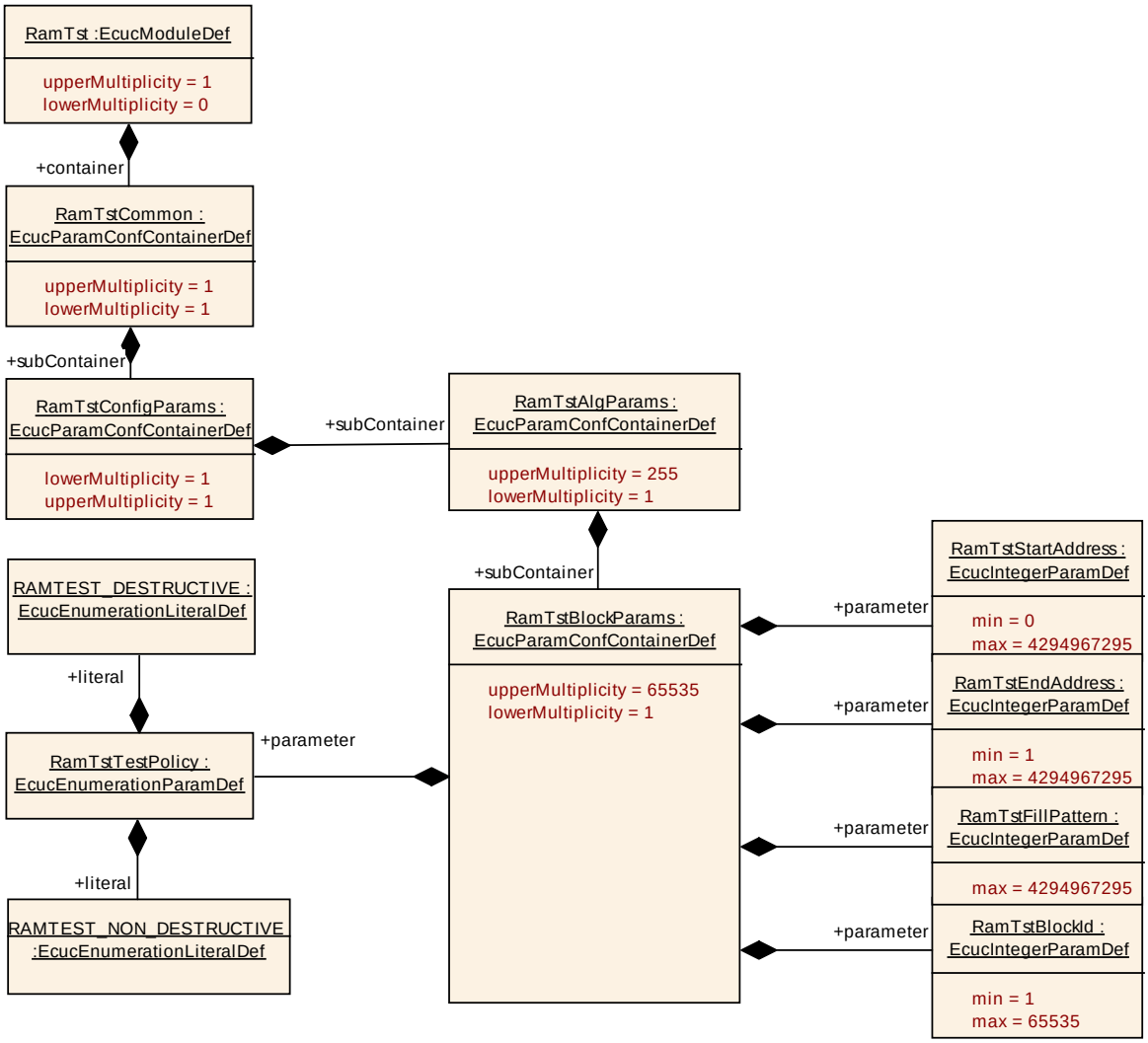
Name	RamTstEndAddress {RAMTST_END_ADDRESS}		
Description	End Address of the RAM block. Constraint: It must be larger than the RamTstStartAddress.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00176 :**		
Name	RamTstFillPattern {RAMTST_FILL_PATTERN}		
Description	Pattern to be filled into each memory cell after destructive test of this block.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00145 :**		
Name	RamTstStartAddress {RAMTST_START_ADDRESS}		
Description	Start Address of the RAM block. Constraint: It must be smaller than the RamTstEndAddress.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_RamTst_00177 :**		
Name	RamTstTestPolicy {RAMTST_TEST_POLICY}		
Description	Policy regarding destruction or non-destruction of memory content.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	RAMTEST_DESTRUCTIVE	RAM test does not restore memory content.	
	RAMTEST_NON_DESTRUCTIVE	RAM test restores memory content.	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers


RamTst :EcucModuleDef
upperMultiplicity = 1
lowerMultiplicity = 0
+container
RamTstCommon :
EcucParamConfContainerDef
upperMultiplicity = 1
lowerMultiplicity = 1
+subContainer
RamTstConfigParams :
EcucParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 1
+subContainer
RamTstAlgParams :
EcucParamConfContainerDef
upperMultiplicity = 255
lowerMultiplicity = 1
+subContainer
RamTstBlockParams :
EcucParamConfContainerDef
upperMultiplicity = 65535
lowerMultiplicity = 1
+parameter
RamTstStartAddress :
EcucIntegerParamDef
min = 0
max = 4294967295
+parameter
RamTstEndAddress :
EcucIntegerParamDef
min = 1
max = 4294967295
+parameter
RamTstFillPattern :
EcucIntegerParamDef
max = 4294967295
+parameter
RamTstBlockId :
EcucIntegerParamDef
min = 1
max = 65535
RAMTEST_DESTRUCTIVE :
EcucEnumerationLiteralDef
+literal
RamTstTestPolicy :
EcucEnumerationParamDef
+parameter
+literal
RAMTEST_NON_DESTRUCTIVE
:EcucEnumerationLiteralDef

10.3 Published Parameters

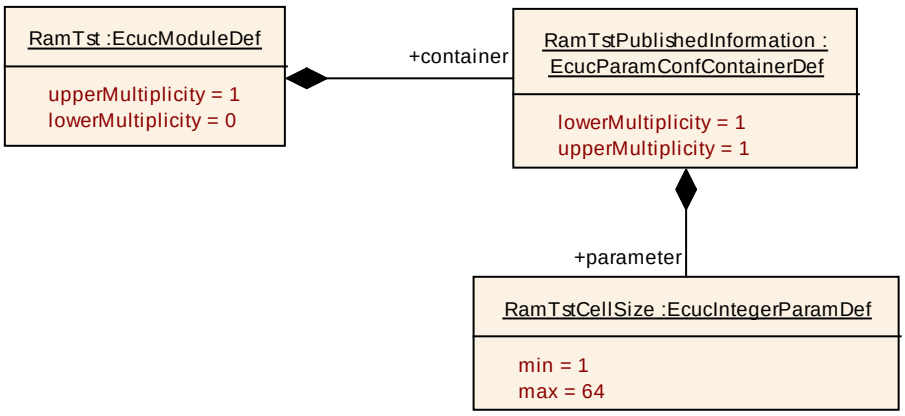
For details refer to the chapter 10.3 "Published Information" in *SWS_BSWGeneral*.

10.3.1 RamTstPublishedInformation

SWS Item	**ECUC_RamTst_00186 :**		
Container Name	RamTstPublishedInformation{RamTst_PublishedInformation}		
Description	Container holding all RamTst specific published information parameter.		
Configuration Parameters			

SWS Item	**ECUC_RamTst_00187 :**		
Name	RamTstCellSize {RAMTST_CELL_SIZE}		
Description	Size of RAM cells (in bits) which can be tested individually by the given implementation.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 64		
Default value	--		
ConfigurationClass	***Published Information***	X	All Variants
Scope / Dependency	scope: local		

No Included Containers



10.4 Implementation Specific Information and Parameters

[SWS_RamTst_00081] ⌈ The implementer shall provide measured or calculated runtime information in the documentation of the module for each algorithm implementation. The information is to be presented as shown in the following table, specifying whether the parameters are measured or calculated. ⌋(SRS_BSW_00402)

Microcontroller	Frequency	RamCellSize [bit]:	No of cells/cycle	Average Runtime	Interrupt lock time	Internal used RAM
--	--	--	--	--	--	--

[SWS_RamTst_00205] ⌈ If an implementation of the RAM Test module supports vendor specific test algorithms or other additional configuration parameters, the implementer shall provide a formal vendor-specific definition of these parameters including their documentation (as part of the BSW Module Description). ⌋()

11 Not applicable requirements

[SWS_RamTst_00999] ⌈ These requirements are not applicable to this specification. ⌋ (SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00336, SRS_BSW_00375, SRS_BSW_00383, SRS_BSW_00386, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00422, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00425, SRS_BSW_00426, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, BSW00434, SRS_BSW_00437, SRS_BSW_00438, SRS_BSW_00005, SRS_BSW_00006, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00164, SRS_BSW_00172, SRS_BSW_00301, SRS_BSW_00302, SRS_BSW_00306, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00312, SRS_BSW_00321, SRS_BSW_00328, SRS_BSW_00333, SRS_BSW_00341, SRS_BSW_00347, SRS_BSW_00353, SRS_BSW_00361, SRS_BSW_00370, SRS_BSW_00374, SRS_BSW_00378, SRS_BSW_00379, SRS_BSW_00413, SRS_BSW_00438, SRS_BSW_00440, SRS_BSW_00441, BSW00443, BSW00444, SRS_BSW_00449, SRS_SPAL_12063, SRS_SPAL_12064, SRS_SPAL_12067, SRS_SPAL_12068, SRS_SPAL_12069, SRS_SPAL_12075, SRS_SPAL_12125, SRS_SPAL_12267, SRS_SPAL_12461, SRS_SPAL_12462, SRS_SPAL_12463, SRS_SPAL_12078, SRS_SPAL_12092, SRS_SPAL_12265)