Document Title	Specification of RAM Test
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	076
Document Classification	Standard

Document Version	1.5.0
Document Status	Final
Part of Release	3.2
Revision	3

Document Change History

Date	Version	Changed by	Change Description
28.02.2014	1.5.0	AUTOSAR Release Management	• Adjustments to ISO 26262 • Resolved inconsistency between FEE013 and BSW004 in chapter 5.1.3 • Editorial changes • Removed chapter(s) on change documentation
17.05.2012	1.4.0	AUTOSAR Administration	• Updated the limitations chapter with respect to the MC-OS. • Removal of a useless statement. • Typos correction
07.04.2011	1.3.0	AUTOSAR Administration	• Alignment of the R3.2 document base to R 4.0.2 • Improvement of error reporting • Typo corrections • 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.
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

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
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

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Advice for users

AUTOSAR 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%) [10]. 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.

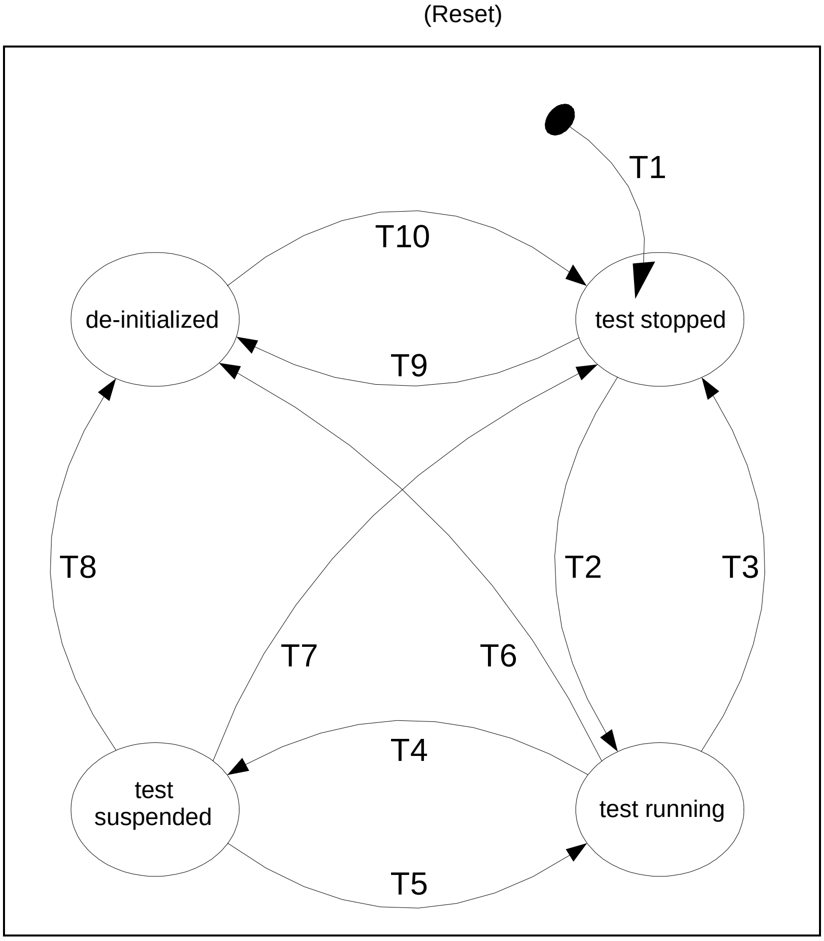
TEST PARAMETER AND FEEDBACK API's
RamTst_GetVersionInfo
RamTst_GetExecutionStatus
RamTst_GetTestResult
RamTst_GetTestResultPerBlock
RamTst_GetAlgParams
RamTst_GetTestAlgorithm
RamTst_GetNumberOfTestedCells
RamTst_SelectAlgParams
RamTst_ChangeNumberOfTestedCells

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.



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:

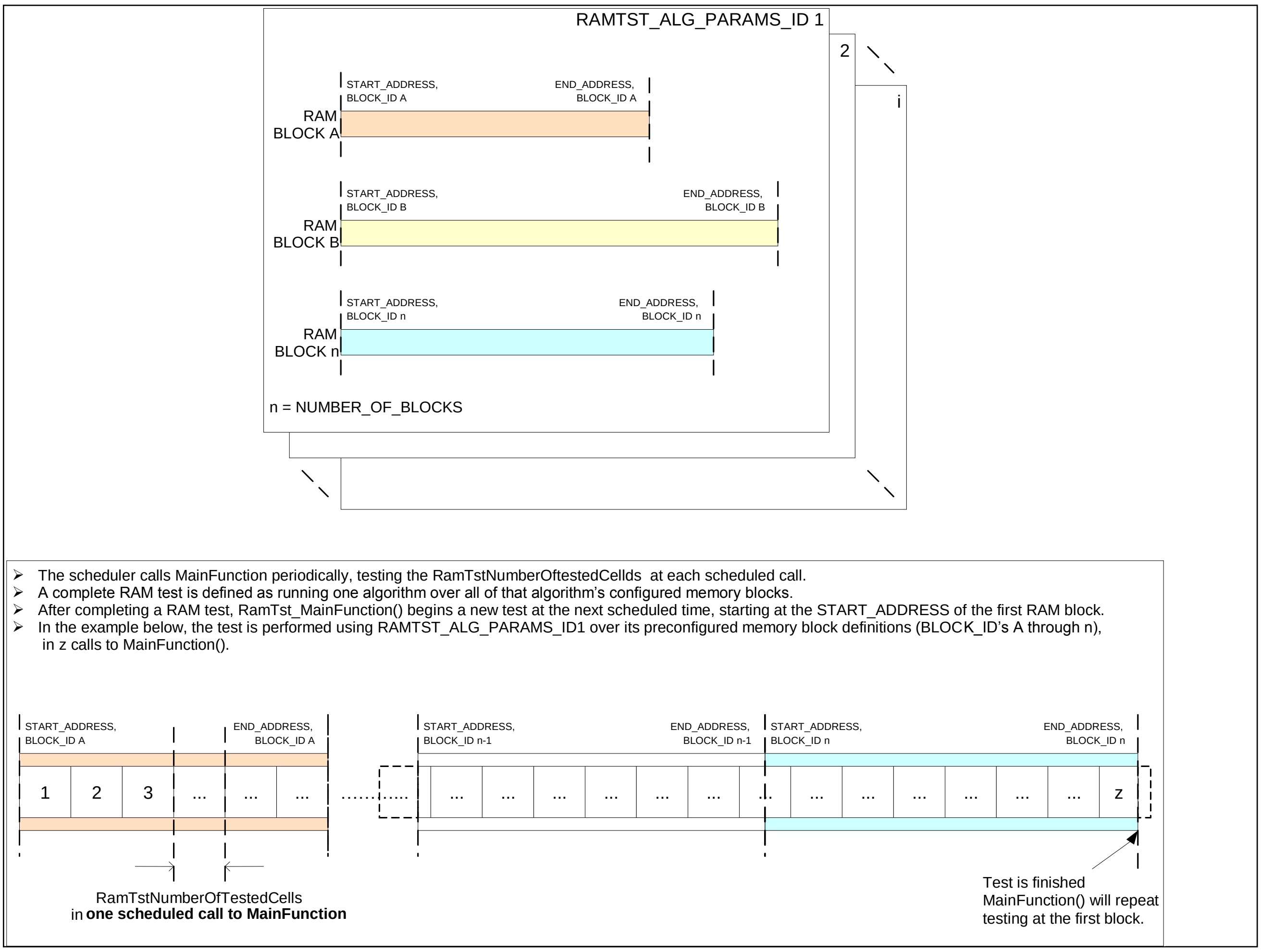
[1] RamTst_Suspend causes a state change to "test suspended" at the end of the current RamTst_MainFunction() atomic sequence if RamTst_MainFunction() is actively testing.

[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.

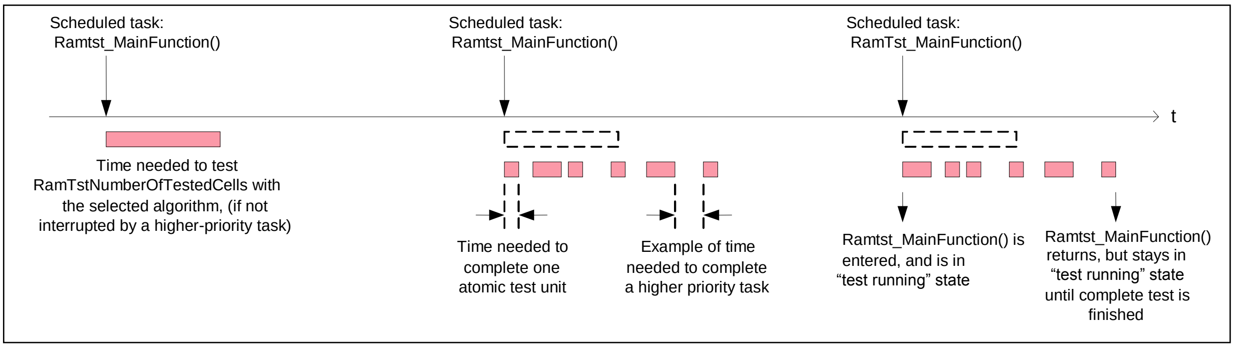
[4] RamTst_ChangeNumberOfTestedCells operates at the end of the current RamTst_MainFunction() atomic sequence if RamTst_MainFunction() is actively testing. For a foreground test, RamTst_ChangeNumberOfTestedCells is not relevant.

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.


RAMTST_ALG_PARAMS_ID 1
2
i
START_ADDRESS, BLOCK_ID A
END_ADDRESS, BLOCK_ID A
RAM BLOCK A
START_ADDRESS, BLOCK_ID B
END_ADDRESS, BLOCK_ID B
RAM BLOCK B
START_ADDRESS, BLOCK_ID n
END_ADDRESS, BLOCK_ID n
RAM BLOCK n
n = NUMBER_OF_BLOCKS
START_ADDRESS, BLOCK_ID A
END_ADDRESS, BLOCK_ID A
START_ADDRESS, BLOCK_ID n-1
END_ADDRESS, BLOCK_ID n-1
START_ADDRESS, BLOCK_ID n
END_ADDRESS, BLOCK_ID n
1
2
3
...
z
RamTstNumberOfTestedCells in **one scheduled call to MainFunction**
Test is finished MainFunction() will repeat testing at the first block.

- The scheduler calls MainFunction periodically, testing the RamTstNumberOftestedCellds at each scheduled call.
- A complete RAM test is defined as running one algorithm over all of that algorithm's configured memory blocks.
- After completing a RAM test, RamTst_MainFunction() begins a new test at the next scheduled time, starting at the START_ADDRESS of the first RAM block.
- In the example below, the test is performed using RAMTST_ALG_PARAMS_ID1 over its preconfigured memory block definitions (BLOCK_ID's A through n), in z calls to MainFunction().

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.


Scheduled task: Ramtst_MainFunction()
Scheduled task: Ramtst_MainFunction()
Scheduled task: RamTst_MainFunction()
t
Time needed to test RamTstNumberOfTestedCells with the selected algorithm, (if not interrupted by a higher-priority task)
Time needed to complete one atomic test unit
Example of time needed to complete a higher priority task
Ramtst_MainFunction() is entered, and is in "test running" state
Ramtst_MainFunction() returns, but stays in "test running" state until complete test is finished

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_Glossary.doc

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

3.2 Related Standards and Norms

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

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

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

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

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

RamTst002: 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).

RamTst082: 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 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 and RamTst203 for additional information.

4.1.1 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 sleep mode as all cores are stopped. This allows the EcuM to delay the sleep state of one of the cores to perform a RAM test 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.

4.1.2 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. MC systems' lack of atomicity causes problems for shared memory.

4.2 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 (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

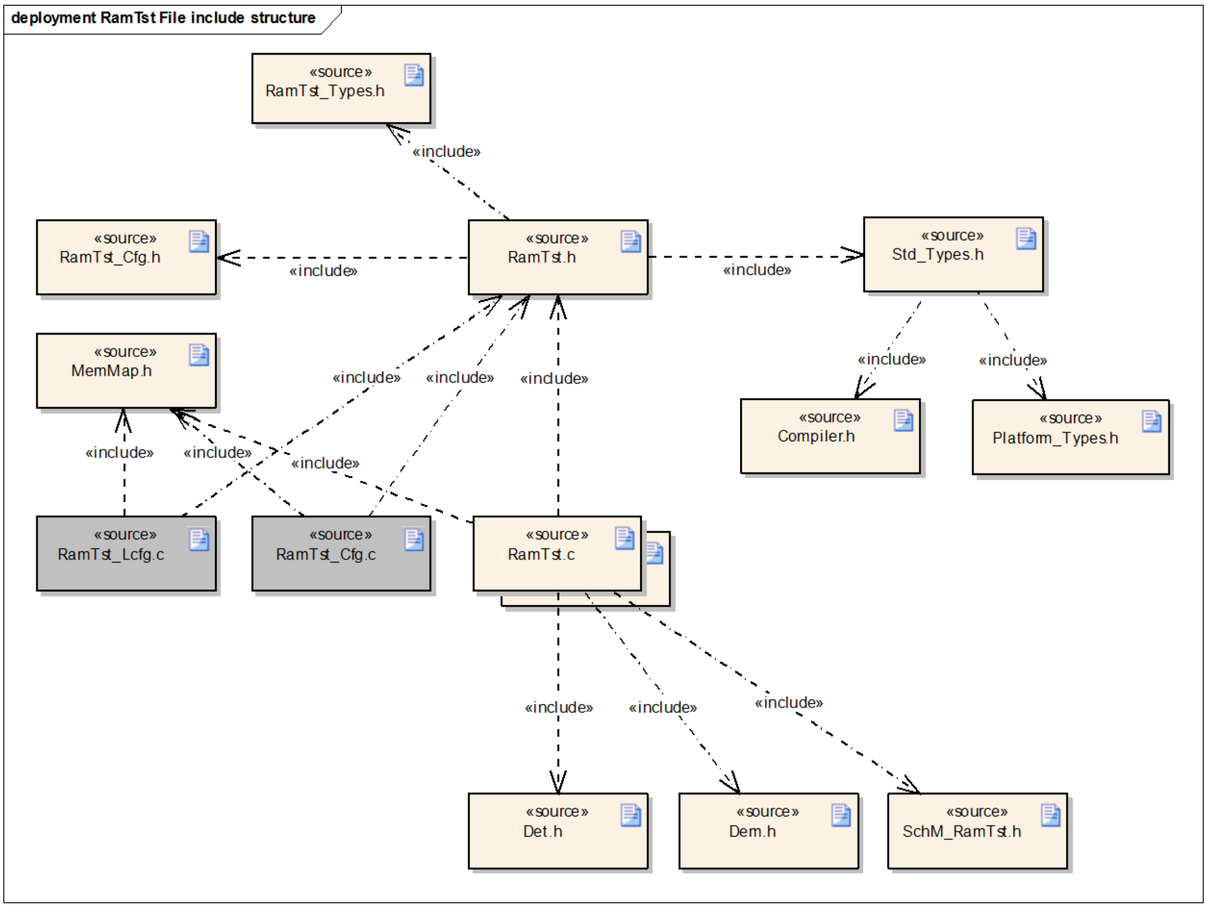
5.1.1 Code File Structure

RamTst086: The code file structure for the RAM Test module shall not be defined within this specification completely. At this point, it shall be pointed out that the code-file structure shall include the following files named:

- `RamTst_Cfg.c` for compile time configurable parameters, which need source code (e.g. tables)
- `RamTst_Lcfg.c` for link time configurable parameters

5.1.2 Header File Structure

RamTst003: The include structure for the source code of the RAM Test module shall be as follows:



RamTst072: The module shall include the `Dem.h` file.

Hint: By this inclusion, the APIs to report errors as well as the required Event Id symbols are included. This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in `Dem_IntErrId.h`.

RamTst087: Pre-compile configuration macros as well as references to c-configuration parameters (compile time or link time) shall be placed into the `RamTst_Cfg.h` file.

RamTst208: Module specific types shall be declared in file `RamTst_Types.h.`

5.1.3 Version Check

RamTst080: The RAM Test Module shall avoid the integration of incompatible files by the following pre-processor checks:

For included (external) header files:

- <MODULENAME>_AR_ MAJOR_VERSION
- <MODULENAME>_AR_ MINOR_VERSION

shall be verified.

If the values are not identical to the values expected by the RAM Test Module, an error shall be reported.

6 Requirements Traceability

Document: General Requirements on Basic Software Modules, [3]

Requirement	***Satisfied by***
Functional Requirements	
[BSW101] Initialization interface	RamTst007, **RamTst099**
[BSW004] Version check	RamTst080
[BSW159] Tool-based configuration	Both static and runtime configuration parameters are located outside the source code of the module. This is the prerequisite for automatic configuration.
[BSW167] Static configuration checking	Constraints are partially stated in the configuration parameter description.
[BSW168] Diagnostic interface of SW components	Not applicable (not a SW-Component)
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable (not a SW-Component)
[BSW171] Configurability of optional functionality	RamTst065
[BSW00323] API parameter checking	RamTst033, RamTst037, RamTst039, RamTst040, RamTst068, RamTst095, **RamTst115**, **RamTst097**, RamTst170, RamTst172, RamTst210, RamTst214
[BSW00336] Shutdown interface	Not applicable (this module does not need such a function)
[BSW00337] Classification of errors	RamTst067
[BSW00338] Detection and reporting of development errors	RamTst067, RamTst068, RamTst069, RamTst070, **RamTst084**, **RamTst097**, **RamTst116**
[BSW00339] Reporting of production relevant error status	RamTst073, RamTst067, RamTst076, RamTst071, **RamTst111**, RamTst011, RamTst213, RamTst216
[BSW00344] Reference to link-time configuration	RamTst026, RamTst027, RamTst066, RamTst090, RamTst091
[BSW00345] Pre-compile-time configuration	RamTst065, RamTts058, RamTst066, RamTst068, RamTst070, RamTst079, RamTst090, RamTst091
[BSW00369] Do not return development error codes via API	Satisfied by all API's
[BSW00375] Notification of wake-up reason	Not applicable (wakeups are not supported by this module)
[BSW00380] Separate C-files for configuration parameters	RamTst086
[BSW00381] Separate configuration header file for pre-compile time parameters	RamTst087
[BSW00383] List dependencies of configuration files	Not applicable (there are no dependencies to other configuration files)
[BSW00384] List dependencies to other modules	RamTst072
[BSW00385] List possible error notifications	RamTst067
[BSW00386] Configuration for detecting an error	Not applicable (no configuration for error detection)
[BSW00387] Specify the configuration class of callback function	RamTst138_Conf, RamTst139_Conf
[BSW00388] Introduce containers	RamTst065, RamTst066, RamTst070, RamTst090, RamTst091
[BSW00389] Containers shall have names	RamTst065, RamTst066, RamTst070,

	RamTst090, RamTst091
[BSW00390] Parameter content shall be unique within the module	RamTst065, , RamTst070, RamTst090, RamTst091
[BSW00391] Parameter shall have unique names	RamTst065, RamTst066, RamTst070, RamTst090, RamTst091 Prefix “RamTst” added to each parameter
[BSW00392] Parameters shall have a type	RamTst065, RamTst066, RamTst070, RamTst090, RamTst091
[BSW00393] Parameters shall have a range	RamTst065, RamTst066, RamTst070, RamTst090, RamTst091
[BSW00394] Specify the scope of the parameters	RamTst065, RamTst066, RamTst070, RamTst090, RamTst091 “Local” marked as Module (template and SRS General are inconsistent)
[BSW00395] List the required parameters (per parameter)	All parameter in section 10.2 Containers and configuration parameters are required.
[BSW00396] Configuration classes	See section 10.2.1 Variants
[BSW00397] Pre-compile-time parameters	RamTst065, RamTst066, RamTst070, RamTst090, RamTst091
[BSW00398] Link-time parameters	RamTst066, RamTst090, RamTst091
[BSW00399] Loadable post-build time parameters	Not applicable (post build time configuration is not supported)
[BSW00400] Selectable post-build time parameters	Not applicable (post build time configuration is not supported)
[BSW00402] Published information	**RamTst080**, **RamTst081**, RamTst186_Conf
[BSW00404] Reference to post build time configuration	Not applicable (post build time configuration is not supported)
[BSW00405] Reference to multiple configuration sets	Not applicable (post build time is not supported)
[BSW00406] Check module initialization	RamTst006. **RamTst012**, **RamTst013**
[BSW00407] Function to read out published parameters	RamTst078, RamTst079, **RamTst109**
[BSW00409] Header files for production code error IDs	RamTst073
[BSW00412] Separate H-file for configuration parameters	RamTst087
[BSW00416] Sequence of Initialization	Not applicable (this is a general software integration requirement)
[BSW00417] Reporting of error events by non-basic software	Not applicable (this is a basic software module)
[BSW00419] Separate C-files for pre-compile time configuration parameters	RamTst086
[BSW00422] Debouncing of production relevant error status	Not applicable (it makes no sense to debounce a Ram error)
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR interfaces	Not applicable (this module has no connection to the RTE)
[BSW00424] BSW main processing function task allocation	Not applicable (the scheduling of a BSW is not part of this SWS)
[BSW00425] Trigger conditions for schedulable objects	Not applicable (requirement for the implementer)
[BSW00426] Exclusive areas in BSW modules	Not applicable (requirement for the implementer), but mentioned as a hint to RamTst_MainFunction
[BSW00427] ISR description for BSW modules	RamTst221
[BSW00428] Execution order dependencies of main processing functions	Not applicable (requirement for the implementer and integrator)
[BSW00429] Restricted BSW OS functionality access	Not applicable (this module does not use OS services)
[BSW00432] Modules should have separate main	Not applicable

processing functions for read/receive and write/transmit data path	(this module does not have send/receive functionality)
[BSW00433] Calling of main processing functions	Requirement for the test environment. Made possible by separate APIs for foreground test.
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (this is a special requirement for the BSW scheduler)
[BSW00437] No-init area in RAM	Not applicable (this is a requirement on the memory manager)
[BSW00438] Post-build configuration data structure	Not applicable (post build time configuration is not supported)
Non-functional Requirements	
[BSW003] Version identification	**RamTst080, RamTst117**
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (this is a requirement on architecture)
[BSW006] Platform independency	Not applicable to MCAL
[BSW007] HIS MISRA C	Common AUTOSAR non-functional requirement for the implementer.
[BSW009] Module User Documentation	Not applicable (requirement for the implementer)
[BSW010] Memory resource documentation	Not applicable (requirement for the implementer)
[BSW158] Separation of configuration from implementation	RamTst086
[BSW160] Human-readable configuration data	Common AUTOSAR non-functional requirement; fulfilled by using XML
[BSW161] Microcontroller abstraction	Not applicable (this is a requirement on architecture)
[BSW162] ECU layout abstraction	Not applicable (this is a requirement on architecture)
[BSW164] Implementation of interrupt service routines	Not applicable (this is a requirement on architecture)
[BSW172] Compatibility and documentation of scheduling strategy	Not applicable (requirement for the implementer)
[BSW00300] Module naming convention	Common AUTOSAR non-functional requirement for the implementer.
[BSW00301] Limit imported information	Not applicable (requirement for the implementer)
[BSW00302] Limit exported information	Not applicable (requirement for the implementer)
[BSW00304] AUTOSAR integer data types	See section 8.1 Imported types
[BSW00305] Self-defined data types naming convention	See section 8.2 Type definitions
[BSW00306] Avoid direct use of compiler and platform specific keywords	Not applicable (requirement for the implementer)
[BSW00307] Global variables naming convention	Common AUTOSAR non functional requirement for the implementer.
[BSW00308] Definition of global data	Not applicable (requirement for the implementer)
[BSW00309] Global data with read-only constraint	Not applicable (requirement for the implementer)
[BSW00310] API naming convention	See chapter 8 API specification
[BSW00312] Shared code shall be reentrant	Not applicable (requirement for the implementer)
[BSW00314] Separation of interrupt frames and service routines	RamTst221
[BSW00318] Format of module version numbers	**RamTst080**
[BSW00321] Enumeration of module version	Not applicable

numbers	(requirement for the implementer)
[BSW00325] Runtime of interrupt service routines	RamTst221
[BSW00326] Transition from ISRs to OS tasks	RamTst221
[BSW00327] Error values naming convention	See section 7.2 Error classification
[BSW00328] Avoid duplication of code	Not applicable (requirement for the implementer)
[BSW00329] Avoidance of generic interfaces	See chapter 8 API specification
[BSW00330] Usage of macros / inline functions instead of functions	**RamTst117**
[BSW00331] Separation of error and status values	**RamTst102**, **RamTst103** 8.2.1 RamTst_ExecutionStatusType and 8.2.2 RamTst_TestResultType
[BSW00333] Documentation of callback function context	Not applicable (requirement for the implementer)
[BSW00334] Provision of XML file	Not applicable (requirement for the implementer)
[BSW00335] Status values naming convention	See section 8.2 Type definitions
[BSW00341] Microcontroller compatibility documentation	Not applicable (requirement for the implementer)
[BSW00342] Usage of source code and object code	Common AUTOSAR non-functional requirement for the implementer.
[BSW00343] Specification and configuration of time	Common AUTOSAR non-functional requirement for the implementer.
[BSW00346] Basic set of module files	RamTst086, RamTst003
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable (only one instance of this module)
[BSW00348] Standard type header	See chapter 8.1Imported types
[BSW00350] Development error detection keyword	RamTst068, RamTst070
[BSW00353] Platform specific type header	Not applicable (requirement for the implementer)
[BSW00355] Do not redefine AUTOSAR integer data types	See section 8.2 Type definitions
[BSW00357] Standard API return type	**RamTst074** See section 8.1 Imported types
[BSW00358] Return type of init() functions	**RamTst099** See section 8.3.1 RamTst_Init
[BSW00359] Return type of callback functions	See section 8.6.3 Configurable interfaces
[BSW00360] Parameters of callback functions	See section 8.6.3 Configurable interfaces
[BSW00361] Compiler specific language extension header	Not applicable (requirement for the implementer)
[BSW00370] Separation of callback interface from API	Not applicable, because callbacks are configured as function pointers
[BSW00371] Do not pass function pointers via API	Satisfied by the APIs
[BSW00373] Main processing function naming convention	**RamTst110** See section 8.5.1 RamTst_MainFunction
[BSW00374] Module vendor identification	Not applicable (requirement for the implementer)
[BSW00376] Return type and parameters of main processing functions	**RamTst110** See section 8.5.1 RamTst_MainFunction
[BSW00377] Module specific API return types	See sections 8.1 Imported types and 8.2 Type definitions
[BSW00378] AUTOSAR boolean type	Not applicable (requirement for the implementer)
[BSW00379] Module identification	Not applicable (requirement for the implementer)

[BSW00401] Documentation of multiple instances of configuration parameters	See section 10.2 Containers and configuration parameters
[BSW00408] Configuration parameter naming convention	See section 10.2 Containers and configuration parameters
[BSW00410] Compiler switches shall have defined values	See section 10.2 Containers and configuration parameters
[BSW00411] Get version info keyword	RamTst070, RamTst079
[BSW00413] Accessing instances of BSW modules	Not applicable (instances makes no sense for this module)
[BSW00414] Parameter of init function	RamTst093 See section 8.3.1 RamTst_Init
[BSW00415] User dependent include files	RamTst003
[BSW00435] Module header file structure for the basic software scheduler	RamTst003
[BSW00436] Module header file structure for the basic software memory mapping	RamTst003
[BSW00438] Post Build Configuration Data Structure	Not applicable (no post-build configuration)
[BSW00439] Declaration of interrupt handlers and ISRs	RamTst221
[BSW00440] Function prototype for callback functions of AUTOSAR Services	Not applicable
[BSW00441] Enumeration literals and #define naming convention	Not applicable (requirement for the implementer)
[BSW00442] Debugging Support in Modules	RamTst153,RamTst155,RamTst157,RamTst158, RamTst159,RamTst161,RamTst162,RamTst163, RamTst164,RamTst165
[BSW00443] Enabling / disabling defensive behavior of BSW	Not applicable
BSW00444] Error reporting and logging for defensive	Not applicable
[BSW00445] Protection against untimely call of BSW initialization	Valid, but not explicitly mentioned
[BSW00446] Protection against untimely call of BSW de-initialization	Valid, but not explicitly mentioned
[BSW00447] Standardizing Include file structure of BSW	RamTst003
[BSW00448] Module SWS shall not contain requirements from Other Modules	Satisfied in general
[BSW00449] BSW Service APIs used by Autosar Application Software shall return a Std_ReturnType	Not applicable
[BSW00450] Main Function Processing for Un-Initialized Modules	RamTst175

Document: General Requirements on SPAL, [4]

Requirement	***Satisfied by***
General Requirements	
[BSW157] Notification mechanisms of drivers and handlers	RamTst042RamTst043, RamTst044, RamTst045, RamTst046, RamTst066
[BSW12056] Configuration of notification mechanisms	RamTst043, RamTst044, RamTst045, RamTst046, RamTst066
[BSW12057] Driver module initialization	RamTst007, RamTst026, RamTst027, RamTst066
[BSW12063] Raw value mode	Not applicable (this module does not provide physical signals)

[BSW12064] Change of operation mode during running operation	Not applicable (only one operation mode available)
[BSW12067] Setting of wake-up conditions	Not applicable (this module has no wakeup sources)
[BSW12068] MCAL initialization sequence	Not applicable (this is a requirement for the implementer)
[BSW12069] Wake-up notification of ECU State Manager	Not applicable (this module has no wakeup sources)
[BSW12075] Use of application buffers	Not applicable (no use of a buffering mechanism)
[BSW12125] Initialization of hardware resources	Not applicable (this module accesses only RAM, so no hardware resource initialization is necessary)
[BSW12129] Resetting of interrupt flags	RamTst221
[BSW12163] Driver module deinitialization	RamTst146
[BSW12169] Control of operation mode	If the test execution status is called an operation mode, then this is satisfied by the APIs to stop/allow/suspend/resume test execution
[BSW12263] Object code compatible configuration concept	RamTst026, RamTst027, RamTst066
[BSW12267] Configuration of wake-up sources	Not applicable (this module has no wakeup sources)
[BSW12448] Behavior after development error detection	RamTst033, RamTst037, RamTst039, RamTst040, **RamTst084**, RamTst095
[BSW12461] Responsibility for register initialization	Not applicable (this module uses no registers)
[BSW12462] Provide settings for register initialization	Not applicable (this module uses no registers)
[BSW12463] Combine and forward settings for register initialization	Not applicable (this module uses no registers)
Non-Functional Requirements	
[BSW12077] Non-blocking implementation	Fulfilled by background test (main function). The foreground test APIs are an exception from this requirement.
[BSW12078] Runtime and memory efficiency	Not applicable (requirement for the implementer)
[BSW12092] Access to drivers	Not applicable
[BSW12264] Specification of configuration items	See chapter 10.2 Containers and configuration parameters
[BSW12265] Configuration data shall be kept constant	Not applicable (requirement for the implementer)

Document: Requirements on RAM Test, [5]

Requirement	***Satisfied by***
[BSW13800] Number of tested cells shall be changeable at runtime	RamTst036, **RamTst107**
[BSW13801] Test cell size shall be a published parameter	RamTst187_Conf
[BSW13802] Multiple RAM areas shall be configurable at post build/ link time	RamTst026, RamTst091_Conf
[BSW13803] A subset of available RAM Test algorithms shall be selectable at pre-compile time	RamTst026, RamTst027, RamTst063, RamTst065, RamTst083, RamTst084, RamTst085, **RamTst097**, **RamTst105**
[BSW13804] A subset of the pre-compile time selected RAM Test algorithms shall be selectable	**RamTst083**, **RamTst105**

at runtime	
[BSW13809] RAM Test execution management	RamTst008, **RamTst026**, RamTst059, RamTst070, RamTst090, RamTst091, **RamTst107**, **RamTst108**
[BSW13810] Current status of RAM Test execution per block shall be available through a get status interface	**RamTst010**, **RamTst011**, RamTst019, RamTst024, RamTst038, **RamTst104**
[BSW13811] Non-destructive RAM Test	RamTst060, RamTst061, RamTst200
[BSW13812] Destructive RAM Test	RamTst061, RamTst201
[BSW13816] Effects of instruction / data queue shall be taken into account	RamTst062
[BSW13820] RAM Test execution status shall be provided by a notification mechanism	RamTst043, RamTst044, RamTst045, RamTst046, **RamTst113**, **RamTst114**
[BSW13822]Test algorithm with low coverage shall be available	RamTst224 See sections 8.2.2, 10.2.5 and 10.2.7
[BSW13823] Test algorithm with medium coverage shall be available	RamTst225 See sections 8.2.2, 10.2.5 and 10.2.7
[BSW13824] Test algorithm with high coverage shall be available	RamTst226 See sections 8.2.2, 10.2.5 and 10.2.7

Implementation requirements originating within this SWS document.

Requirement
RamTst002
RamTst005
RamTst200
RamTst018
RamTst047
RamTst082
RamTst096 (should be a general requirement?)
RamTst100 (should be an SRS requirement)
RamTst101 (should be an SRS requirement)
RamTst195 (should be an SRS requirement)
RamTst197 (should be an SRS requirement)
RamTst204 (should be an SRS requirement)
RamTst221

7 Functional Specification

7.1 Requirements

RamTst005: The RAM Test module shall provide the background RAM test as an asynchronous service.

RamTst206: The RAM Test module shall provide the foreground RAM test as an synchronous service.

RamTst063: The configuration process for the RAM Test module shall allow the selection of a subset of different RAM Test algorithms during pre-compile time.

This subset is to be chosen from the different RAM Test algorithms as specified in RamTst224, RamTst225, RamTst226,RamTst204

RamTst060: 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.

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.

RamTst061: 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.

RamTst062: 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.

[RamTst224]: The RAM Test module shall provide a test algorithm with low coverage as stated in [10], Table D.1.

[RamTst225]: The RAM Test module shall provide a test algorithm with medium coverage as stated in [10], Table D.1.

[RamTst226]: The RAM Test module shall provide a test algorithm with high coverage as stated in [10], Table D.1.

RamTst204: 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 RamTst205).

RamTst221: 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 BSW00427, BSW12129, BSW00325, BSW00326, BSW00439, BSW00314).

7.2 Error Classification

RamTst073: Values for production code Event Ids are assigned externally by the configuration of the DEM. They are published in the file Dem_IntErrId.h and included via Dem.h.

RamTst074: Development error values are of type uint8.

RamTst067: The following errors and exceptions shall be detectable by the RAM Test depending on its build version (development/production mode):

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	**RamTst033**, **RamTst037**, **RamTst095**, **RamTst097**, RamTst170, RamTst172, RamTst210, RamTst214,
API parameter out of specified range	Development	RAMTST_E_OUT_OF_RANGE	0x02	**RamTst039**, **RamTst040**, **RamTst084**, Ramtst223
API service used without module initialization	Development	RAMTST_E_UNINIT	0x03	**RamTst089**
API service called with a NULL pointer	Development	RAMTST_E_PARAM_POINTER	0x04	Ramtst222
RAM failure	Production	RAMTST_E_RAM_FAILURE	Assigned by DEM	**RamTst011**, RamTst213, RamTst216

7.3 Error Detection

RamTst068: The detection of development errors is configurable (*ON* / *OFF*) at pre-compile time. The switch RamTstDevErrorDetect (see RamTst121_Conf) shall activate or deactivate the detection of all development errors.

RamTst115: If the `RamTstDevErrorDetect` switch is enabled, API parameter checking is enabled. The detailed description of the detected errors can be found in section 7.2 and chapter 8.

RamTst076: The detection of production code errors cannot be switched off.

RamTst089: 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

RamTst069: 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 RamTst067.

RamTst071: Production errors shall be reported to Diagnostic Event Manager (DEM) via the `Dem_ReportErrorStatus` API.

RamTst116: Detected development errors shall be reported to the `Det_ReportError` service of the Development Error Tracer (DET) if the pre-processor switch `RamTstDevErrorDetect` is set (see RamTst121_Conf).

7.5 Debugging

RamTst153: Each variable that shall be accessible by AUTOSAR Debugging shall be defined as global variable.

RamTst155: All type definitions of variables which shall be debugged shall be accessible by the header file RamTst.h.

RamTst157: The declaration of variables in the header file shall be such, that it is possible to calculate the size of the variables by C-"`sizeof`".

RamTst158: Variables available for debugging shall be described in the respective Basic Software Module Description.

RamTst159: The internal variable holding the current test execution status shall be available for debugging.

RamTst161: The internal variable holding the current overall test result shall be available for debugging.

RamTst162: The internal variables holding the current test results per block shall be available for debugging.

RamTst163: The internal variable holding the ID for the currently selected test algorithm parameter set shall be available for debugging.

RamTst164: The internal variable holding the current number of cells to be tested per cycle shall be available for debugging.

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.

RamTst200: 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).

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.

RamTst201: 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).

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.

RamTst202: 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 deinit).

- 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.

RamTst207: 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:

RamTst203: 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.

RamTst098:

Module	*Imported Type*
Dem	Dem_EventIdType
Std_Types	Std_VersionInfoType

8.2 Type Definitions

8.2.1 RamTst_ExecutionStatusType

RamTst189:

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().	

RamTst006: 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.

8.2.2 RamTst_TestResultType

RamTst190:

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().	

RamTst012: 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.

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

8.2.3 RamTst_AlgParamsIdType

RamTst191:

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

RamTst188: For the type `RamTst_AlgParamsIdType`, the value 0 shall indicate, that no test parameters (and thus no test algorithm) are selected. This shall be the default value of the corresponding variable after reset.

8.2.4 RamTst_AlgorithmType

RamTst192:

Name:	RamTst_AlgorithmType	
Type:	Enumeration	
Range:	RAMTST_ALGORITHM_UNDEFINED	Undefined algorithm (uninitialized value)
	RAMTST_ALGORITHM_COVERAGE_LOW	Test algorithm with low coverage
	RAMTST_ALGORITHM_COVERAGE_MEDIUM	Test algorithm with medium coverage
	RAMTST_ALGORITHM_COVERAGE_HIGH	Test algorithm with high coverage
Description:	This is a value returned by the API service RamTst_GetTestAlgorithm().	

RamTst013: 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.

RamTst058: The type `RamTst_AlgorithmType` shall contain only the enumerations of the algorithms selected at pre-compile time.

Note that if vendor specific algorithms were defined (see RamTst205), the enumeration fields of `RamTst_AlgorithmType` should be extended accordingly by the implementer (or by a code generator).

8.2.5 RamTst_NumberOfTestedCellsType

RamTst173:

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

8.2.6 RamTst_NumberOfBlocksType

RamTst174:

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

RamTst099:

Service name:	RamTst_Init
Syntax:	void RamTst_Init( )
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.

Note: See also **RamTst093**: in 10.2.1 Variants.

RamTst007: 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.

RamTst096: 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

RamTst146:

Service name:	RamTst_DeInit

Syntax:	`void RamTst_DeInit(`
	
	`)`

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

RamTst147: 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

RamTst100:

Service name:	RamTst_Stop

Syntax:	`void RamTst_Stop(`
	
	`)`

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

RamTst014: When the `RamTst_Stop` function is called, `RamTst_MainFunction` shall still finish the current atomic sequence (if it is executing), afterward the status shall be set to `RAMTST_EXECUTION_STOPPED`. The test result is retained, but test parameters and loop data are not.

RamTst148: After a `RamTst_Stop` call, `RamTst_MainFunction` shall not begin testing again when called by the scheduler until after a `RamTst_Allow` call.

RamTst033: 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.

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

8.3.4 RamTst_Allow

RamTst149:

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

RamTst169: 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.

RamTst170: 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.

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

8.3.5 RamTst_Suspend

RamTst150:

Service name:	RamTst_Suspend
Syntax:	void RamTst_Suspend( )
Service ID[hex]:	0x0d
Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (in-out):	None
Parameters (out):	None
Return value:	None

Description:	Service for suspending current operation of background RAM Test, until RESUME is called.

RamTst171: 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`.

RamTst172: 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.

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

8.3.6 RamTst_Resume

RamTst101:

Service name:	RamTst_Resume

Syntax:	`void RamTst_Resume(`
	
	`)`

Service ID[hex]:	0x0e
Sync/Async:	Asynchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (in-out):	None
Parameters (out):	None
Return value:	None
Description:	Service for allowing to continue the background RAM Test at the point is was suspended.

RamTst018: 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`.

RamTst037: 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.

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

8.3.7 RamTst_GetExecutionStatus

RamTst102:

Service name:	RamTst_GetExecutionStatus	

Syntax:	`RamTst_ExecutionStatusType RamTst_GetExecutionStatus(`
	
	`)`	

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

RamTst019: The function `RamTst_GetExecutionStatus` shall return the current RAM Test execution status.

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

8.3.8 RamTst_GetTestResult

RamTst103:

Service name:	RamTst_GetTestResult	

Syntax:	`RamTst_TestResultType RamTst_GetTestResult(`
	
	`)`	

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

RamTst024: The function `RamTst_GetTestResult` shall return the current RAM test result.

The test result is determined according to RamTst202.

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

8.3.9 RamTst_GetTestResultPerBlock

RamTst104:

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 (in-out):	None	
Parameters (out):	None	
Return value:	RamTst_TestResultType	See type definition
Description:	Service returns the current RAM Test result for the specified block.	

RamTst038: The function `RamTst_GetTestResultPerBlock` shall return the current RAM test result for the specified block.

The test result per block is determined according to RamTst207.

RamTst039: 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`.

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 RamTst143_Conf.
The `RamTst_GetTestResultPerBlock` API can be enabled or disabled by the configuration parameter `RamTstGetTestResultPerBlockApi` within the container `RamTstCommon.`

8.3.10 RamTst_GetVersionInfo

RamTst109:

Service name:	RamTst_GetVersionInfo	

Syntax:	`void RamTst_GetVersionInfo(`
	`    Std_VersionInfoType* versioninfo`
	`)`	

Service ID[hex]:	0x0a	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (in-out):	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.

RamTst078: The function `RamTst_GetVersionInfo` shall return the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407).

RamTst079: The function `RamTst_GetVersionInfo` shall be configurable at pre-compile time (On/Off) by the configuration parameter `RamTstVersionInfoApi` in the container `RamTstCommon`.

RamTst117: If source code for caller and callee of `RamTst_GetVersionInfo` is available, the RAM test module should realize `RamTst_GetVersionInfo` as a macro, defined in the module's header file.

RamTst222 : 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

RamTst193:

Service name:	RamTst_GetAlgParams	

Syntax:	`RamTst_AlgParamsIdType RamTst_GetAlgParams(`
	`)`	

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

RamTst194: 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 function `RamTst_GetAlgParams` requires the configuration parameter `RamTstGetAlgParamsApi` within the container `RamTstCommon`.

8.3.12 RamTst_GetTestAlgorithm

RamTst106:

Service name:	RamTst_GetTestAlgorithm	

Syntax:	`RamTst_AlgorithmType RamTst_GetTestAlgorithm(`
	
	`)`	

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

RamTst021: The function `RamTst_GetTestAlgorithm` shall return the current RAM Test algorithm.

The function `RamTst_GetTestAlgorithm` requires the configuration parameter `RamTstGetTestAlgorithmApi` within the container `RamTstCommon.`

8.3.13 RamTst_GetNumberOfTestedCells

RamTst108:

Service name:	RamTst_GetNumberOfTestedCells	

Syntax:	`RamTst_NumberOfTestedCellsType`
	`RamTst_GetNumberOfTestedCells(`
	
	`)`	

Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (in-out):	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.	

RamTst034: The function `RamTst_GetNumberofTestedCells` shall read the current number of tested cells per cycle.

The function `RamTst_GetNumberOfTestedCells` requires the configuration parameter `RamTstGetNumberOfTestedCellsApi` in 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 (in-out):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service used to set the test algorithm and its parameter set.	

RamTst083: 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.

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.

RamTst085: The function RamTst_SelectAlgParams shall re-initialize all RAM Test relevant registers and global variables with the values for "NewAlgParamsId".

RamTst084: 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.

Hint: "Out of range" means, that the "NewAlgParamsId" does not match to one of the configured values for RamTstAlgParams/RamTstAlgParamsId, see RamTst179_Conf.

RamTst097: 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.

RamTst094: The function RamTst_SelectAlgParams shall initialize the test result status (according to RamTst207 and RamTst202).

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

RamTst107:

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 (in-out):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service changes the current number of tested cells.	

RamTst036: The function RamTst_ChangeNumberOfTestedCells shall change the current number of tested cells (for background tests).

RamTst040: If DET is enabled and the parameter NewNumberOfTestedCells is out of range
(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.

RamTst095: 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.

The function RamTst_ChangeNumberOfTestedCells also requires the configuration parameter RamTstChangeNumOfTestedCellsApi in the container RamTstCommon.

8.3.16 RamTst_RunFullTest

RamTst195:

Service name:	RamTst_RunFullTest

Syntax:	`void RamTst_RunFullTest(`
	
	`)`

Service ID[hex]:	0x10

Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (in-out):	None
Parameters (out):	None
Return value:	None
Description:	Service for executing the full RAM Test in the foreground.

RamTst196: If the RAM Test execution status is RAMTST_EXECUTION_STOPPED, the function RamTst_RunFullTest shall test all RAM blocks defined in the selected RamTstAlgParams.

RamTst210: 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.

RamTst211: 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.

RamTst212: The function RamTst_RunFullTest shall update the test result status of single blocks according to RamTst207.

RamTst213: The function RamTst_RunFullTest shall update the overall test result status according to RamTst202. 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.

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 RamTst200 and RamTst201.

For pre-conditions on the function RamTst_RunFullTest, see requirement RamTst002.

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

RamTst197:

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 (in-out):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service for testing one RAM block in the foreground.	

RamTst198: If the RAM Test execution status is `RAMTST_EXECUTION_STOPPED` or `RAMTST_EXECUTION_SUSPENDED`, the function `RamTst_RunPartialTest` shall test the specified RAM block.

RamTst214: 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.

RamTst215: 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).

RamTst216: The function `RamTst_RunPartialTest` shall update the test result status of the tested block according to RamTst207. 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.

RamTst217: 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.

RamTst223 : 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 [**Ramtst223**] 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 RamTst200 and RamTst201.

For pre-conditions on the function RamTst_RunPartialTest, see requirement RamTst002.

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

The Basic Software Scheduler calls these functions directly. The following functions shall have no return value and no parameter. All functions shall be non-reentrant.

Terms and definitions:

Fixed cyclic: Fixed cyclic means that one cycle time is defined at configuration and shall not be changed because functionality is requiring that fixed timing (e.g. filters).

Variable cyclic: Variable cyclic means that the cycle times are defined at configuration but might be mode dependent and therefore vary during runtime.

On pre-condition: On pre-condition means that no cycle time can be defined. The function is called when the conditions are fulfilled. Alternatively, the function may be called cyclically, however the cycle time is assigned dynamically during runtime by other modules.

8.5.1 RamTst_MainFunction

RamTst110:

Service name:	RamTst_MainFunction

Syntax:	`void RamTst_MainFunction(`
	
	`)`

Service ID[hex]:	0x01
Timing:	VARIABLE_CYCLIC_OR_ON_PRECONDITION
Description:	Scheduled function for executing the RAM Test in the background.

RamTst008: 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` .

RamTst009: If the RAM Test execution status is `RAMTST_EXECUTION_RUNNING`, the function `RamTst_MainFunction` shall start testing with the first RAM block in the selected `RamTstAlgParams`.

RamTst175: 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.

RamTst010: The function `RamTst_MainFunction` shall update the test result status of single blocks according to RamTst207.

RamTst011: The function `RamTst_MainFunction` shall update the overall test result status according to RamTst202. 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.

RamTst047: 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.

RamTst059: 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.

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 RamTst200 and RamTst201.

For pre-conditions on the function RamTst_MainFunction, see requirement RamTst002.

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], BSW00426), leaving the actual locking method to the BSW Scheduler, or be directly implemented via interrupt locking (see [3], BSW00429). 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.

RamTst111:

API function	***Description***
Dem_ReportErrorStatus	Reports errors to the DEM.

8.6.2 Optional Interfaces

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

RamTst112:

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).

RamTst043: The callback notifications shall have no parameters and no return value.

RamTst044: If a callback notification is configured as null pointer, the RAM Test module shall not execute the callback.

8.6.3.1 RamTst_TestCompletedNotification

RamTst113:

Service name:	RamTst_TestCompletedNotification

Syntax:	`void RamTst_TestCompletedNotification(`
	`)`

Sync/Async:	--
Reentrancy:	Don't care
Parameters (in):	None
Parameters (in-out):	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.

RamTst045: 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.

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

8.6.3.2 RamTst_ErrorNotification

RamTst114:

Service name:	RamTst_ErrorNotification

Syntax:	`void RamTst_ErrorNotification(`
	`)`

Sync/Async:	--
Reentrancy:	Don't care
Parameters (in):	None
Parameters (in-out):	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.

RamTst046: 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.

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

9 Sequence Diagrams

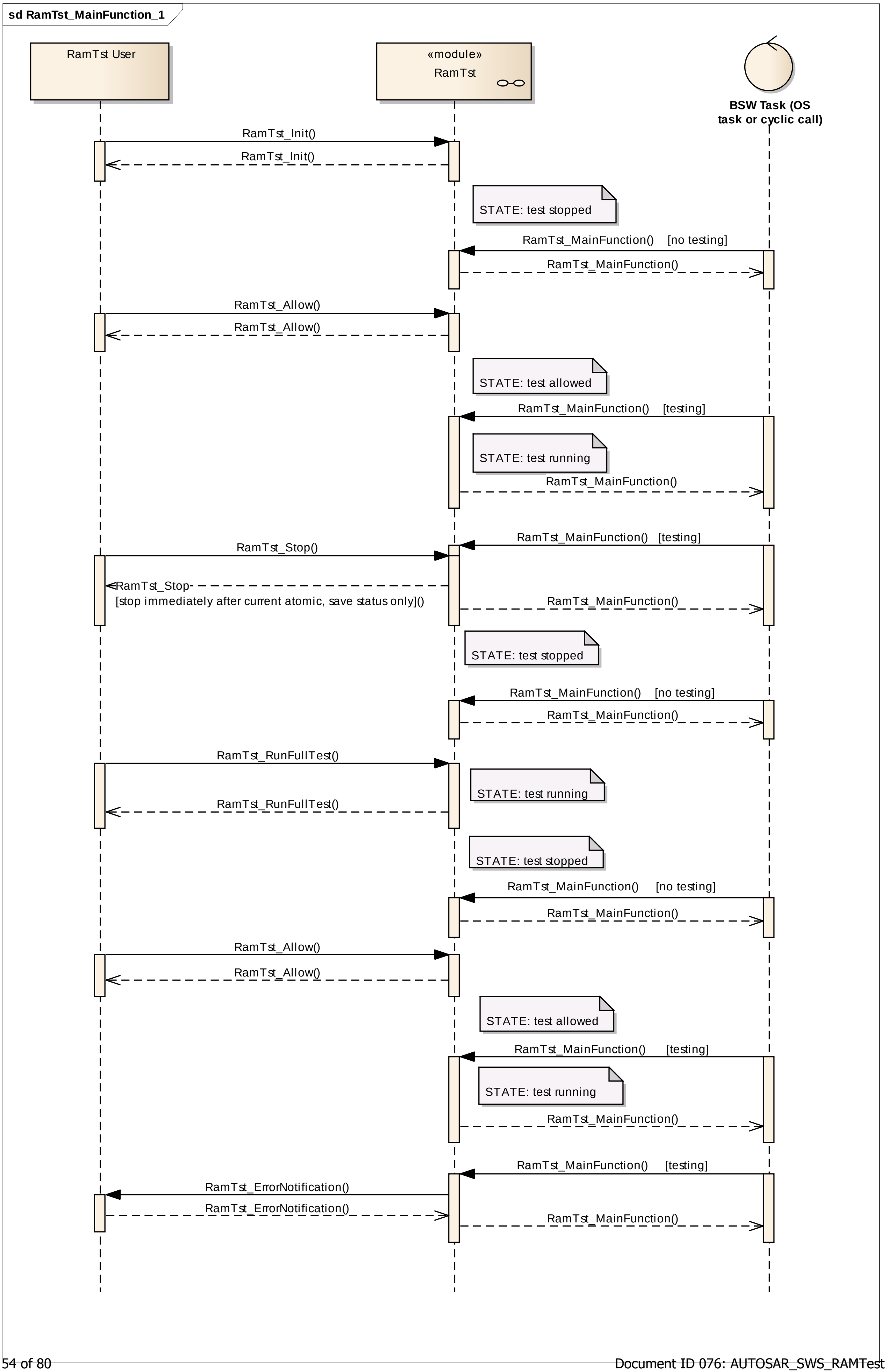
9.1 RamTst_MainFunction (Examples)

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.

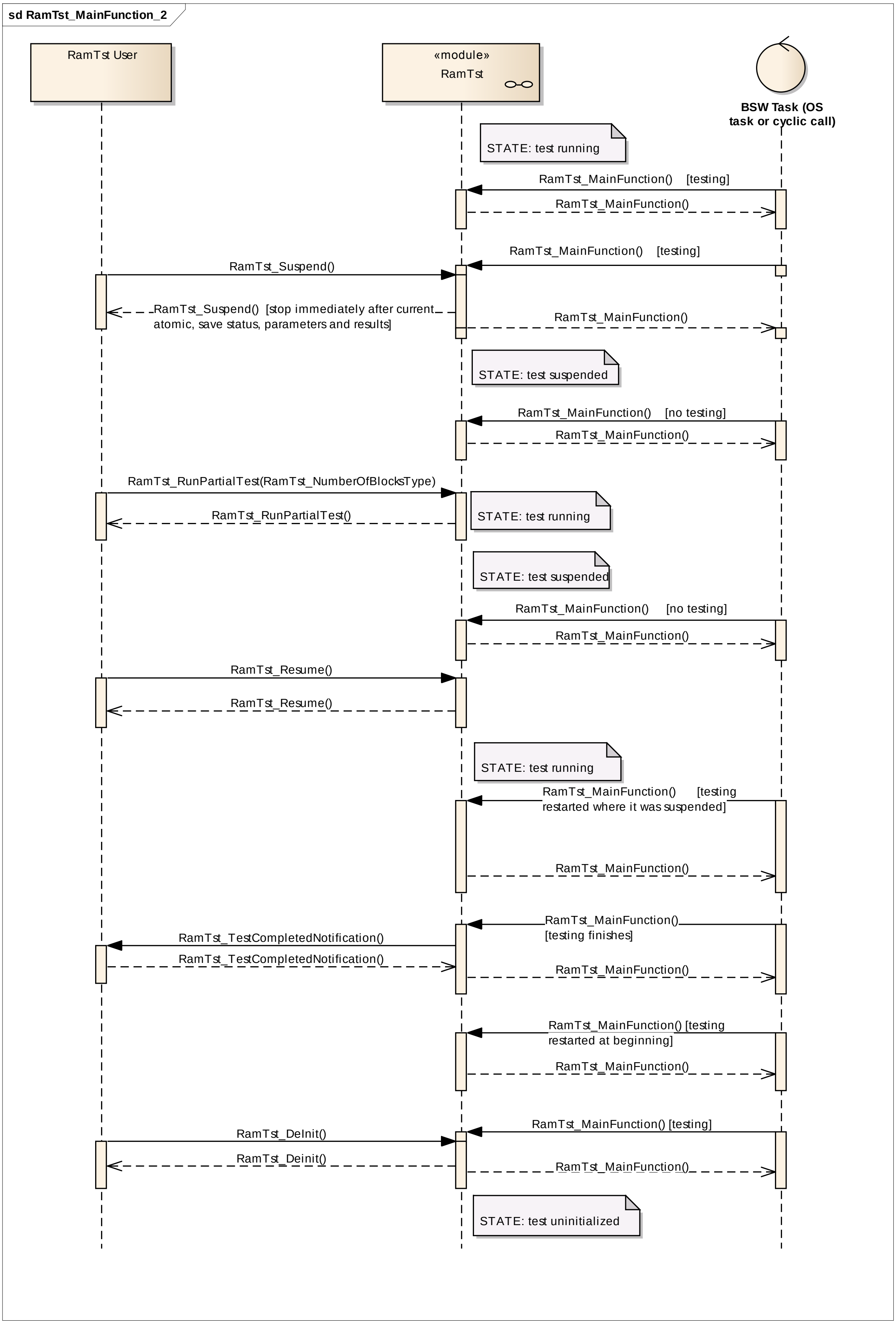
A cyclic background task called by a scheduler consists of several small atomic sequences in succession. At the end of each atomic sequence, the command variables are checked to see if any command has been received, and corresponding actions are taken.

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.

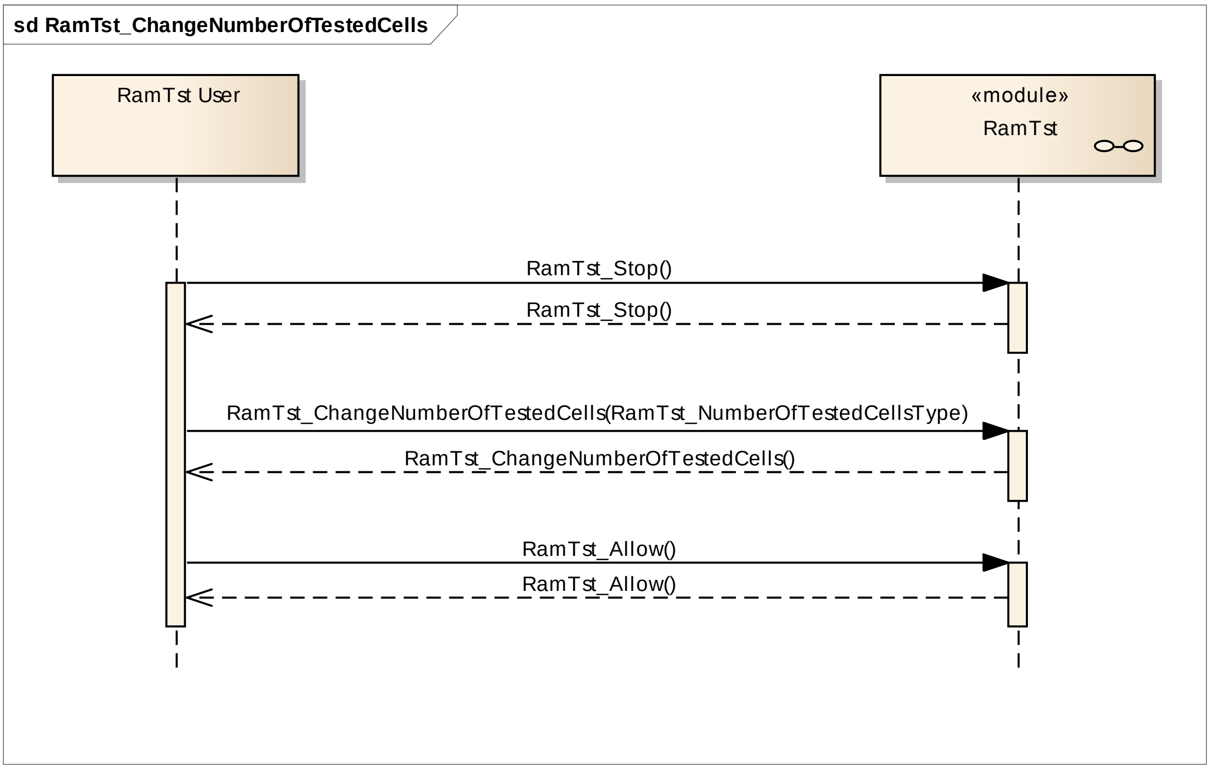
The second example shows the Suspend and Resume commands, a foreground Run Partial Test request, a Test Completed notification, and the DeInit command.



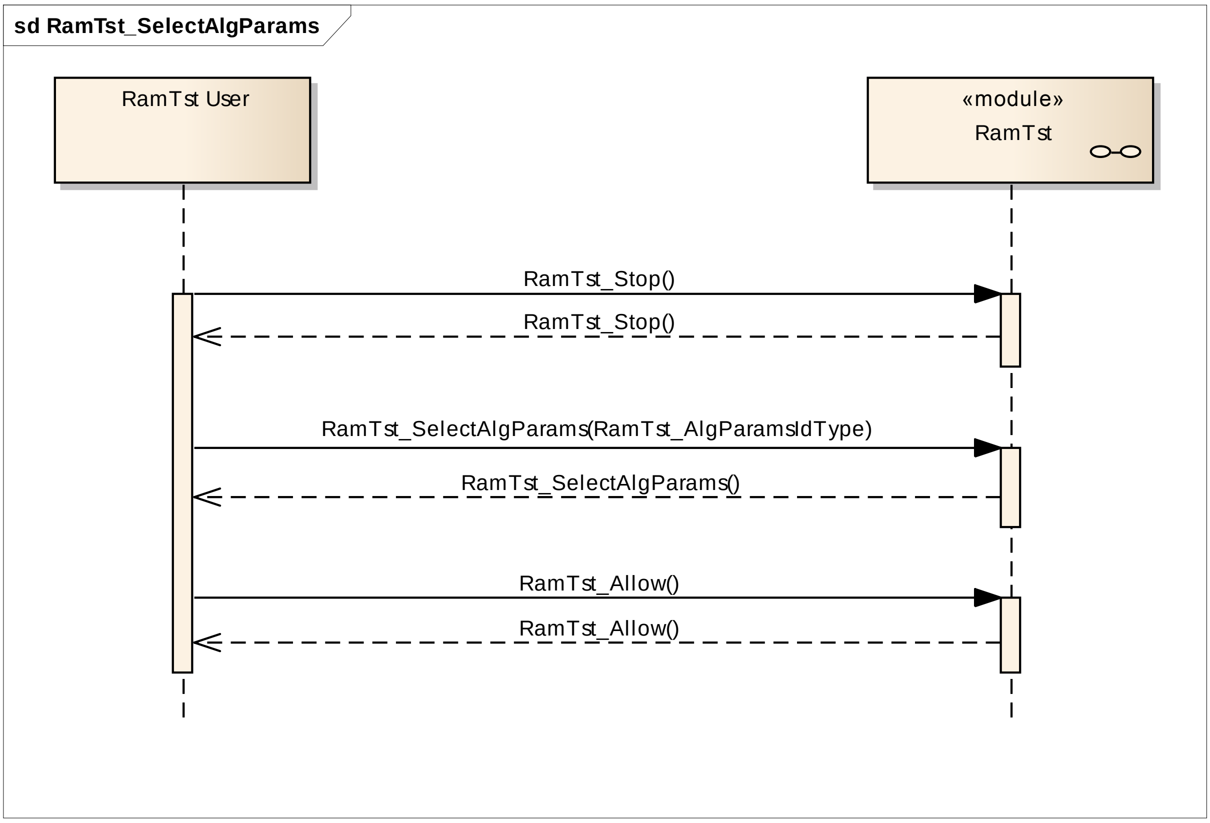
sd RamTst_MainFunction_1
RamTst User
«module»
RamTst
BSW Task (OS task or cyclic call)
RamTst_Init()
RamTst_Init()
STATE: test stopped
RamTst_MainFunction() [no testing]
RamTst_MainFunction()
RamTst_Allow()
RamTst_Allow()
STATE: test allowed
RamTst_MainFunction() [testing]
STATE: test running
RamTst_MainFunction()
RamTst_MainFunction() [testing]
RamTst_Stop()
RamTst_Stop-
[stop immediately after current atomic, save status only]()
RamTst_MainFunction()
STATE: test stopped
RamTst_MainFunction() [no testing]
RamTst_MainFunction()
RamTst_RunFullTest()
STATE: test running
RamTst_RunFullTest()
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_MainFunction() [testing]
RamTst_ErrorNotification()
RamTst_ErrorNotification()
RamTst_MainFunction()


sd RamTst_MainFunction_2
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_MainFunction() [testing]
RamTst_DeInit()
RamTst_Deinit()
RamTst_MainFunction()
STATE: test uninitialized

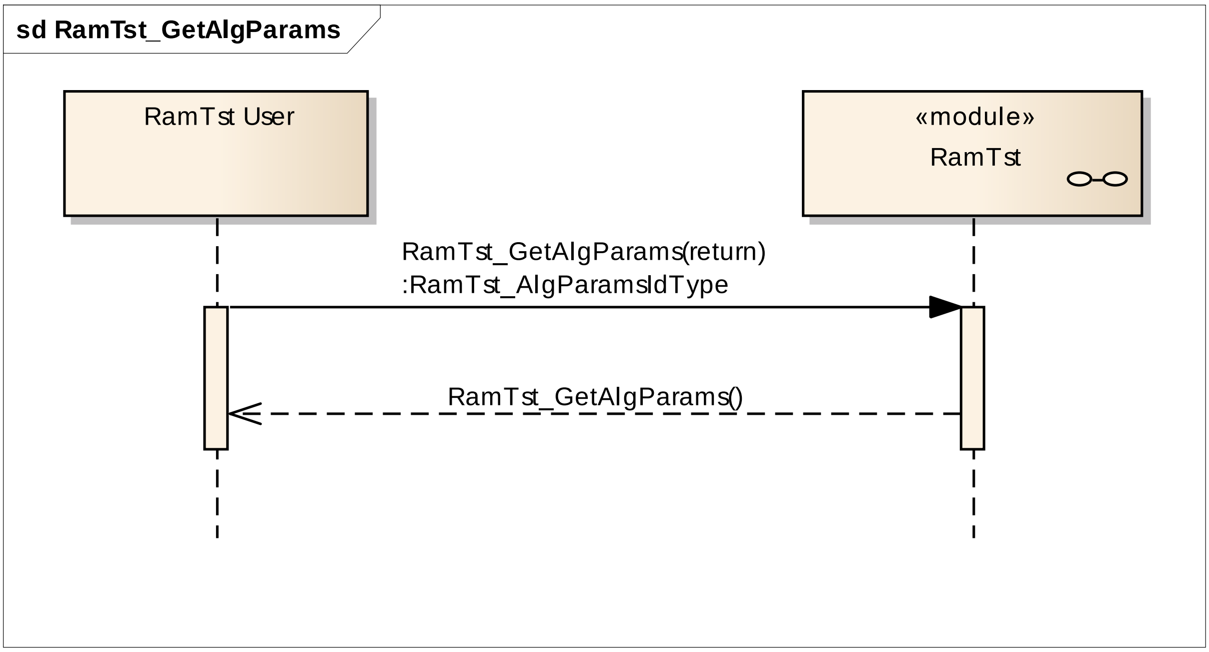
9.2 RamTst_ChangeNumberOfTestedCells


sd RamTst_ChangeNumberOfTestedCells
RamTst User
«module»
RamTst
RamTst_Stop()
RamTst_Stop()
RamTst_ChangeNumberOfTestedCells(RamTst_NumberOfTestedCellsType)
RamTst_ChangeNumberOfTestedCells()
RamTst_Allow()
RamTst_Allow()

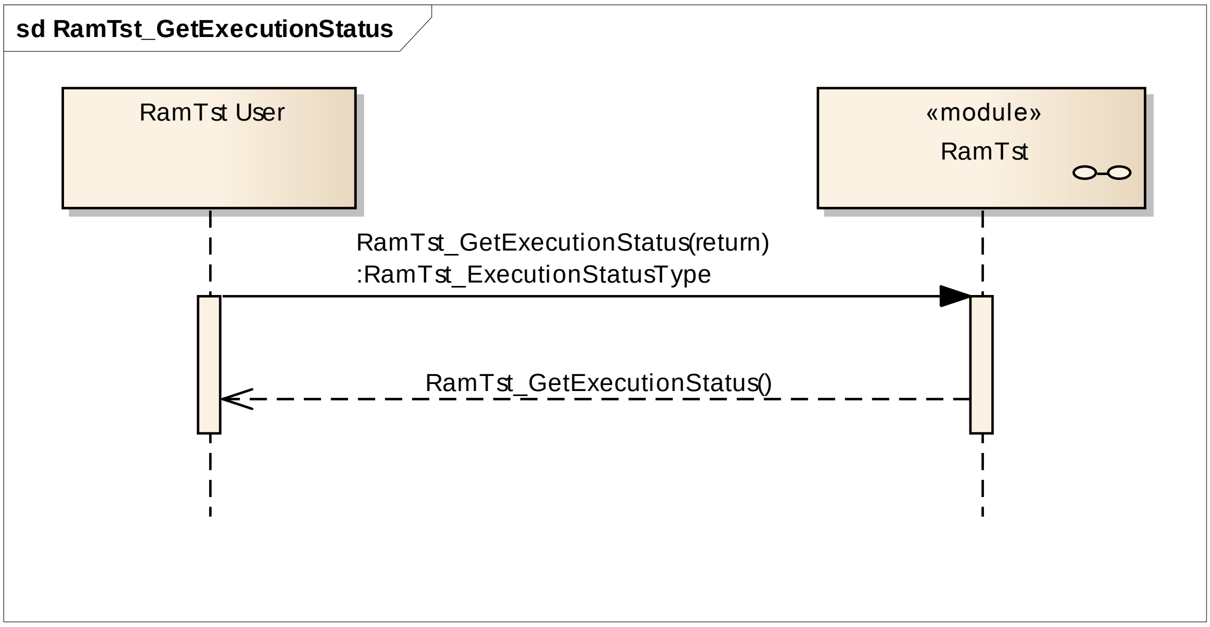
9.3 RamTst_SelectAlgParams



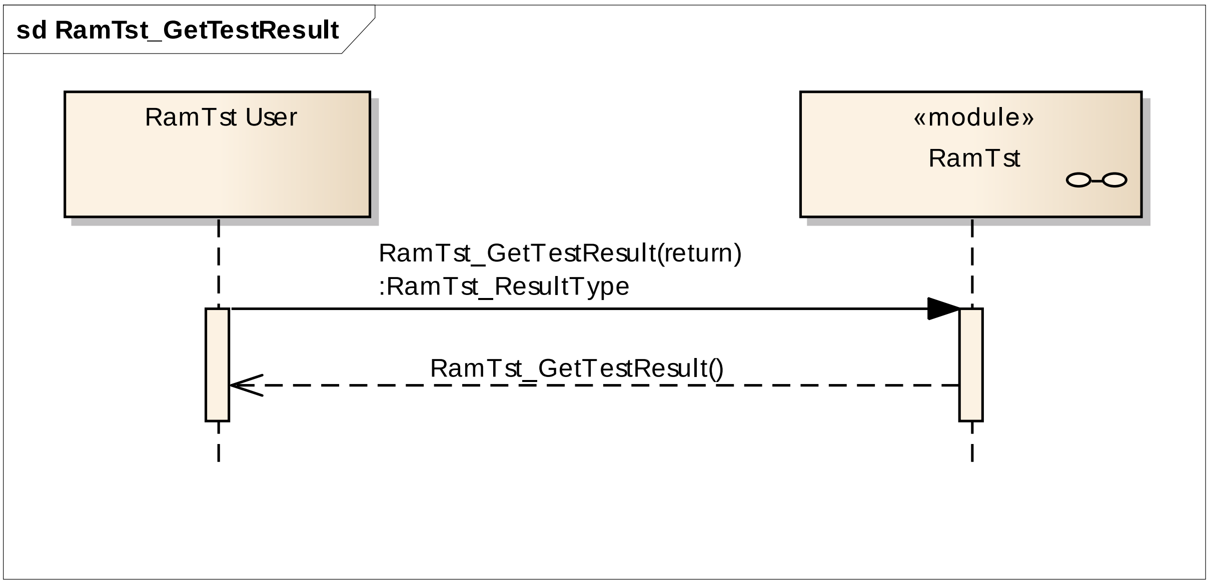
9.4 RamTst_GetAlgParams



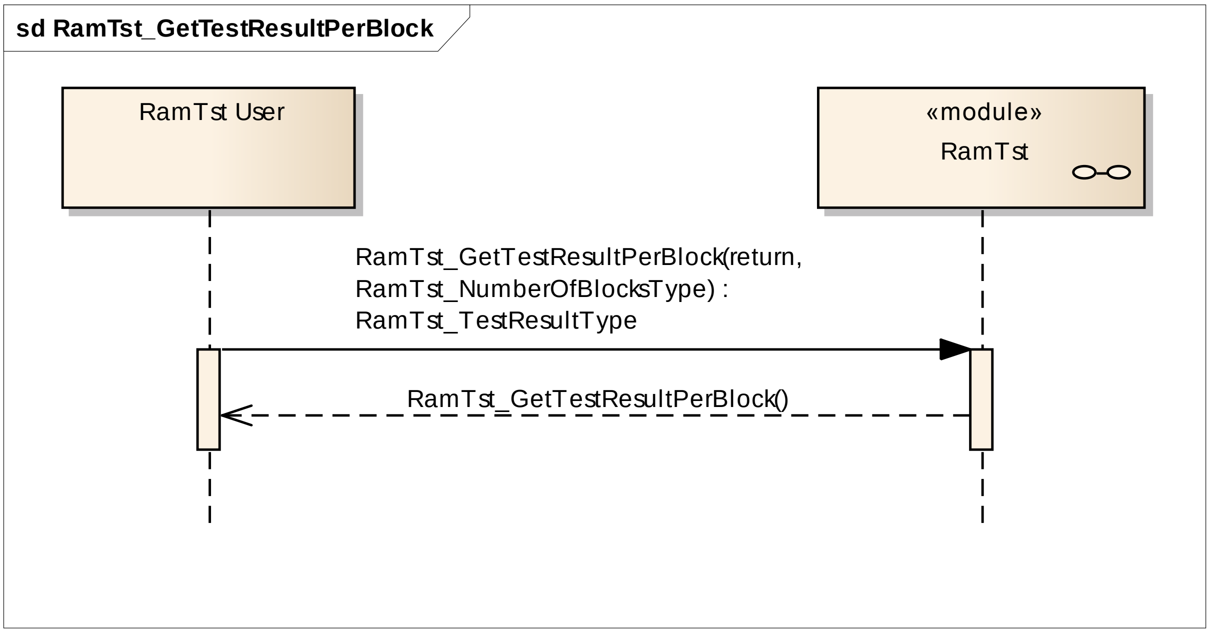
9.5 RamTst_GetExecutionStatus


sd RamTst_GetExecutionStatus
RamTst User
«module»
RamTst
RamTst_GetExecutionStatus(return)
:RamTst_ExecutionStatusType
RamTst_GetExecutionStatus()

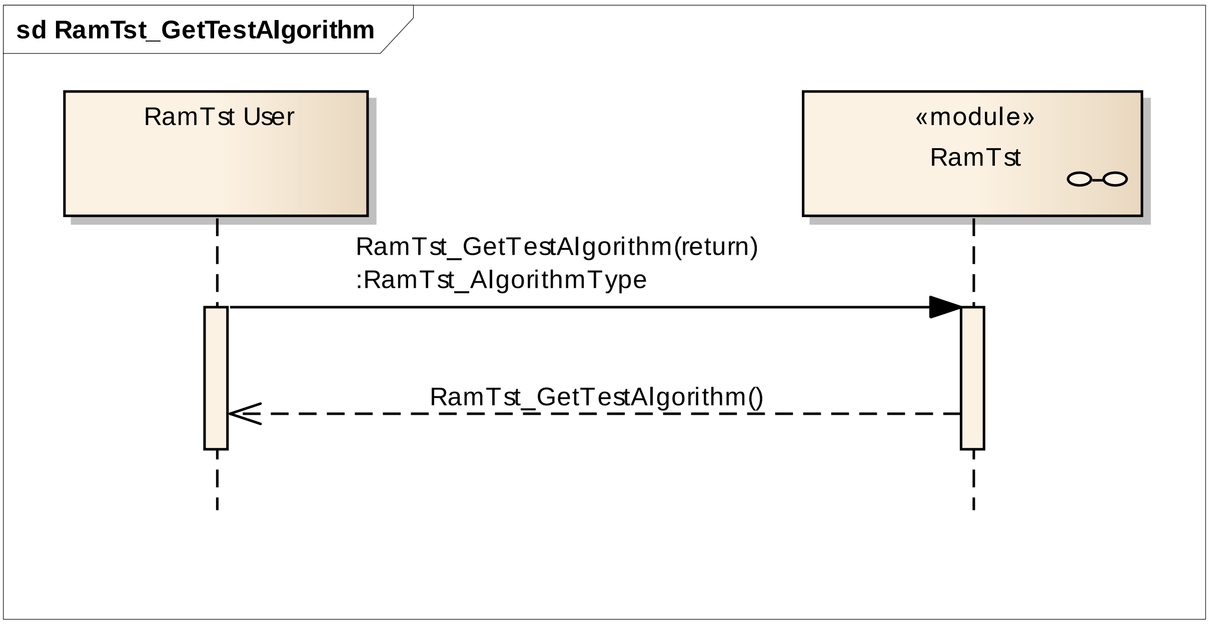
9.6 RamTst_GetTestResult



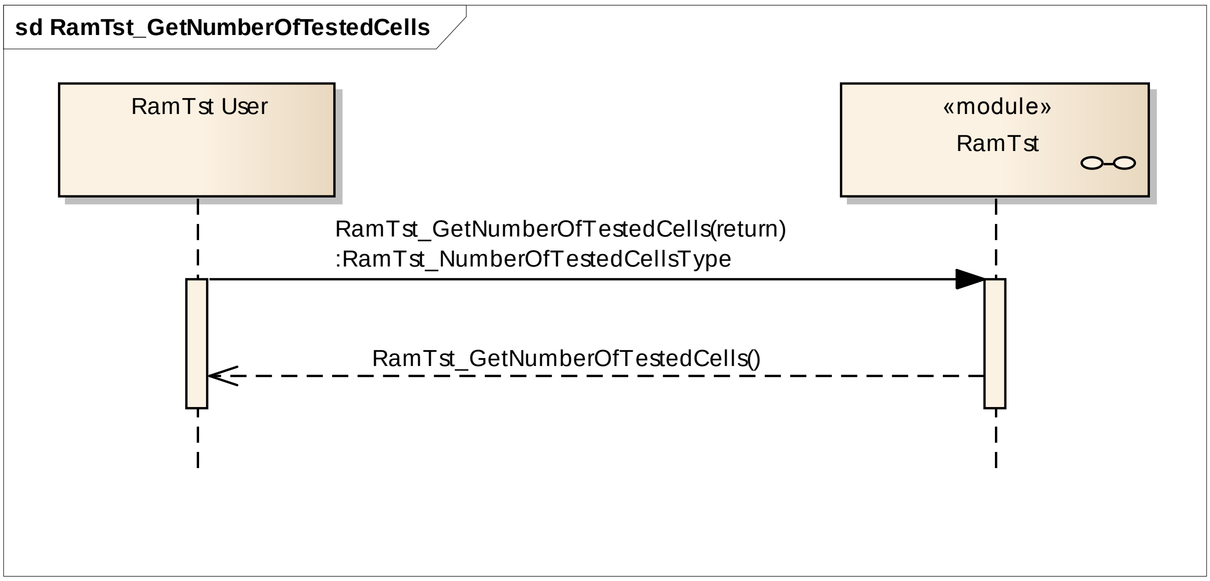
9.7 RamTst_GetTestResultPerBlock


sd RamTst_GetTestResultPerBlock
RamTst User
«module»
RamTst
RamTst_GetTestResultPerBlock(return,
RamTst_NumberOfBlocksType) :
RamTst_TestResultType
RamTst_GetTestResultPerBlock()

9.8 RamTst_GetTestAlgorithm


sd RamTst_GetTestAlgorithm
RamTst User
«module»
RamTst
RamTst_GetTestAlgorithm(return)
:RamTst_AlgorithmType
RamTst_GetTestAlgorithm()

9.9 RamTst_GetNumberOfTestedCells


sd RamTst_GetNumberOfTestedCells
RamTst User
«module»
RamTst
RamTst_GetNumberOfTestedCells(return)
:RamTst_NumberOfTestedCellsType
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

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification [5]
    - This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.1.3 Specification template for configuration parameters

The following tables consist of three sections:

- the general section
- the configuration parameter section
- the section of included/referenced containers

Pre-compile time - specifies whether the configuration parameter shall be of configuration class *Pre-compile time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Pre-compile time*.
--	The configuration parameter shall never be of configuration class *Pre-compile time*.

Link time - specifies whether the configuration parameter shall be of configuration class *Link time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Link time*.
--	The configuration parameter shall never be of configuration class *Link time*.

Post Build - specifies whether the configuration parameter shall be of configuration class *Post Build* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Post Build* and no specific implementation is required.
L	*Loadable* - the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* - the configuration parameter shall be of configuration class *Post Build* and is selected out of a set of multiple parameters by passing a dedicated to the init function of the module.
--	The configuration parameter shall never be of configuration class *Post Build*.

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.

RamTst026: 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`.

RamTst027: Within the configuration data for the RAM Test module, each parameter set of type `RamTstAlgParams` (as required in RamTst026) 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.

10.2.1 Variants

RamTst167: 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 BSW00397.

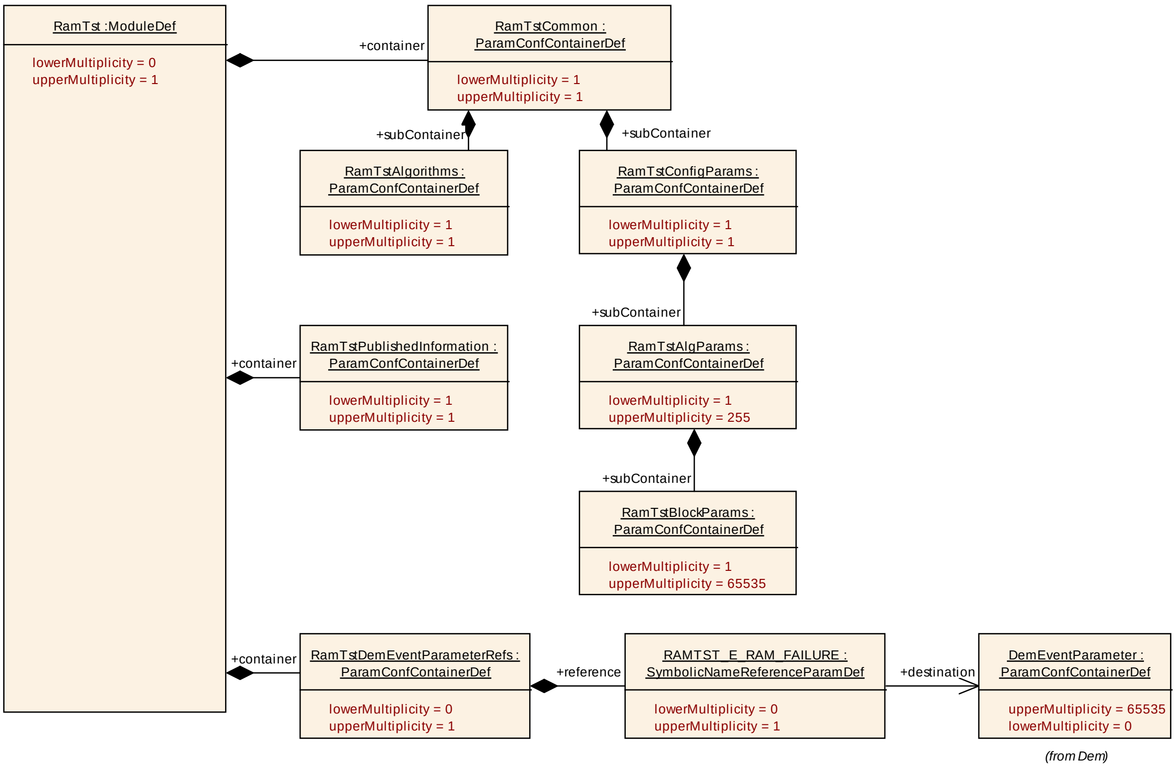
RamTst168: This module shall support the configuration variant VARIANT-LINK-TIME. Parameters with "Pre-compile time" and "Link time" are allowed in this variant. The intention of this variant is to optimize the parameter configuration for an object code delivery. See BSW00398.

RamTst093: The initialization function of this module shall always have a “void” as parameter. This means that, in contradiction to BSW00414 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.

10.2.2 RamTst

SWS Item	**RamTst150_Conf :**
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.
RamTstDemEventParameter-Refs	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	**RamTst188_Conf :**
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	**RamTst189_Conf :**		
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	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.4 RamTstCommon

SWS Item	**RamTst070_Conf :**
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	**RamTst120_Conf :**		
Name	RamTstAllowApi {RAMTST_ALLOW_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_Allow".		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst118_Conf :**		
Name	RamTstChangeNumOfTestedCellsApi {RAMTST_CHANGE_NUMBER_OF_TESTED_CELLS_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_ChangeNumberOfTestedCells".		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst121_Conf :**		
Name	RamTstDevErrorDetect {RAMTST_DEV_ERROR_DETECT}		
Description	Preprocessor switch to select the development error tracer (DET) ON or OFF		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst183_Conf :**
Name	RamTstGetAlgParamsApi {RAMTST_GET_ALG_PARAMS_API}
Description	Preprocessor switch to disable / enable the function "RamTst_GetAlgParams".

Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst122_Conf :**		
Name	RamTstGetExecutionStatusApi {RAMTST_GET_EXECUTION_STATUS_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_GetExecutionStatus"		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst123_Conf :**		
Name	RamTstGetNumberOfTestedCellsApi {RAMTST_GET_NUMBER_OF_TESTED_CELLS_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_GetNumberOfTestedCells".		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst124_Conf :**		
Name	RamTstGetTestAlgorithmApi {RAMTST_GET_TEST_ALGORITHM_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_GetTestAlgorithm"		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst125_Conf :**
Name	RamTstGetTestResultApi {RAMTST_GET_TEST_RESULT_API}
Description	Preprocessor switch to disable / enable the function "RamTst_GetTestResult"
Multiplicity	1
Type	BooleanParamDef

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

SWS Item	**RamTst126_Conf :**		
Name	RamTstGetTestResultPerBlockApi {RAMTST_GET_TEST_RESULT_PER_BLOCK_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_GetTestResultPerBlock"		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst128_Conf :**		
Name	RamTstGetVersionInfoApi {RAMTST_GET_VERSION_INFO_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_GetVersionInfo"		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst155_Conf :**		
Name	RamTstResumeApi {RAMTST_RESUME_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_Resume".		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst184_Conf :**		
Name	RamTstRunFullTestApi {RAMTST_RUN_FULL_TEST_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_RunFullTest"		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile***	X	All Variants

	time		
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst185_Conf :**		
Name	RamTstRunPartialTestApi {RAMTST_RUN_PARTIAL_TEST_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_RunPartialTest"		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

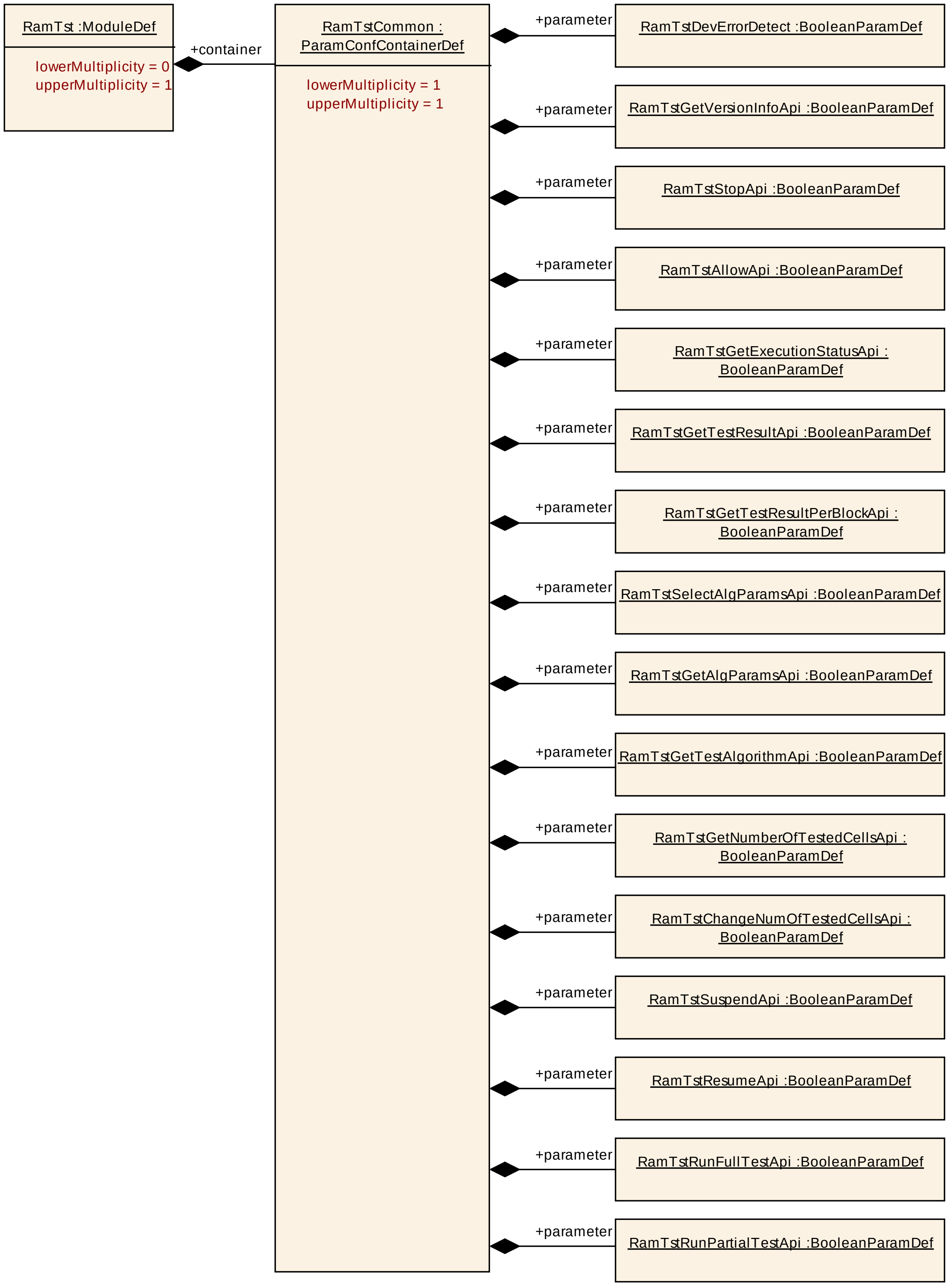
SWS Item	**RamTst182_Conf :**		
Name	RamTstSelectAlgParamsApi {RAMTST_SELECT_ALG_PARAMS_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_SelectAlgParams".		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst127_Conf :**		
Name	RamTstStopApi {RAMTST_STOP_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_Stop"		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst156_Conf :**		
Name	RamTstSuspendApi {RAMTST_SUSPEND_API}		
Description	Preprocessor switch to disable / enable the function "RamTst_Suspend".		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	scope: Module

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.


RamTst :ModuleDef
lowerMultiplicity = 0
upperMultiplicity = 1
+container
RamTstCommon : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 1
+parameter
RamTstDevErrorDetect :BooleanParamDef
RamTstGetVersionInfoApi :BooleanParamDef
RamTstStopApi :BooleanParamDef
RamTstAllowApi :BooleanParamDef
RamTstGetExecutionStatusApi : BooleanParamDef
RamTstGetTestResultApi :BooleanParamDef
RamTstGetTestResultPerBlockApi : BooleanParamDef
RamTstSelectAlgParamsApi :BooleanParamDef
RamTstGetAlgParamsApi :BooleanParamDef
RamTstGetTestAlgorithmApi :BooleanParamDef
RamTstGetNumberOfTestedCellsApi : BooleanParamDef
RamTstChangeNumOfTestedCellsApi : BooleanParamDef
RamTstSuspendApi :BooleanParamDef
RamTstResumeApi :BooleanParamDef
RamTstRunFullTestApi :BooleanParamDef
RamTstRunPartialTestApi :BooleanParamDef

10.2.5 RamTstAlgorithms

SWS Item	**RamTst065_Conf :**

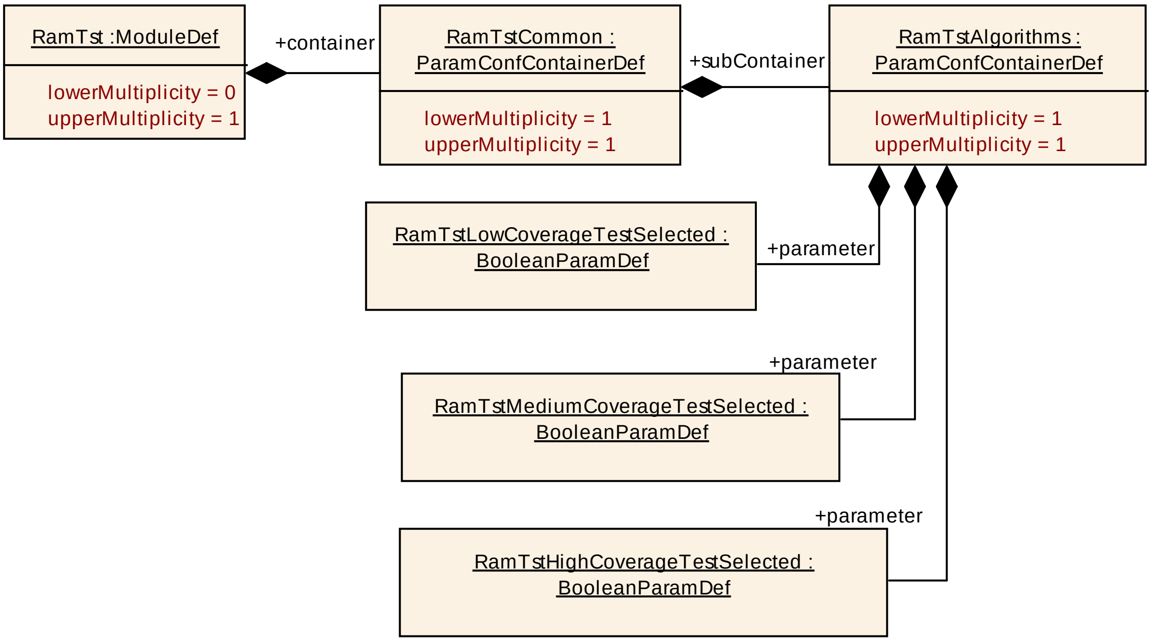
Container Name	RamTstAlgorithms{RamTst_Algorithms}
Description	This container holds all of the available test algorithms for the specific microcontroller. Each test algorithm is selected ON or OFF before compiling so that only the desired test algorithms are in the compiled code.
Configuration Parameters	

SWS Item	**RamTst192_Conf :**		
Name	RamTstHighCoverageTestSelected {RAMTST_HIGH_COVERAGE_TEST_SELECTED}		
Description	Preprocessor switch to select the test with high coverage ON or OFF		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**RamTst190_Conf :**		
Name	RamTstLowCoverageTestSelected {RAMTST_LOW_COVERAGE_TEST_SELECTED}		
Description	Preprocessor switch to select the test with low coverage ON or OFF		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**RamTst191_Conf :**		
Name	RamTstMediumCoverageTestSelected {RAMTST_MEDIUM_COVERAGE_TEST_SELECTED}		
Description	Preprocessor switch to select the test with medium coverage ON or OFF		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers


RamTst :ModuleDef
lowerMultiplicity = 0
upperMultiplicity = 1
+container
RamTstCommon : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 1
+subContainer
RamTstAlgorithms : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 1
+parameter
RamTstLowCoverageTestSelected : BooleanParamDef
+parameter
RamTstMediumCoverageTestSelected : BooleanParamDef
+parameter
RamTstHighCoverageTestSelected : BooleanParamDef

10.2.6 RamTstConfigParams

SWS Item	**RamTst066_Conf :**
Container Name	RamTstConfigParams{RamTst_ConfigParams}
Description	This container specifies configuration parameters which are set at pre-compile or link time.
Configuration Parameters	

SWS Item	**RamTst181_Conf :**		
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	IntegerParamDef		
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: Module		

SWS Item	**RamTst154_Conf :**	
Name	RamTstMinNumberOfTestedCells {RAMTST_MIN_NUMBER_OF_TESTED_CELLS}	
Description	Minimum number of tested cells for one cyle of a background test, as defined by implementer.	
Multiplicity	1	
Type	IntegerParamDef	
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: Module		

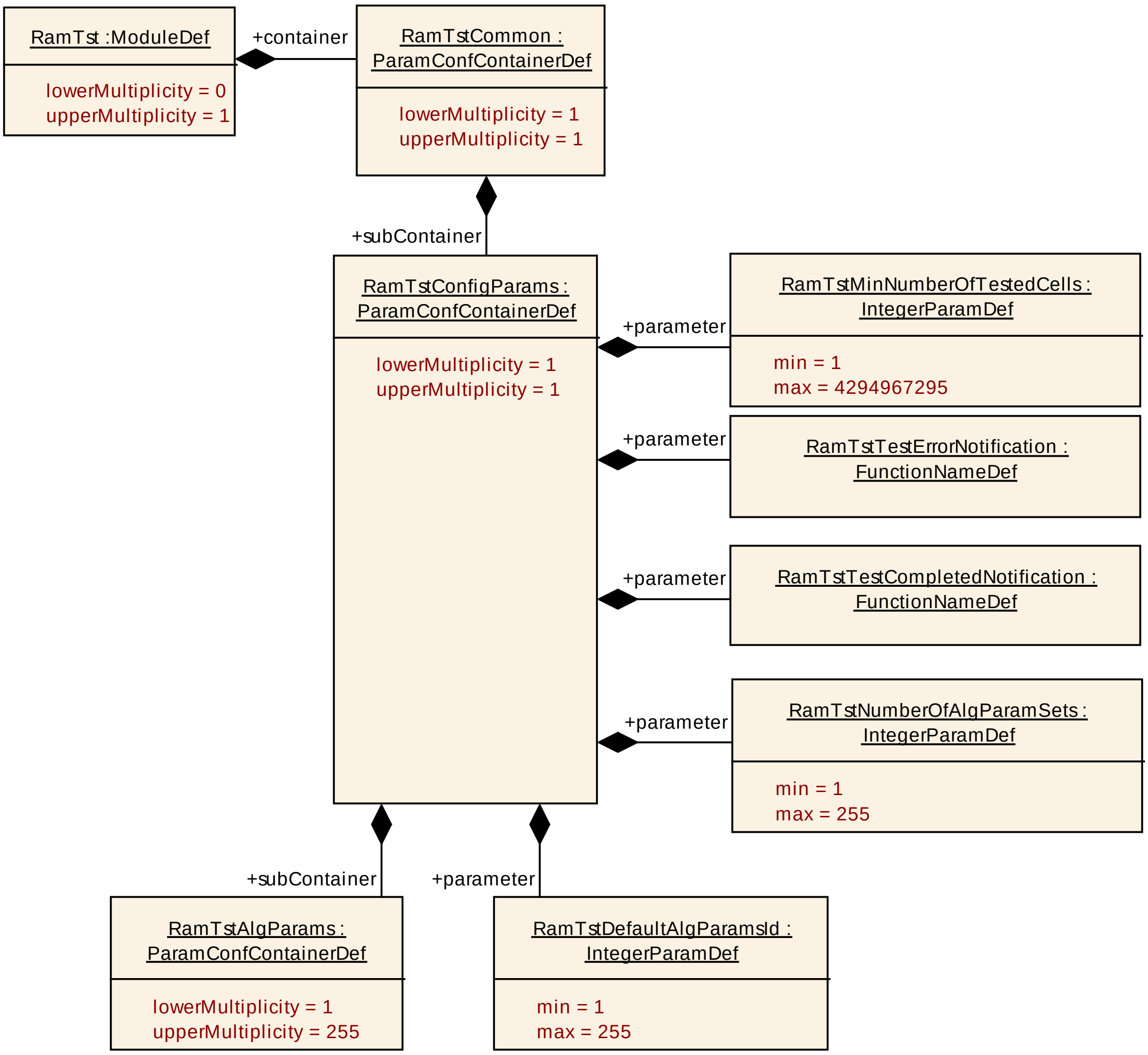
SWS Item	**RamTst180_Conf :**		
Name	RamTstNumberOfAlgParamSets {RAMTST_NUMBER_OF_ALG_PARAM_SETS}		
Description	Number of configured parameter sets for the available test algorithms. count of the container RamTstAlgParams		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module 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	**RamTst138_Conf :**		
Name	RamTstTestCompletedNotification {RAMTST_TEST_COMPLETED_NOTIFICATION}		
Description	This function will be called from abackground test after finishing the RAM test without an error.		
Multiplicity	1		
Type	FunctionNameDef		
Default value	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst139_Conf :**		
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	FunctionNameDef		
Default value	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

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 con-

		tainer 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.


RamTst :ModuleDef
lowerMultiplicity = 0
upperMultiplicity = 1
+container
RamTstCommon : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 1
+subContainer
RamTstConfigParams : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 1
+parameter
RamTstMinNumberOfTestedCells : IntegerParamDef
min = 1
max = 4294967295
+parameter
RamTstTestErrorNotification : FunctionNameDef
+parameter
RamTstTestCompletedNotification : FunctionNameDef
+parameter
RamTstNumberOfAlgParamSets : IntegerParamDef
min = 1
max = 255
+subContainer
+parameter
RamTstAlgParams : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 255
RamTstDefaultAlgParamsId : IntegerParamDef
min = 1
max = 255

10.2.7 RamTstAlgParams

SWS Item	**RamTst090_Conf :**
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	**RamTst179_Conf :**
Name	RamTstAlgParamsId {RAMTST_ALG_PARAMS_ID}
Description	This is the identifier by which this RamTstAlgParams set can be selected.

Multiplicity	1		
Type	IntegerParamDef		
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: Module		

SWS Item	**RamTst193_Conf :**		
Name	RamTstAlgorithmCoverage {RAMTST_ALGORITHM_COVERAGE}		
Description	This is the coverage of the defined RamTstAlgParams set. 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	EnumerationParamDef		
Range	RAMTST_ALGORITHM_COVERAGE_HIGH		
	RAMTST_ALGORITHM_COVERAGE_LOW		
	RAMTST_ALGORITHM_COVERAGE_MEDIUM		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**RamTst152_Conf :**		
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	IntegerParamDef		
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: Module		

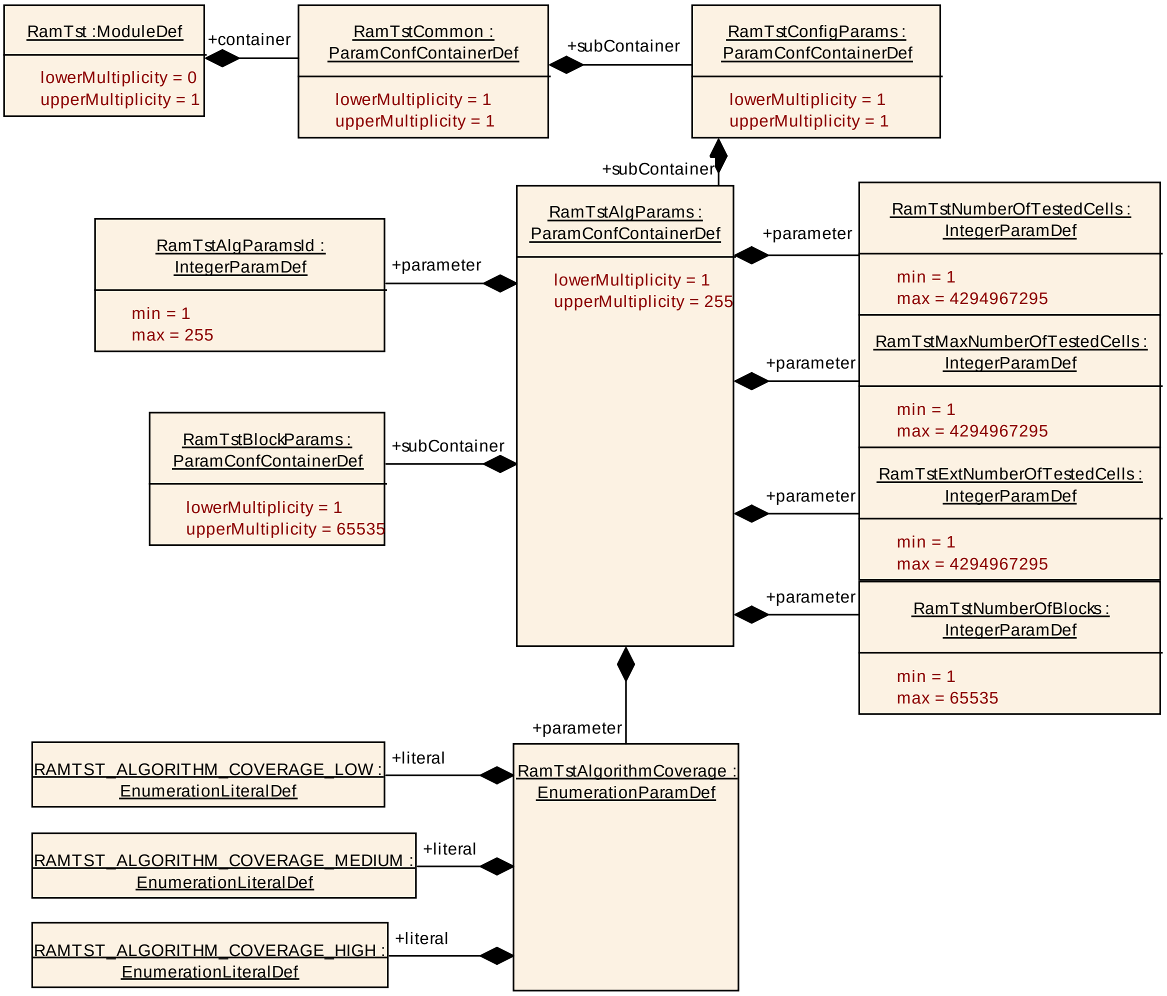
SWS Item	**RamTst153_Conf :**		
Name	RamTstMaxNumberOfTestedCells {RAMTST_MAX_NUMBER_OF_TESTED_CELLS}		
Description	This is the maximum value for the number of cells that can be tested in one cycle of a background test.		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**RamTst141_Conf :**

Name	RamTstNumberOfBlocks {RAMTST_NUMBER_OF_BLOCKS}		
Description	Number of RAM blocks configured using the container "RamTst_BlockParams" Count of RamTstBlockParams contained in this RamTstAlgParams.		
Multiplicity	1		
Type	IntegerParamDef		
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: Module 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	**RamTst142_Conf :**		
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	IntegerParamDef		
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: Module		

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.


RamTst :ModuleDef
lowerMultiplicity = 0
upperMultiplicity = 1
+container
RamTstCommon : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 1
+subContainer
RamTstConfigParams : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 1
+subContainer
RamTstAlgParams : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 255
RamTstAlgParamsId : IntegerParamDef
min = 1
max = 255
+parameter
RamTstBlockParams : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 65535
+subContainer
+parameter
RamTstNumberOfTestedCells : IntegerParamDef
min = 1
max = 4294967295
+parameter
RamTstMaxNumberOfTestedCells : IntegerParamDef
min = 1
max = 4294967295
+parameter
RamTstExtNumberOfTestedCells : IntegerParamDef
min = 1
max = 4294967295
+parameter
RamTstNumberOfBlocks : IntegerParamDef
min = 1
max = 65535
+parameter
RamTstAlgorithmCoverage : EnumerationParamDef
+literal
RAMTST_ALGORITHM_COVERAGE_LOW : EnumerationLiteralDef
+literal
RAMTST_ALGORITHM_COVERAGE_MEDIUM : EnumerationLiteralDef
+literal
RAMTST_ALGORITHM_COVERAGE_HIGH : EnumerationLiteralDef

10.2.8 RamTstBlockParams

SWS Item	**RamTst091_Conf :**
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	**RamTst143_Conf :**		
Name	RamTstBlockId {RAMTST_BLOCK_ID}		
Description	ID of the RAM block		
Multiplicity	1		
Type	IntegerParamDef		
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: Module		

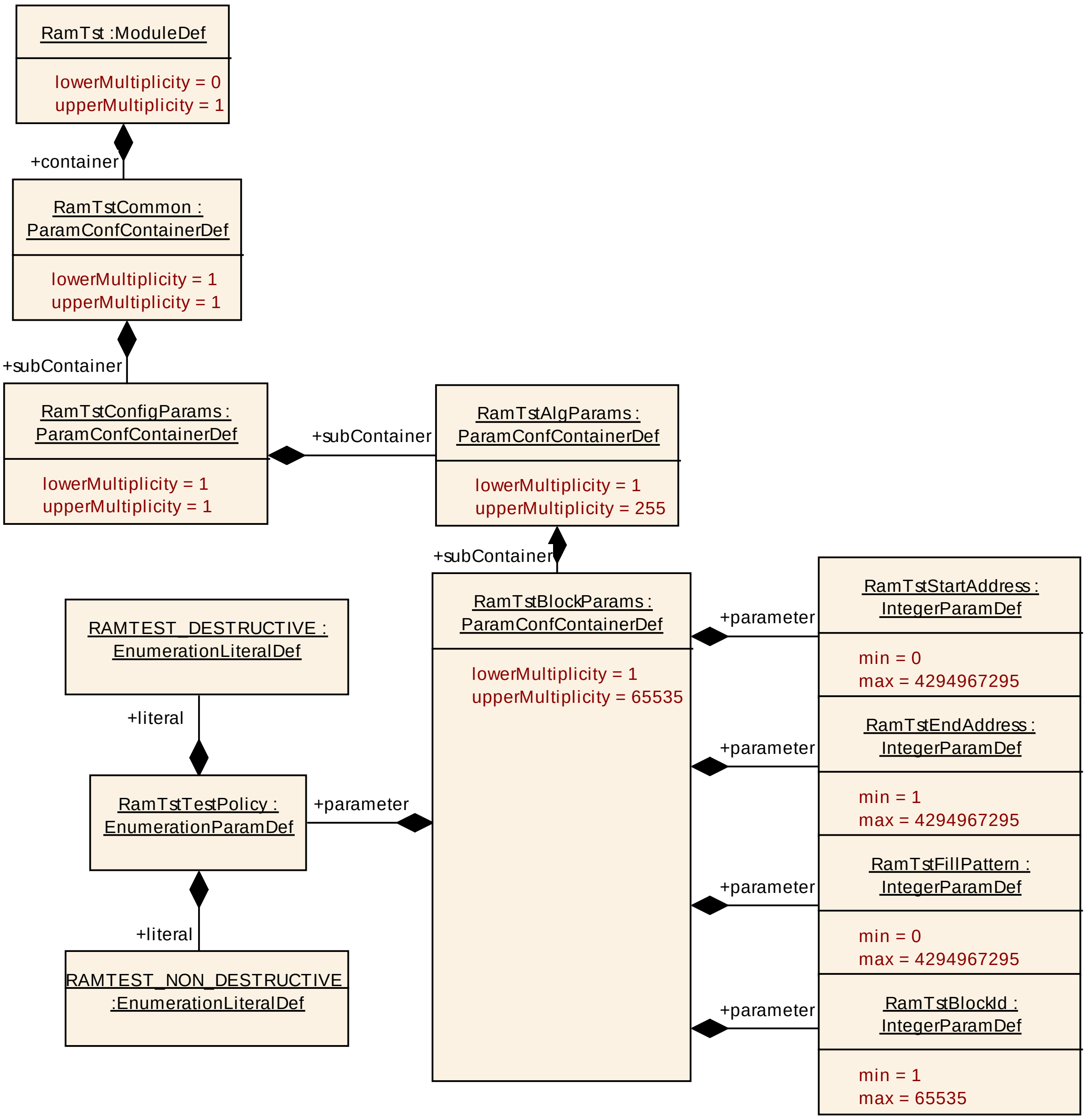
SWS Item	RamTst144_Conf :		
Name	RamTstEndAddress {RAMTST_END_ADDRESS}		
Description	End Address of the RAM block. Constraint: It must be larger than the RamTstStartAddress.		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 4294967295		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	RamTst176_Conf :		
Name	RamTstFillPattern {RAMTST_FILL_PATTERN}		
Description	Pattern to be filled into each memory cell after destructive test of this block.		
Multiplicity	1		
Type	IntegerParamDef		
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: Module		

SWS Item	RamTst145_Conf :		
Name	RamTstStartAddress {RAMTST_START_ADDRESS}		
Description	Start Address of the RAM block. Constraint: It must be smaller than the RamTstEndAddress.		
Multiplicity	1		
Type	IntegerParamDef		
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: Module		

SWS Item	RamTst177_Conf :		
Name	RamTstTestPolicy {RAMTST_TEST_POLICY}		
Description	Policy regading destruction or non-destruction of memory content.		
Multiplicity	1		
Type	EnumerationParamDef		
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: Module		

No Included Containers


RamTst :ModuleDef
lowerMultiplicity = 0
upperMultiplicity = 1
+container
RamTstCommon : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 1
+subContainer
RamTstConfigParams : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 1
+subContainer
RamTstAlgParams : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 255
+subContainer
RamTstBlockParams : ParamConfContainerDef
lowerMultiplicity = 1
upperMultiplicity = 65535
+parameter
RamTstStartAddress : IntegerParamDef
min = 0
max = 4294967295
+parameter
RamTstEndAddress : IntegerParamDef
min = 1
max = 4294967295
+parameter
RamTstFillPattern : IntegerParamDef
min = 0
max = 4294967295
+parameter
RamTstBlockId : IntegerParamDef
min = 1
max = 65535
RAMTEST_DESTRUCTIVE : EnumerationLiteralDef
+literal
RamTstTestPolicy : EnumerationParamDef
+parameter
+literal
RAMTEST_NON_DESTRUCTIVE :EnumerationLiteralDef

10.3 Published Parameters

RamTst166: The standardized common published parameters as required by BSW00402 in the SRS General on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description (see [6]).

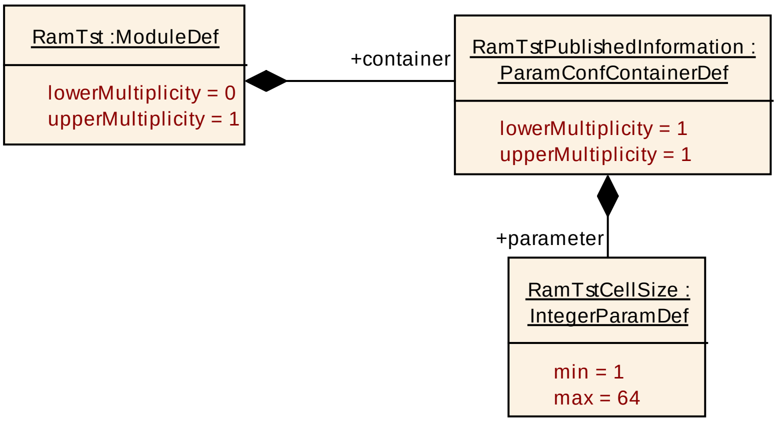
Additional module-specific published parameters are listed below if applicable.

10.3.1 RamTstPublishedInformation

SWS Item	RamTst186_Conf :
Container Name	RamTstPublishedInformation{RamTst_PublishedInformation}
Description	Container holding all RamTst specific published information parameter.
Configuration Parameters	

SWS Item	RamTst187_Conf :		
Name	RamTstCellSize {RAMTST_CELL_SIZE}		
Description	Size of RAM cells (in bits) which can be tested individually by the given implementation.		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 64		
Default value	--		
ConfigurationClass	Published Information	X	All Variants
Scope / Dependency	scope: Module		

No Included Containers



10.4 Implementation Specific Information and Parameters

RamTst081: 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.

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

RamTst205: 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).