MICROSAR RAMTST

Technical Reference

Version 1.01.00

Authors	Christoph Ederer, Christian Leder
Status	Released

Document Information

History

Author	Date	Version	Remarks
Christoph Ederer	2011-05-19	1.00.00	Initial version of the document
Christoph Ederer	2011-08-29	1.00.01	> Update Figure 2-2 and Table 4-1 > Reworked Table 4-3 > Further small modifications
Bethina Mausz	2014-06-26	1.00.02	Config Parameter MainFunctionPeriod added
Christian Leder	2015-01-08	1.01.00	> Small modifications due to change of generation tool (CFG5) > Update of Figure 2-2 > Update of Figure 4-1 > Usage of new template 5.9.0

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_RAMTest.pdf	V1.5.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	V4.2.0
[4]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	V1.6.0

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.00.00	> Foreground and background testing of configurable RAM areas > Algorithm-based configuration, each algorithm parameter set can contain several memory blocks > The following algorithms are available (according to IEC 61508): Abraham, Checkerboard, Galpat, March (MATS), WalkPath, Transparent Galpat
3.00.00	> Change of generator (for DaVinci Configurator Pro 5)

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module RamTst as specified in [1].

Supported AUTOSAR Release*:	4.0.x	
Supported Configuration Variants:	pre-compile, link-time	
Vendor ID:	RAMTST_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	RAMTST_MODULE_ID	093 decimal (according to ref. [4])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

This document describes the functionality and the API of the AUTOSAR RAM Test. The module implements an interface in C programming language.

The RAM Test checks RAM cells for physical health. Checking of the data content of the configured RAM areas is neither possible, nor intended.

In the current implementation there are (according to IEC 61508) six different test algorithms available. They have different complexities and have – depending on their complexity – different fault detection rates. The selection of a test algorithm should at best be based on a safety analysis of the ECU that considers the necessary diagnostic coverage rates.

All RAM testing is uninterruptable, so that an expected pattern or expected data cannot be modified by another task before the RAM Test module has finished it's testing. Nevertheless, it is possible to adapt the RAM testing to the available resources at any time, because for each algorithm parameter set, the test can be executed on the whole as a foreground test or alternatively as a main-function-driven background test, whereas only the single main function calls are uninterruptible.

2.1 Architecture Overview

The following figure shows where the RamTst is located in the AUTOSAR architecture.

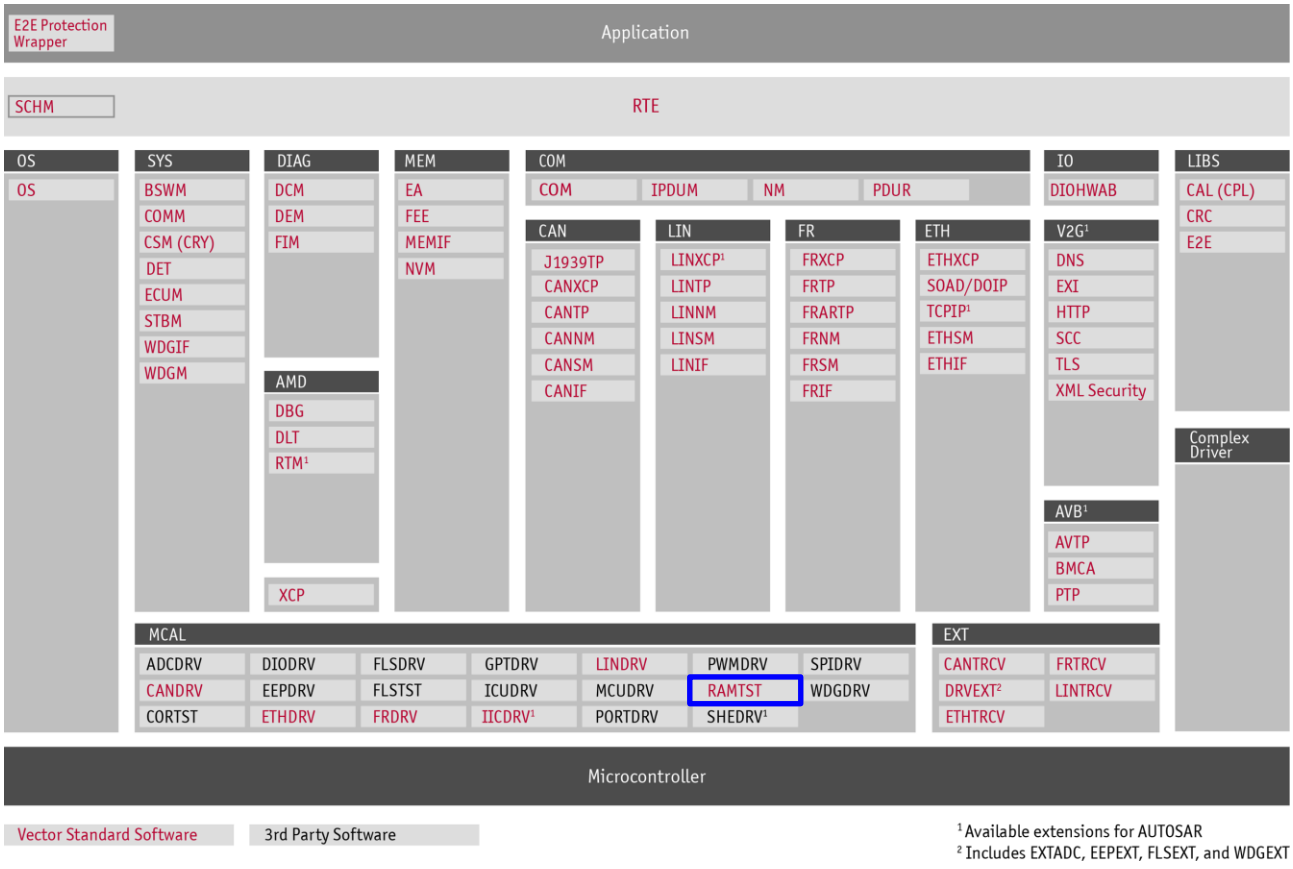

Figure 2-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the RamTst. These interfaces are described in chapter 5.

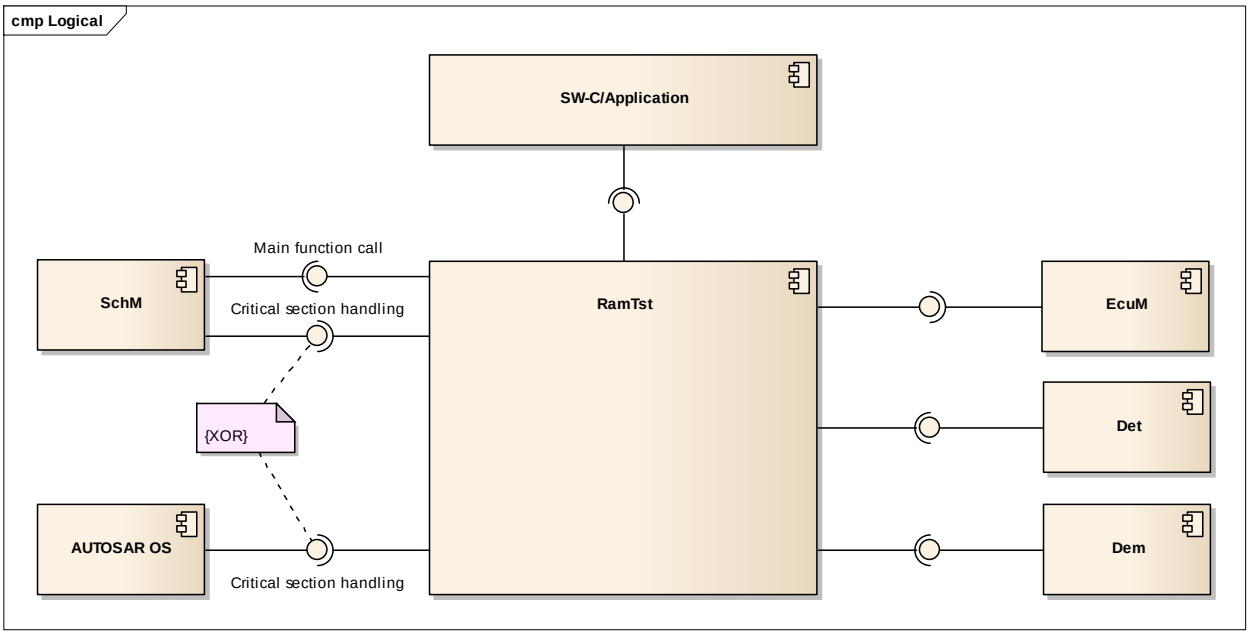

Figure 2-2 Interfaces to adjacent modules of the RamTst

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the RamTst.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further RamTst functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
> Services to initialize and de-initialize the driver
> Services for executing a full or partial foreground test
> Services for starting, stopping and suspending a background test
> Services for retrieving information about the test execution status, the test result (overall, per block), the currently selected test algorithm, the currently configured number of tested cells
> Services for selecting the algorithm parameter set and configuring the number of tested cells
> Support of the test algorithms Abraham, Checkerboard, March (MATS), Galpat, WalkPath, Transparent Galpat

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

The following features specified in [1] are not supported:

Not Supported AUTOSAR Standard Conform Features
n/a

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Configurability of development error reporting: Development error detection and reporting itself can be en-/ or disabled separately.

Table 3-3 Features provided beyond the AUTOSAR standard

3.1.3 Limitations

There are no limitations.

3.2 Initialization

The RAM Test module is being initialized by calling `RamTst_Init`(). All global variables are initialized by calling `RamTst_InitMemory`(). So, `RamTst_InitMemory`() has to be called prior to `RamTst_Init`(). To re-initialize the module, call `RamTst_DeInit`() followed by `RamTst_Init`(). After initialization the module is in the state `RAMTST_EXECUTION_STOPPED`.

3.3 States

This module implements the following states:

Module State	Description
`RAMTST_EXECUTION_UNINIT`	The module is not initialized. No API services but `RamTst_Init()` and `RamTst_GetVersion()` can be called.
`RAMTST_EXECUTION_STOPPED`	The module is initialized and ready for running a foreground test or for being switched to the allowed state to run a background test.
`RAMTST_EXECUTION_RUNNING`	A testing procedure is currently running. This does not necessarily mean that a continuous test is running at the moment. For a background test, it means that the module will test small parts of the configured RAM blocks with every call of `RamTst_MainFunction()`.
`RAMTST_EXECUTION_SUSPENDED`	This state means that a running background test has been suspended by calling the service `RamTst_Suspend()`. The suspended test can be continued by calling the service `RamTst_Resume()` and further calls of the main function.
`RAMTST_EXECUTION_ALLOWED`	The module is allowed to start a background test. By calling the service `RamTst_MainFunction()` cyclically, all configured blocks of the currently selected algorithm parameter set will be tested.

Table 3-4 RAM Test execution states

The following table shows, which API service is permitted to be called in which module execution state:

API Service \ Execution State	RAMTST_EXECUTION_STOPPED	RAMTST_EXECUTION_RUNNING	RAMTST_EXECUTION_ALLOWED	RAMTST_EXECUTION_SUSPENDED	RAMTST_EXECUTION_UNINIT
RamTst_Init					■
RamTst_RunFullTest	■				
RamTst_RunPartialTest	■			■	
RamTst_Suspend		■	■		
RamTst_Resume				■	
RamTst_Stop		■	■	■	
RamTst_Allow	■				
RamTst_DeInit	■	■	■	■	
RamTst_GetVersionInfo	■	■	■	■	■
RamTst_ExecutionStatus	■	■	■	■	
RamTst_GetTestResult	■	■	■	■	
RamTst_GetTestResultPerBlock	■	■	■	■	
RamTst_GetAlgParams	■	■	■	■	
RamTst_GetTestAlgorithm	■	■	■	■	
RamTst_GetNumberOfTestedCells	■	■	■	■	
RamTst_SelectAlgParams	■				
RamTst_ChangeNumberOfTestedCells	■				

Table 3-5 API service to execution state mapping

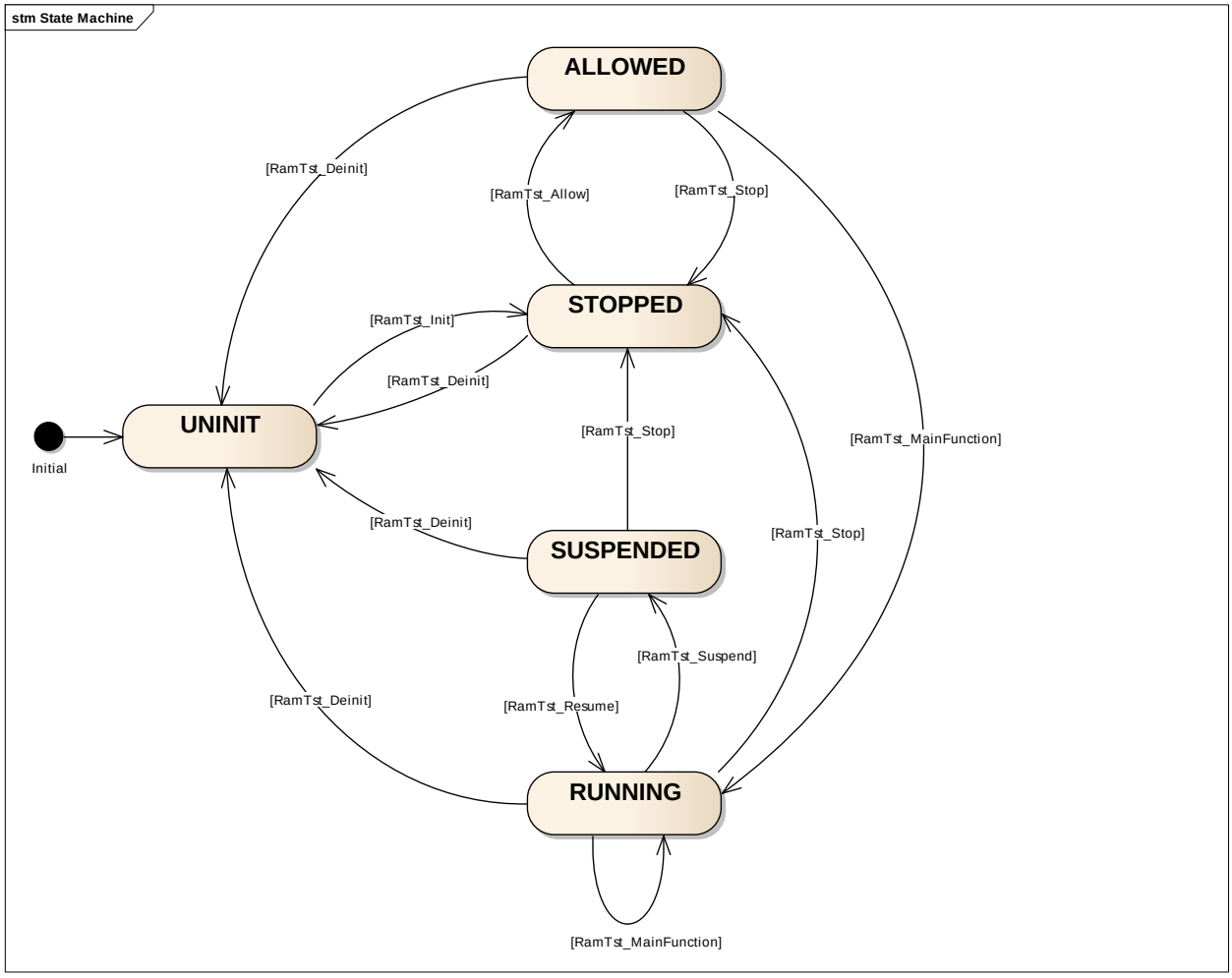

Figure 3-1 RAM Test state machine

3.4 Main Functions

3.4.1 RamTst_MainFunction

The RAM Test module implements the service `RamTst_MainFunction()` (see also chapter 5.2.18) for running a background test.

By calling this service periodically, the module will check all configured RAM blocks of the selected algorithm parameter set continuously. Each call checks the currently configured number of tested cells and iterates over the configured RAM blocks. The test procedure can be interrupted at the end of each atomic test sequence, i.e. after each main function call.

When the end of the RAM area to test has been reached, `RamTst_MainFunction()` issues the configured test end notification and repeats testing the configured blocks.

Caution
As the background testing is circular, the test restarts at the beginning of the configured RAM blocks after having tested the whole range. I.e. the background test cannot be stopped by checking the execution status for being `RAMTST_EXECUTION_STOPPED`. Stopping of the test after one or several passes can only be done in the test end notification.

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `RAMTST_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported RamTst ID is 093.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	RamTst_Init()
0x01	RamTst_DeInit()
0x02	RamTst_Stop()
0x03	RamTst_Allow()
0x04	RamTst_Suspend()
0x05	RamTst_Resume()
0x06	RamTst_GetExecutionStatus()
0x07	RamTst_GetTestResult()
0x08	RamTst_GetTestResultPerBlock()
0x09	RamTst_GetVersionInfo()
0x0A	RamTst_GetAlgParams()
0x0B	RamTst_GetTestAlgorithm()
0x0C	RamTst_GetNumberOfTestedCells()
0x0D	RamTst_SelectAlgParams()
0x0E	RamTst_ChangeNumberOfTestedCells()
0x0F	RamTst_RunFullTest()
0x10	RamTst_RunPartialTest()

Table 3-6 Service IDs

The errors reported to DET are described in the following table:

Error Code		Description
0x01	RAMTST_E_STATUS_FAILURE	A particular API service has been called in an unexpected state.
0x02	RAMTST_E_OUT_OF_RANGE	An API function parameter that is out of range has been given.
0x03	RAMTST_E_UNINIT	An API service (other than `RamTst_Init()` or `RamTst_GetVersioninfo()`) has been called before the module was initialized.
0x10	RAMTST_E_ALREADY_INITIALIZED	`RamTst_Init()` has been called while the module is already initialized.
0x11	RAMTST_E_PARAM_VINFO	`NULL_PTR` has been given to the service `RamTst_GetVersionInfo()`.

Table 3-7 Errors reported to DET

3.5.1.1 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 3-8 are internal parameter checks of the API functions.

The following table shows which parameter checks are performed on which services:

Service \ Check	Check Status Failure	Check Range	Check Uninitialized State	Check VersinInfo Pointer	
RamTst_Init			■		
RamTst_RunFullTest	■		■		
RamTst_RunPartialTest	■	■	■		
RamTst_Suspend	■		■		
RamTst_Resume	■		■		
RamTst_Stop	■		■		
RamTst_Allow	■		■		
RamTst_DeInit			■		
RamTst_GetVersionInfo				■	
RamTst_ExecutionStatus			■		
RamTst_GetTestResult			■		
RamTst_GetTestResultPerBlock		■	■		
RamTst_GetAlgParams			■		
RamTst_GetTestAlgorithm			■		
RamTst_GetNumberOfTestedCells			■		

Service \ Check	Check Status Failure	Check Range	Check Uninitialized State	Check VersinInfo Pointer	
RamTst_SelectAlgParams	■	■	■		
RamTst_ChangeNumberOfTested Cells	■	■	■		

Table 3-8 Development Error Reporting: Assignment of checks to services

3.5.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3].

If another module is used for production code error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Dem_ReportErrorStatus()`.

The errors reported to DEM are described in the following table:

Error Code	Description
RAMTST_E_RAM_FAILURE	A RAM block has been tested with the result ‘NOT OK’.

Table 3-9 Errors reported to DEM

4 Integration

This chapter gives necessary information for the integration of the MICROSAR RamTst into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the RamTst contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
RamTst.h	The module header declares the API of the RamTst. To use the module, this file has to be included.
RamTst.c	This C-source file contains the implementation of the module functionalities
SysService_AsrRamTst.jar	The jar-File contains the generator for DaVinci Configurator Pro inclusive validations rules to check if valid configuration code can be generated with the current configuration settings.
RamTst_bswmd.arxml	The description (AUTOSAR BSWMD) contains the formal notation of all configuration parameters of the RAM Test.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator.

File Name	Description
RamTst_Cfg.h	The configuration header contains the static configuration part of the component.
RamTst_Lcfg.c	The configuration-source contains the object-independent part of the configuration.
RamTst_PrivateCfg.h	The private configuration-header contains configuration for interrupt locking and error reporting.

Table 4-2 Generated files

4.2 Critical Sections

The RAM Test implements the following critical section:

> `RAMTST_EXCLUSIVE_AREA_0`: This critical section is used to protect all uninterruptable testing sequences, i.e. the whole procedure of foreground testing as well as the main function for background testing. It shall lock all interrupt sources and task switches.

4.3 Include Structure

The code file structure of the RAM Test module is structured as follows:

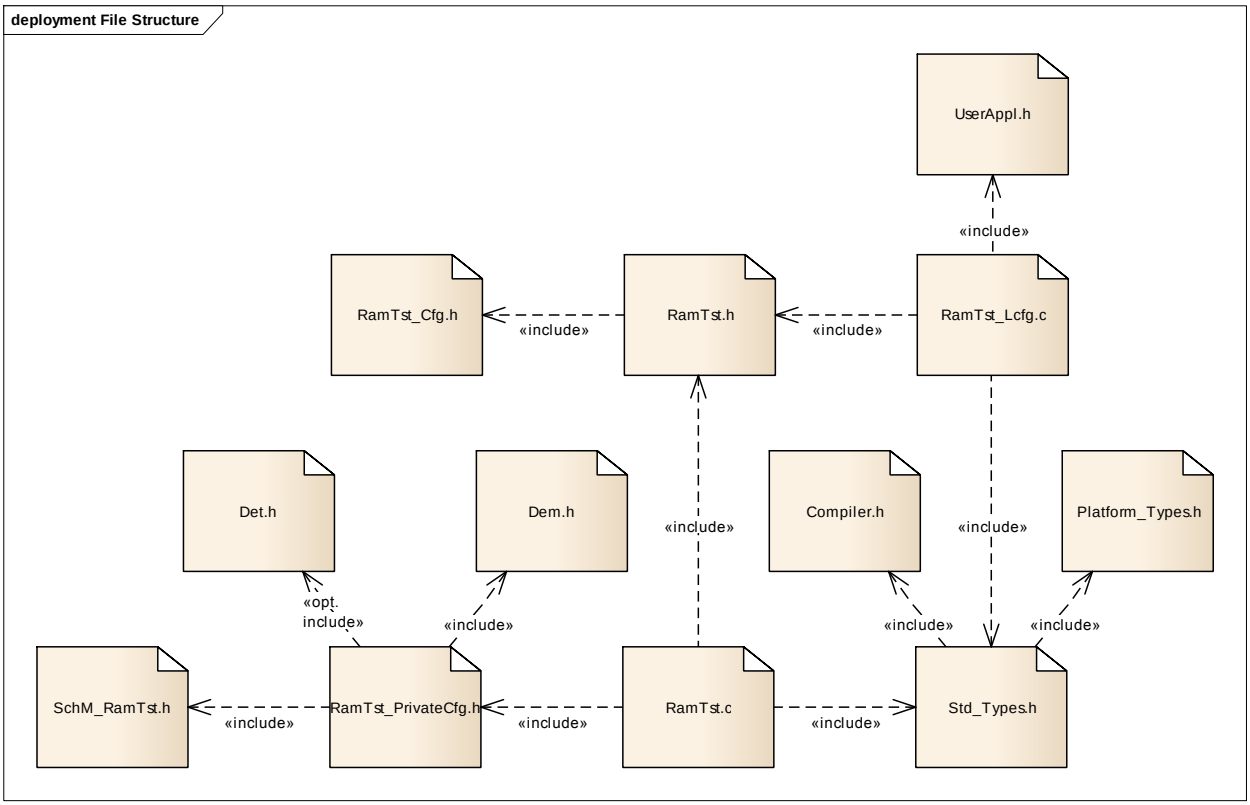

Figure 4-1 Include structure

4.4 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions of the RamTst and illustrates their assignment among each other.

Compiler Abstraction Definitions / Memory Mapping Sections	RAMTST_APPL_DATA	RAMTST_CODE	RAMTST_VAR	RAMTST_CONST	RAMTST_PBCFG
RAMTST_START_SEC_CODE RAMTST_STOP_SEC_CODE		■			
RAMTST_START_SEC_VAR_INIT_UNSPECIFIED RAMTST_STOP_SEC_VAR_INIT_UNSPECIFIED	■				
RAMTST_START_SEC_VAR_NOINIT_8BIT RAMTST_STOP_SEC_VAR_NOINIT_8BIT			■		
RAMTST_START_SEC_VAR_NOINIT_32BIT RAMTST_STOP_SEC_VAR_NOINIT_32BIT			■		
RAMTST_START_SEC_VAR_NOINIT_UNSPECIFIED RAMTST_STOP_SEC_VAR_NOINIT_UNSPECIFIED			■		
RAMTST_START_SEC_CONST_32BIT RAMTST_STOP_SEC_CONST_32BIT				■	
RAMTST_START_SEC_CONST_UNSPECIFIED RAMTST_STOP_SEC_CONST_UNSPECIFIED				■	
Points to functions or variables in the application	■				

Table 4-3 Compiler abstraction and memory mapping

4.5 Dependencies to other SW Modules

4.5.1 OSEK/AUTOSAR OS (Optional)

An operating system can be used for task scheduling and suspending and restoring interrupts globally.

4.5.2 SchM (Optional)

Beside the OSEK/AUTOSAR OS, the Basic Software Scheduler provides functions that the Ram Test can use for interrupt locking.

4.5.3 DET (Optional)

The RAM Test depends on the DET (by default) in order to report development errors. Detection and reporting of development errors can be enabled or disabled by the switches ‘Development Mode’ and ‘Development Error Reporting’ on the tab ‘General Settings’ in the configuration of the RAM Test.

4.5.4 DEM

The RAM Test reports RAM errors that have been recognized during the test to the DEM.

5 API Description

For an interfaces overview please see Figure 2-2.

5.1 Type Definitions

The types defined by the RamTst are described in this chapter.

Type Name	C-Type	Description	Value Range
RamTst_ExecutionStatusType	enum	This type describes the module state. A value of this type is returned by the API service `RamTst_GetExecutionStatus()`.	`RAMTST_EXECUTION_UNINIT` The module is not initialized. This is the default state after a reset.
			`RAMTST_EXECUTION_STOPPED` The RAM test is stopped and ready for a foreground test. This is the default state after initialization.
			`RAMTST_EXECUTION_RUNNING` A RAM test is currently running.
			`RAMTST_EXECUTION_SUSPENDED` A background test has been suspended.
			`RAMTST_EXECUTION_ALLOWED` A background test has been permitted by the API service `RamTst_Allow()`.
RamTst_TestResultType	enum	This type describes the test state of either a complete test or a single block. A value of this type is returned by the services `RamTst_GetTestResult()` and `RamTst_GetTestResultPerBlock()`.	`RAMTST_RESULT_NOT TESTED` The currently selected RAM block(s) have not been tested.
			`RAMTST_RESULT_OK` The currently selected RAM block(s) have been tested without any errors.
			`RAMTST_RESULT_NOT_OK` The currently selected RAM block(s) have been tested and a RAM error has been found in at least one block.
			`RAMTST_RESULT_UNDEFINED` Testing the currently selected RAM block(s) is in progress.
RamTst_AlgParamsIdType	uint8	This type is used to define a numeric identifier for a certain algorithm parameter set.	`0x00 … 0xFF`
RamTst_Algorithm	enum	This type describes the	`RAMTST_ALGORITHM_UNDEFINED`

Type Name	C-Type	Description	Value Range
Type		set of available test algorithms. A value of this type is returned by the API service `RamTest_GetTestAlgorithm()`.	No algorithm is defined.
			`RAMTST_CHECKERBOARD_TEST` The Checkerboard algorithm is selected.
			`RAMTST_MARCH_TEST` The March algorithm is selected.
			`RAMTST_WALK_PATH_TEST` The Walk Path algorithm is selected.
			`RAMTST_GALPAT_TEST` The Galpat algorithm is selected.
			`RAMTST_TRANSP_GALPAT_TEST` The Transparent Galpat algorithm is selected.
			`RAMTST_ABRAHAM_TEST` The Abraham algorithm is selected.
RamTst_NumberOfTestedCellsType	uint32	This type abstracts the the number of tested cells.	`0x00 … 0xFFFFFFFF`
RamTst_NumberOfBlocksType	uint16	This type is used to define a numeric identifier for a RAM block.	`0x00 … 0xFFFF`

Table 5-1 Type definitions

5.2 Services provided by RamTst

5.2.1 RamTst_InitMemory

Prototype	
`void` **`RamTst_InitMemory`** `(void)`	
Parameter	
`void`	--
Return code	
`void`	--
Functional Description	
The service `RamTst_InitMemory()` has to be called prior to the initialization function in order to initialize all global variables, which are initialized by the startup code, usually.	
Particularities and Limitations	
> This function is synchronous. > This function is non-reentrant.	

Expected Caller Context
> This service is expected to be called in application context.

Table 5-2 RamTst_InitMemory

5.2.2 RamTst_Init

Prototype	
`void` **`RamTst_Init`**`(void)`	
Parameter	
`void`	--
Return code	
`void`	--
Functional Description	
The service `RamTst_Init()` initializes the RAM Test. It initializes all relevant variables, sets the execution state to `RAMTST_EXECUTION_STOPPED` and selects the configured default algorithm parameter set (Parameter: RamTstDefaultAlgParamsId).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service is always available. > This service shall be called only once after a reset.	
Expected Caller Context	
> This service is expected to be called in application context.	

Table 5-3 RamTst_Init

5.2.3 RamTst_DeInit

Prototype	
`void` **`RamTst_DeInit`** `(void)`	
Parameter	
`void`	--
Return code	
`void`	--
Functional Description	
The service `RamTst_Deinit()` deinitializes the RAM Test. It resets all relevant variables and sets the execution status to `RAMTST_EXECUTION_DEINIT`.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service is always available.	

Expected Caller Context
> This service is expected to be called in application context.

Table 5-4 RamTst_DeInit

5.2.4 RamTst_Stop

Prototype	
`void` **`RamTst_Stop`** `(void)`	
Parameter	
`void`	--
Return code	
`void`	--
Functional Description	
The service `RamTst_Stop()` stops a running background test. After calling this service, the main function will finish the currently running atomic sequence and set the execution status to `RAMTST_EXECUTION_STOPPED`. The stopped test cannot be resumed, as the test parameters and the loop data are discarded. After calling the service `RamTst_Allow()` again, the test will start at the beginning of the configured RAM area.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter ‘Enable RamTst_Stop’ (RamTstStopApi) in the DaVinci Configurator	
Expected Caller Context	
> This service is expected to be called in application context.	

Table 5-5 RamTst_Stop

Note
This service is only relevant for background testing.

5.2.5 RamTst_Allow

Prototype	
`void` **`RamTst_Allow`** `(void)`	
Parameter	
`void`	--
Return code	
`void`	--

Functional Description
The service `RamTst_Allow()` permits the main function to perform a background test. When called, the service changes the execution state to `RAMTST_EXECUTION_ALLOWED`.
Particularities and Limitations
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter ‘Enable RamTst_Allow’ (RamTstAllowApi) in the DaVinci Configurator
Expected Caller Context
> This service is expected to be called in application context.

Table 5-6 RamTst_Allow

Note
This service is only relevant for background testing.

5.2.6 RamTst_Suspend

Prototype	
`void` **`RamTst_Suspend`** `(void)`	
Parameter	
`void`	--
Return code	
`void`	--
Functional Description	
The service `RamTst_Suspend()` suspends a currently running background test. After the service has been called, the main function will not continue the test at its next scheduled call. The execution state will be set to `RAMTST_EXECUTION_SUSPENDED`. The suspended test can be continued by calling the service `RamTst_Resume()`.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter ‘Enable RamTst_Suspend’ (RamTstSuspendApi) in the DaVinci Configurator	

Expected Caller Context
> This service is expected to be called in application context.

Table 5-7 RamTst_Suspend

Note
This service is only relevant for background testing.

5.2.7 RamTst_Resume

Prototype	
`void` **`RamTst_Resume`** `(void)`	
Parameter	
`void`	--
Return code	
`void`	--
Functional Description	
The service `RamTst_Resume()` reactivates a suspended test run. After this service has been called, the main function will continue testing at its next scheduled call. The test will be continued after the address that has been checked during the last scheduled call of the main function right before suspension. The execution state will be set to `RAMTST_EXECUTION_RUNNING`.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter ‘Enable RamTst_Resume’ (RamTstResumeApi) in the DaVinci Configurator	
Expected Caller Context	
> This service is expected to be called in application context.	

Table 5-8 RamTst_Resume

Note
This service is only relevant for background testing.

5.2.8 RamTst_GetExecutionStatus

Prototype
`RamTst_ExecutionStatusType` **`RamTst_GetExecutionStatus`** `(void)`

Parameter	
`void`	--
Return code	
`RamTst_ExecutionStatusType`	Current execution status
Functional Description	
The service `RamTst_GetExecutionStatus()` returns the current RAM Test execution status. The returned value is influenced by the following services: > `RamTst_Stop()` > `RamTst_Allow()` > `RamTst_Suspend()` > `RamTst_Resume()` (see also chapter 3.3)	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter ‘Enable RamTst_GetExecutionStatus’ (RamTstGetExecutionStatusApi) in the DaVinci Configurator	
Expected Caller Context	
> This service is expected to be called in application context.	

Table 5-9 RamTst_GetExecutionStatus

5.2.9 RamTst_GetTestResult

Prototype	
`RamTst_TestResultType` **`RamTst_GetTestResult`** `(void)`	
Parameter	
`void`	--
Return code	
`RamTst_TestResultType`	Current test result
Functional Description	
The service `RamTst_GetTestResult()` returns the current overall RAM test result.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter ‘Enable RamTst_GetTestResult’ (RamTstGetTestResultApi) in the DaVinci Configurator	

Expected Caller Context
> This service is expected to be called in application context.

Table 5-10 RamTst_GetTestResult

5.2.10 RamTst_GetTestResultPerBlock

Prototype	
`RamTst_TestResultType` **`RamTst_GetTestResultPerBlock`** `(RamTst_NumberOfBlocksType BlockID)`	
Parameter	
`BlockID`	Numeric identifier or symbolic name of the desired block
Return code	
`RamTst_TestResultType`	Current test result for the given block
Functional Description	
The service `RamTst_GetTestResultPerBlock()` returns the current test result for the block given in the parameter `BlockID`.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter ‘Enable RamTst_GetTestResultPerBlock’ (RamTstGetTestResultPerBlockApi) in the DaVinci Configurator	
Expected Caller Context	
> This service is expected to be called in application context.	

Table 5-11 RamTst_GetTestResultPerBlock

5.2.11 RamTst_GetAlgParams

Prototype	
`RamTst_AlgParamsIdType` **`RamTst_GetGetAlgparams`** `(void)`	
Parameter	
`void`	--
Return code	
`RamTst_AlgParamsIdType`	Numeric identifier of the currently selected algorithm parameter set
Functional Description	
The service `RamTst_GetAlgparams()` returns the currently selected algorithm parameter set. This value is either the default algorithm parameter set given in the configuration parameter ‘Default Algorithm Parameter Set’ (RamTstDefaultAlgParams) or a value that has been selected by using the service `RamTst_SelectAlgParams()`.	

Particularities and Limitations
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter ‘Enable RamTst_GetAlgParams’ (RamTstGetAlgParamsApi) in the DaVinci Configurator
Expected Caller Context
> This service is expected to be called in application context.

Table 5-12 RamTst_GetAlgParams

5.2.12 RamTst_GetTestAlgorithm

Prototype	
`RamTst_AlgorithmType` **`RamTst_GetTestAlgorithm`** `(void)`	
Parameter	
`void`	--
Return code	
`RamTst_AlgorithmType`	Active test algorithm
Functional Description	
The service `RamTst_GetTestAlgorithm()` returns the test algorithm, the currently selected algorithm parameter set is using.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter ‘Enable RamTst_GetTestAlgorithm’ (RamTstGetTestAlgorithmApi) in the DaVinci Configurator	
Expected Caller Context	
> This service is expected to be called in application context.	

Table 5-13 RamTst_GetTestAlgorithm

5.2.13 RamTst_GetNumberOfTestedCells

Prototype	
`RamTst_NumberOfTestedCellsType` **`RamTst_GetNumberOfTestedCells`** `(void)`	
Parameter	
`void`	--
Return code	
`RamTst_NumberOfTestedCellsType`	Number of tested cells

Functional Description
The service `RamTst_GetNumberOfTestedCells()` returns the currently configured number of cells to test per main function call. The number of cells to test is either the value given by the configuration parameter 'Number Of Tested Cells' (RamTstNumberOfTestedCells) or a value that has been set by using the service `RamTst_ChangeNumberOfTestedCells()`.
Particularities and Limitations
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter 'Enable RamTst_GetNumberOfTestedCells' (RamTstGetNumberOfTestedCellsApi) in the DaVinci Configurator
Expected Caller Context
> This service is expected to be called in application context.

Table 5-14 RamTst_GetNumberOfTestedCells

5.2.14 RamTst_SelectAlgParams

Prototype	
`void` **`RamTst_SelectAlgParams`** `(RamTst_AlgParamsIdType NewAlgParamsId)`	
Parameter	
`NewAlgParamsId`	Numeric identifier or symbolic name of the algorithm parameter set to switch to
Return code	
`void`	--
Functional Description	
The service `RamTst_SelectAlgParams()` selects a new algorithm parameter set for the upcoming test run. Depending on the configuration, this service may select a new test algorithm, a new set of blocks to test or both. This service deletes the test result of the last executed test and sets the overall- and per-block result to `RAMTST_RESULT_NOT_TESTED`. The module has to be in the execution state `RAMTST_EXECUTION_STOPPED`, when this function is called.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter 'Enable RamTst_SelectAlgParams' (RamTstSelectAlgParamsApi) in the DaVinci Configurator	
Expected Caller context	
> This service is expected to be called in application context.	

Table 5-15 RamTst_SelectAlgParams

5.2.15 RamTst_ChangeNumberOfTestedCells

Prototype	
`void` **`RamTst_ChangeNumberOfTestedCells`** `(RamTst_NumberOfTestedCellsType NewNumberOfTestedCells)`	
Parameter	
`NewNumberOfTestedCells`	New number of cells to test in each main function call at the next test run
Return code	
`void`	--
Functional Description	
The service `RamTst_ChangeNumberOfTestedCells()` adjusts the number of tested cells for each scheduled main function call for the upcoming test run. The module has to be in the execution state `RAMTST_EXECUTION_STOPPED`, when this function is called.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter ‘Enable RamTst_ChangeNumberOfTestedCells’ (RamTstChangeNumberOfTestedCellsApi) in the DaVinci Configurator	
Expected Caller Context	
> This service is expected to be called in application context.	

Table 5-16 RamTst_ChangeNumberOfTestedCells

5.2.16 RamTst_RunFullTest

Prototype	
`void` **`RamTst_RunFullTest`** `(void)`	
Parameter	
`void`	--
Return code	
`void`	--
Functional Description	
The service `RamTst_RunFullTest()` performs a foreground test on all the RAM blocks defined in the currently selected algorithm parameter set. During the test run, the execution state will be `RAMTST_EXECUTION_RUNNING`, after the test is finished, the state will be set back to `RAMTST_EXECUTION_STOPPED`. Afterwards, the overall test result as well as the test results for the single blocks can be retrieved by the services `RamTst_GetTestResult()` and `RamTst_GetTestResultPerBlock()`.	

Particularities and Limitations
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter ‘Enable RamTst_RunFullTest’ (RamTstRunFullTestApi) in the DaVinci Configurator
Expected Caller Context
> This service is expected to be called in application context.

Table 5-17 RamTst_RunFullTest

5.2.17 RamTst_RunPartialTest

Prototype	
`void` **`RamTst_RunPartialtest`** `(RamTst_NumberOfBlocksType BlockId)`	
Parameter	
`BlockId`	Numeric identifier or symbolic name of the block to test
Return code	
`void`	--
Functional Description	
The service `RamTst_RunPartialTest()` performs a foreground test on the RAM block `BlockId` defined in the currently selected algorithm parameter set. During the test run, the execution state will be `RAMTST_EXECUTION_RUNNING`, after the test is finished, the state will be set back to `RAMTST_EXECUTION_STOPPED`. Afterwards, the overall test or the test result for tested block (which will be the same) can be retrieved by the services `RamTst_GetTestResult()` and `RamTst_GetTestResultPerBlock()`.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter ‘Enable RamTst_RunPartialTest’ (RamTstRunPartialTestApi) in the DaVinci Configurator	
Expected Caller Context	
> This service is expected to be called in application context.	

Table 5-18 RamTst_RunPartialTest

5.2.18 RamTst_MainFunction

Prototype	
`void` **`RamTst_MainFunction`** `(void)`	
Parameter	
`void`	--
Return code	
`void`	--

Functional Description
The service `RamTst_MainFunction()` performs a background test on all the RAM blocks defined in the currently selected algorithm parameter set. Therefore, the service has to be called cyclically. Each time the service is called, it tests the currently configured number of tested cells, which can either be the value given by the configuration parameter 'Number Of Tested Cells' (RamTstNumberOfTestedCells) or a value that has been set by using the service `RamTst_ChangeNumberOfTestedCells()`. After all blocks defined in the currently selected algorithm parameter set are tested, the test restarts at the beginning of the blocks to test. To stop the test after one or several test runs are complete, the test completed notification can be used. This notification is called after every test run and it can be configured by the parameter 'Test Complete Notification' (RamTstTestCompletedNotification). The test can be stopped by calling `RamTst_Stop()` in this notification. Before the service executions any testing, the module execution state has to be switched to `RAMTST_EXECUTION_ALLOW` by calling the service `RamTst_Allow()`. During the test run, the execution state will be `RAMTST_EXECUTION_RUNNING`, even in the `RamTst_MainFunction()` is not active at this moment.
Particularities and Limitations
> This function is synchronous. > This function is non-reentrant. > This service is always available.
Expected Caller Context
> This service is expected to be called in application context.

Table 5-19 RamTst_MainFunction

5.2.19 RamTst_GetVersionInfo

Prototype	
`void` **`RamTst_GetVersionInfo`** `(Std_VersionInfoType* versioninfo)`	
Parameter	
`versioninfo`	description
Return code	
`void`	--
Functional Description	
The service `RamTst_GetVersionInfo()` returns the version of this module. The version information includes: > Module Id > Vendor Id > Instance Id > Vendor specific version numbers	
Particularities and Limitations	
> Service ID: see table 'Service IDs' (chapter 3.5.1) > This function is synchronous. > This function is non-reentrant. > This service can be enabled or disabled by the parameter 'Enable RamTst_GetVersionInfo' (RamTstGetVersionInfoApi) in the DaVinci Configurator	

Expected Caller Context
> This service is expected to be called in application context.

Table 5-20 RamTst_GetVersionInfo

5.3 Services used by RamTst

In the following table services provided by other components, which are used by the RamTst are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET	Det_ReportError
DEM	Dem_ReportErrorStatus

Table 5-21 Services used by the RamTst

5.4 Configurable Interfaces

5.4.1 Notifications

At its configurable interfaces the RamTst defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the RamTst but can be performed at configuration time. The function prototypes that can be used for the configuration have to match the appropriate function prototype signatures, which are described in the following sub-chapters.

5.4.1.1 RamTst_TestCompletedNotification

Prototype	
`void` **`RamTst_TestCompletedNotification`** `(void)`	
Parameter	
`void`	--
Return code	
`void`	--
Functional Description	
This notification is called every time all configured RAM blocks of the currently selected algorithm parameter set have been tested in a background test. The function to be called for this notification is configurable by the parameter ‘Test Completed Notification’ (RamTstTestCompletedNotification). The function name given above is the default value.	
Particularities and Limitations	
> None	
Call context	
> This notification is called in task context.	

Table 5-22 RamTst_TestCompletedNotification

5.4.1.2 RamTst_ErrorNotification

Prototype	
`void` **`RamTst_ErrorNotification`** `(void)`	
Parameter	
`void`	--
Return code	
`void`	--
Functional Description	
This notification is called every time a RAM error is recognized in a background test. The function to be called for this notification is configurable by the parameter ‘RAM Error Notification’ (RamTstTestErrorNotification). The function name given above is the default value.	
Particularities and Limitations	
> None	
Call context	
> This notification is called in task context.	

Table 5-23 RamTst_ErrorNotification

6 Configuration

6.1 Configuration Variants

The RamTst supports the configuration variants

- VARIANT-PRE-COMPILE
- VARIANT-LINK-TIME

The configuration classes of the RamTst parameters depend on the supported configuration variants. For their definitions please see the RamTst_bswmd.arxml file.

6.2 Configuration with DaVinci Configurator 5

The RamTst is configured with the help of the configuration tool DaVinci Configurator 5 (CFG5). The definition of each parameter is given in the corresponding BSWMD file.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
Algorithm parameter set	A test configuration set that defines parameters for a single test run, e.g. the used test algorithm and the RAM blocks to test
DaVinci Configurator	Configuration and generation tool for MICROSAR components

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
RTE	Runtime Environment
SWC	Software Component
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com